Universiteit Leiden
The Netherlands

Opleiding Informatica

Finding points of interest using GPS data

Eric Manintveld

Supervisors:
Matthijs van Leeuwen & Daniela Gawehns

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

12/04/2022

Abstract

The aim of this research is to find an algorithm that detects points of interest (POIs) from small-scale GPS data. Small-scale GPS data is GPS data collected in a small area. The data used in this research is simulated data that aims to represent GPS data recorded in real life. The simulation is based of the Vossenberg Park in Kaatsheuvel. The simulation allows us the generate a large dataset. The Vosssenberg Park belongs to the adjacent retirement home. Therefore it is often used by elderly with dementia. The detected points of interest can be used when designing a park that is used by these people. The park is relatively small scale for GPS research. This makes it more difficult since GPS data is not always perfectly accurate. The data will most likely contain noise. To stay as close as possible to reality, this noise is also simulated.

We detect POIs using staypoints. Staypoints are places where the holder of a GPS tracker is located for an extended period of time. These staypoints can be seen as POIs for an individual GPS trace. After combining staypoints from multiple traces, we will obtain the POIs we are looking for. We will compare the C-DBSCAN algorithm to a baseline algorithm using three performance indicators. Firstly, the average percentage of correctly found POIs. This is the percentage of the ground truth POIs that the algorithm found. Secondly, the average deviation of the correctly detected POIs to the ground truth POIs. This is measured in meters and tells us about the accuracy of the algorithm. Lastly, the average number of false positives. A false positive is a detected point that does not exist in the ground truth. The experiments suggest that the C-DBSCAN algoritm finds more actual POIs at the cost of producing more false positives. Therefore the use of either of the two methods depends on the situation.

Contents

1 Introduction

This thesis focuses on a park called Vossenberg Park. See Figure 1 for an overview of the park. This park is mainly used by elderly with dementia from the adjacent care home. The goal is to find out how these people use this park. By studying the behaviour of the elderly we can obtain insight in how to design parks for these people in the future. We will be looking at points of interest (POIs) specifically. These POIs are hotspots, meaning they are often visited by different people. They show what areas of the park the elderly use and what parts they do not use. It might be that some benches are used frequently, while others are never used. The frequently used bench will be a POI in this case. This gives insight in what the optimal position for a bench is. For example, not too far from the entrance of the park or vice versa. All of this information can help to design dementia-friendly parks in the future.

We will be looking at point of interest (POI) extraction from small-scale GPS data. Small-scale GPS data is collected within a small area. We define a small area to be smaller than a city block or a neighborhood. This small scale is a challenge since GPS data is not always very accurate. GPS data can contain a lot of noise. We want to find an algorithm that can be used to find points of interest in the Vossenberg park in Kaatsheuvel. This algorithm should be able to handle the noise that comes with GPS data.

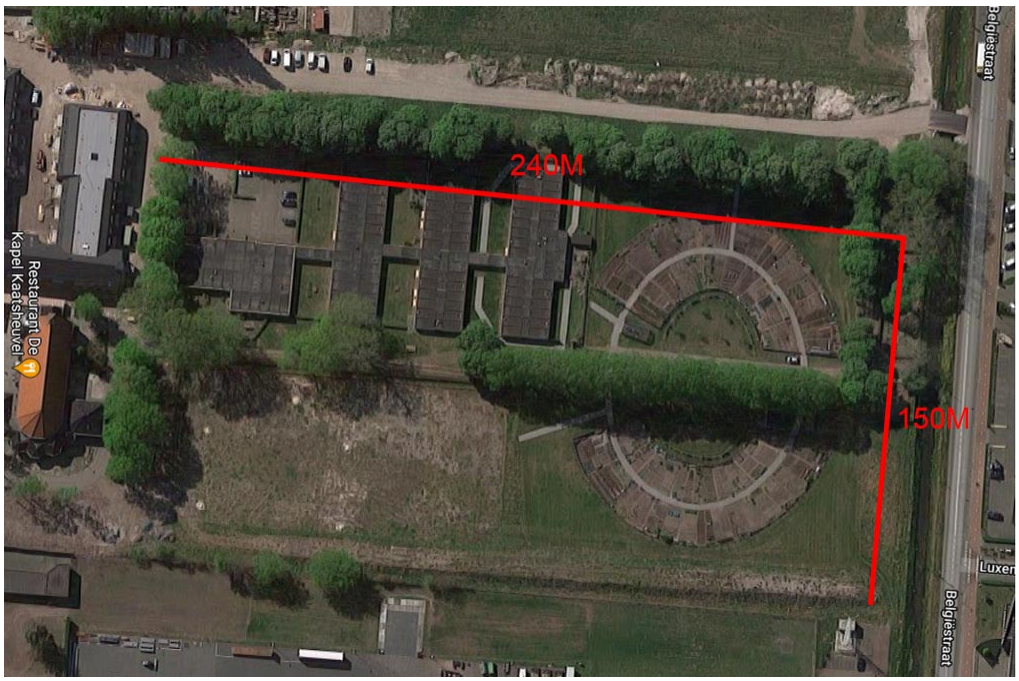


Figure 1: Park Vossenberg

The process of finding POIs consists of two main steps. First we need to extract staypoints from the individual walkers. These staypoints can be seen as POIs for a single walker. After this we create clusters from these staypoints. The resulting clusters are the detected POIs.

This means we are dealing with a clustering problem. We want to find a place that multiple people visit. Clustering is unsupervised, meaning it does not train on a dataset with labels. A popular clustering algorithm that can handle noise is the "Density-based spatial clustering of applications with noise" algorithm. This algorithm was presented by Martin Ester et al. in 1996 [EKS+96]. We will be using an extension of this algorithm called C-DBSCAN as introduced by Gong et al. [GSY+15]. This extension also takes the direction change and time sequence of the data into consideration. These extension are designed to improve the DBSCAN algorithm for finding POIs. Although DBSCAN is an older algorithm, it is still considered very useful [SSE+17]. We will compare the C-DBSCAN algorithm with our own baseline algorithm. This baseline algorithm looks at the distances between measurements. When the distance between two measurements, that are for example 30 seconds apart, is smaller than a certain threshold, the algorithm will mark these measurements and the measurements in between as a staypoint.

For our research we will use simulated data. This data represents visitors moving through a park with predetermined POIs. We will call these simulated visitors "walkers". The size of the simulated park is based of the Vossenberg park. We use simulated data to obtain a larger sample size than we would be able to obtain using real data. This large sample size gives us more confidence in our findings. The simulated walker starts at the park entrance and walks towards a random selection of POIs. When the walker is in range of a POI it stays in this range for a random number of seconds. After this it moves on to the next POI. This continues until the walker has visited all the POIs. After visiting the last POI the walker moves towards the exit of the park.

Both methods will be compared based on three performance indicators. The first indicator is the percentage of correctly found POIs. This indicator shows the percentage of POIs that the method found out of all the generated POIs. The second indicator is the average deviation from the actual POI. This indicator shows the average distance from the detected the POI to the actual POI. The third and final indicator is the number of false positives. This shows the average number of falsely detected POIs. Apart from these performance indicators we will also look at the Davies-Bouldin Index. This index rates the produced clusters based on spacing and density. It does not take any context into consideration. Therefore it will only tell us about the shape of a cluster. High scoring clusters can be an indication of a good clustering method. We expect that clusters are densely formed around POIs.

Real GPS trackers produce noise. This noise is replicated in the simulation. After the simulation is done a noise pass is applied to the data. All data points are shifted in a random direction by up to five meters. Research by van Diggelen and Enge shows that the GPS of a smartphone under an open sky is typically accurate within a 4.9 meter radius [vDE15]. We assume that the trackers used in real data collection are similar to the GPS systems found in smartphones. We therefore think this random shift up to 5 meters replicates real noise closely.

1.1 Research questions

The main challenge in this research is the small scale of the Vossenberg park. This makes it harder to work with GPS data inside the park, since GPS data will always suffer from inaccuracy. Using algorithms that consider noise we want to see if it is still possible to find POIs in this small scale

data. Therefore we have to following research question:

- What algorithm can be used for finding POIs from GPS data according to previous research?
- Is this algorithm suitable for detecting POIs from the GPS data collected in Kaatsheuvel?

We will answer the first question by looking at earlier research and literature. Once we have found an algorithm we will compare it to a baseline algorithm. If the found algorithm outperforms the baseline it is suited for the small scale data collected in Kaatsheuvel.

1.2 Thesis overview

We will shortly discuss the contents of this thesis. This chapter contains the introduction to the problem and experiments and the research questions. In the next chapter, Chapter 2, we look at background information regarding our topic of detecting POIs. We explain definitions used throughout the paper and look at related research. Chapter 3 explains the methods used to compare the two algorithms. In this chapter we also look at the results of the experiments. The final chapter is Chapter 4. In this chapter we answer the research questions defined in this introduction.

This bachelor thesis is made under supervision of Matthijs van Leeuwen and Daniela Gawehns for my bachelor course at LIACS.

2 Background

2.1 Definitions

Staypoint
Existing research regarding POI mining uses staypoints to detect points of interest (Khetarpaul, Subramaniam & Nambiar, 2011). When a tracker is in a certain area for a certain amount of time, this place is marked as staypoint. The size of the area and the amount of time the tracker needs to stay in this area can be adjusted. A staypoint can be seen as a POI for a single GPS trace. The exact definition of a staypoint depends on the staypoint detection method.

Point of Interest
A point of interest (POI) represents a place in the real world that is often visited by different people. In this research we consider a POI to be a cluster of staypoints. The size of this cluster can be adjusted by tweaking the (hyper)parameters of the used algorithms. Therefore the exact definition of a POI depends on the algorithm used.

Simulated park
A simulated park, or park, is the representation of a real world park that is used in the simulation of the data. The size of the park is consistent between the different parks. The simulated park is 240 meters long and has a width of 150 meters. For simplicity reasons, the park is rectangular. A park contains POIs. The number of POIs that a park contains is random. The minimum number of POIs in a park is 1 and the maximum is 7. In other words, a park P is a set of POIs, where $1 \leq |P| \leq 7$.

Simulated walker
A simulated walker represents a visitor moving through a park. The simulated walker starts at the entrance of the park and will move towards POIs. The POIs that a walker will visit are randomly selected before the simulation starts. Every second a measurement is taken. The walker will always stay within the boundaries of the park. When the walker has visited all the POIs it needed to visit, it will start moving towards the exit of the park. When the walker reaches the exit, the measurements are stopped. Suppose P is the set of POIs in a particular park, and $R \subseteq P$. Then a simulated walker is a path between the elements of R.

2.2 Related Work

2.2.1 Staypoints

Existing research uses staypoints to detect POIs [KCG+11]. The aim of this research was to analyze aggregate GPS information of multiple users to mine a list of interesting locations and rank them. Interesting locations are geographical locations visited by several users. For example, an office, university, historical place, restaurant, shopping complex, stadium, etc. Various relational algebra and statistical operations were applied to the GPS data of multiple users. The information extracted from the GPS data can be used for planning billboard locations, traffic planning and various other city planning related tasks. The algorithm used is similar to our baseline method. It also uses the distance between the measurements along with the time between these measurements. We will discuss our baseline method later.

2.2.2 DBSCAN

Throughout this paper we will use the DBSCAN algorithm. DBSCAN is a clustering algorithm. It was presented by Martin Ester et al in their paper called "A density-based algorithm for discovering clusters in large spatial databases with noise" [EKS+96]. It was created to solve the following requirements for clustering algorithms: minimal requirements of domain knowledge to determine the input parameters, discovery of clusters with arbitrary shape and good efficiency on large databases. The well-known clustering algorithms at the time did not offer a solution to the combination of these requirements.
The paper called "DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation" [GT15] from 2015 by Gan and Tao addresses a mis-claim from the original DBSCAN paper. In 2017 Erich Schubert et al. [SSE+17] pointed out some innaccuracies in the way DBSCAN was represented in the 2015 paper. This 2017 paper also discusses why and how DBSCAN should still be used. So even though DBSCAN is an older algorithm, it is still relevant. We explain how the DBSCAN algorithm works in Section 3.3.1.

2.2.3 C-DBSCAN

C-DBSCAN is an extension of the DBSCAN algorithm. It introduces two constraints over the original DBSCAN algorithm. Firstly, a time sequence constraint. This means that the algorithm will take the time sequence of the GPS measurements into consideration. Secondly, a direction change constraint. This constraint looks at the average direction change of the GPS measurements.

It assumes this average is different within the range of a POI compared to outside this range. We use the implementation as discussed by Gong et al in the paper "Identification of activity stop locations in GPS trajectories by density-based clustering method combined with support vector machines" [GSY+15]. The GPS data they collected for their research was of a way larger scale. The data was recorded in the Nagoya area in Japan. The city of Nagoya has a size of 326,43 km^2 [Wb21]. This is significantly larger than the Vossenberg park.

3 Methods & Results

We will detect staypoints using two different methods. The first method is the baseline method. The second method uses the C-DBSCAN algorithm. These two methods will both generate staypoints. These staypoints need to be clustered into POIs. For both methods we will use the DBSCAN clustering algorithm to cluster the staypoints into POIs. After this we can compare both methods with the ground truth from the simulated data.

3.1 Data simulation

Without a ground truth, measuring performance is difficult. Therefore we will use simulated data. With this simulated data we have a base truth. We know the number and location of the different POIs beforehand. This way we have a form of labels in an otherwise unsupervised task.
The simulated data consists of multiple 'parks' with POIs located in them. A 'park' has a collection of simulated walkers. The size of the different parks will be the same. The number of POIs is different between the parks, with a maximum of seven POIs and a minumum of one. We will not consider parks with more than seven POIs in this research, since we do not expect parks of this size to have more than seven POIs in reality. After a visual inspection of the Vossenberg park, we believe that seven POIs is realistically the maximum number of POIs in a park of this size. We want to find an algorithm that works well whether a park contains a small or large number of POIs. In the simulation a POI is represented by a center point with a 5 meter radius around it. If the simulated walker is inside this radius, we consider the walker to be visiting the POI. The location of the POIs is also random, except for the restriction that POIs cannot overlap. Since if two POIs would overlap in real life, we would consider this to be a single POI. The parks will be around the same size as the Vossenberg park (240 meter in height and 150 meters in width). We keep all parks the same size to more easily compare the methods. Since the main aim is to find POIs in the Vossenberg park, we use the scale of this park.

3.1.1 Assumptions

Before we start generating the data we need to do some assumptions. Firstly we assume that POIs are stationary. We also assume that the simulated park has no obstacles, no paths and is flat. This means there is no elevation change. We make this assumption because there might be cases where the altitude can have an effect on the detection of POIs. If a park has a spiral staircase for example. If we look at the latitude and longitude of someone using these stairs, it might look like they are staying in roughly the same position. This might cause us to think that this is a POI, when in reality this is not the case. Since most public parks in The Netherlands are relatively flat, like for example the Vossenberg park, we make the assumption that the park is flat in our data simulation.

3.1.2 How does it work?

The first step is to set a simulated park. This park contains a random number of POIs between one and seven. The POIs are randomly located throughout the park. The size of the park has been set before running the algorithm. This park will be used to generate a set number of walkers. These walkers will visit POIs. The POIs that a route visits are randomly decided, since a real person will not always visit all the available POIs. They might only visit some POIs, while they are not interested by others. The amount of time the walker stays within the range of a POI is random between thirty seconds and five minutes. A walker is considered to be in range of a POI, when it is within a five meter radius of the center of the POI. The walker starts at a predefined entrance and leaves at a predefined exit. The predefined entry and exit are the same for each park.
When the algorithm starts the walker is at the entrance. The algorithm will now calculate the exact angle in which the walker needs to travel to reach the POI. A random angle between -90 and 90 degrees is calculated and added to the calculated angle. This way the walker move roughly in the direction of the POI, but will not travel straight to it. In reality a person will most likely not travel to their next destination in a perfectly straight line either. With every new measurement, the walker travels 1.1 meters. If we consider that every measurement represents a second, the walker moves at a speed of 4 km/h. This process is repeated until the walker eventually reaches a POI. At this point the algorithm will notice this. The walker will now stay within the range of the POI for 30 seconds to five minutes.
This repeats itself until the walker has reached the exit. The walker moves to the exit in the same way as it walks towards POIs. Once the walker has reached the exit the route is completed and saved in .csv format.

3.1.3 Pseudo-code

Algorithm 1 shows the pseudo-code for the data simulation algorithm. Where *POIs_on_route* is a list of POIs the simulated walker will visit. This also includes the exit of the park. *POI* is the POI the walker is currently traveling to. The function *getAngle* returns the angle from the walker to the given position. The *addAngle* function adds a random angle to the given angle. To get this random angle we first randomize a number between $-\pi$ and π. This number will be x in the following formula: $y = x^3 * \frac{1}{\pi^2}$. This formula is plotted in Figure 2. The resulting value of y will be added to the angle calculated by the *getAngle* function. In most cases y will be less than the randomized value of x. This means the simulated walker is biased to walk in the direction of the next POI. The *move* function moves the simulated walker along the calculated angle by updating the current location. The start location and every location update are measurements in the dataset.

Algorithm 1 Data simulation algorithm:

```
currentLocation = entrance

while num_POIs_visited < POIs_on_route do

    POI = getNextPOI(POIs_on_route)

    while getDistance(POI, currentLocation) > 5 do
        angle = getAngle(currentLocation, POI)
        angle = addAngle(angle)
        while Next step is not within park boundaries do
            Add different value to angle
        move(angle)

    // Walker is now in range of the POI

    while currentTime < POI_duration do
        angle = getAngle()
        angle = addAngle(angle)
        while Next step is not within park boundaries do
            Add different value to angle
        move(angle)
```

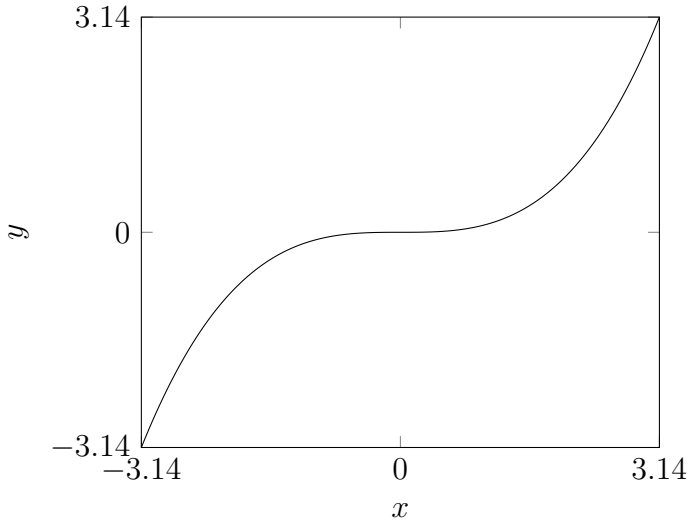


Figure 2: Plot of formula: $y = x^3 * \frac{1}{\pi^2}$

3.1.4 How much data will we use

Since we work with simulated data we can generate as much data as we want. But we need to set a limit to the amount of data we generate. We can not keep generating data forever. The size of

our total dataset is determined by two parameters. Firstly, the number of parks we generate. And secondly, the number of routes that each park contains.
To determine the number of walkers, or in other words: the number of people that visit the park, we need to specify the time period. If we specify this period to be one day, we would, based on our observation in the Vossenberg park, have around 150 daily visitors.
We will simulate 150 parks. This number ensures we have a large enough sample size to use statistical tests. Figure 3 shows the number of parks per group. Each group consists of all the parks with the same number of POIs.

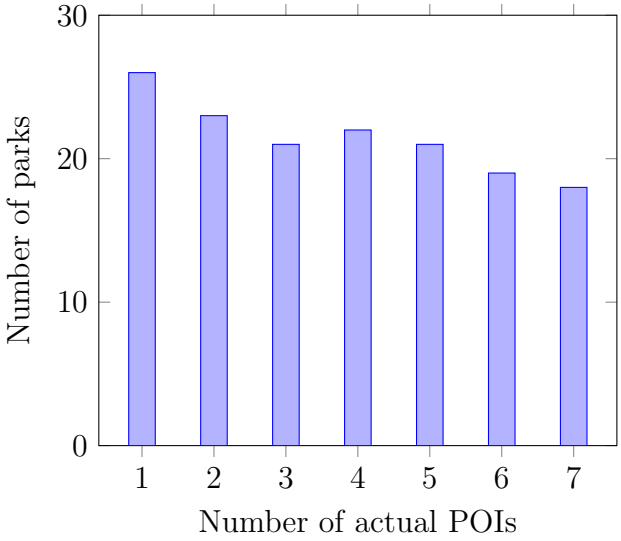


Figure 3: Number of parks per number of actual POIs.

3.1.5 Noise

To better represent real GPS data, we will add artificial noise. This is done by slightly shifting every point. We shift every point by a random distance between zero and five meters. We chose 5 meters, because a GPS systems are usually accurate within 5 meters [vDE15].

3.2 Performance indicators

Average deviation from ground truth POI
We use three performance indicators. The first one is the average deviation from the ground truth POI. In the simulation we assumed a POI to be a centre point with a radius of 5 meters around it. We can compare how far the centre point of the detected POI is off from the ground truth POI. With this performance indicator we are only looking at the predicted POIs that have detected a ground truth POI. This means the predicted POI is inside the 5 meter radius of a ground truth POI. If we were to look at all the predictions, there is no clear way to determine what predictions belong to what ground truth POIs. We might have more predictions than POIs or vice versa. We can note this performance indicator as shown below:

$$\text{Average deviation from ground truth POI} = \frac{\sum_{i=0}^{n} D_n}{n}$$

Where D is the distance between the centers of the predicted and ground truth POIs. The number of detected POIs is n.

Number of correctly detected POIs
The second indicator is the number of correctly detected POIs. This indicator shows the number of POIs that have been predicted correctly. A ground truth POI has been correctly found, if the center of a predicted POI lies within five meters of the center of this ground truth POI. So $D < 5$, where D is the distance between the centers of the predicted and ground truth POIs. A ground truth POI can also only be detected by one predicted POI. If there are two predicted POIs within the range of a ground truth POI, one of the predicted POIs will be labeled as a false positive.

$$CD = \{p \in P \mid D_g < 5\}$$
$$\text{Number of correctly detected POIs} = |CD|$$

Where CD is the set of correctly detected POIs. D_g is the distance between a ground truth POI and the nearest predicted POI. And P is the set of predicted POIs.

False positives
The number of false positives is the last performance indicator. These are all the predicted POIs that are not within a 5 meter radius of an undiscovered ground truth POI.

$$FP = \{p \in P \mid D_o > 5\}$$
$$\text{Number of false positives} = |FP|$$

Where FP is the set of false positives. D_o is the distance between the center of the predicted POI and the nearest open ground truth POI. An open ground truth POI is a ground truth POI, that has not been detected yet by a predicted POI. P is the set of predicted POIs.

3.3 Clustering staypoints with DBSCAN

First we extract the staypoints from the individual walkers. Then we want to cluster these staypoints to find POIs. The formed clusters are the POIs. To cluster the staypoints we use the DBSCAN algorithm.

3.3.1 How does it work?

The DBSCAN is a clustering algorithm. DBSCAN is short for 'Density-based spatial clustering of applications with noise'. It is a non-parametric algorithm. It groups together points that are close to each other. Points that are not near other points are marked as outliers. [Wik21a] The algorithm works by counting the number of points that surround a point. The size of the radius in which points are counted can be adjusted by setting a threshold value. A second threshold value determines how many points need to be in the radius for a point to be marked as part of a cluster. The DBSCAN algorithm does not require the user to specify the number of clusters beforehand. As the name implies, the algorithm is designed to deal with noise. [BK07] Since our data contains

artificial noise, this algorithm is a good fit.

We will be using the DBSCAN algorithm to cluster the detected staypoints into POIs for both staypoint detection methods. The functionality of this part of the algorithm will be the same regardless of the staypoint detection method used. This ensures a fair comparison between the two methods.

3.3.2 DBSCAN hyperparameters

The DBSCAN algorithm requires the user to set two threshold values. To ensure a fair comparison between the two methods, we used the same threshold values in both tests. We used the Scikit Learn implementation of the DBSCAN algorithm [sci]. In the table below you can find the threshold values that were used in our experiments.

eps	0.0000172
min_samples	5
metric	haversine

Table 1: Threshold values for Scikit Learn DBSCAN algorithm.

Eps is the maximum distance between two samples for one to be considered as in the neighborhood of the other. The average human walking speed is about 5 km/h [Wik21c]. Therefore the value we have chosen equals about four meters. This means that the measurements taken when someone is walking from POI to POI, will most likely not be considered. This is desirable, since we want to find the POIs and are not interested in the routes in between.
Min_samples is the number of samples in a neighborhood for a point to considered as a core point. A core point is a point that has at least *min_pts* within a distance of *eps* of itself [Wik21a]. The *min_points* threshold also includes the point itself. This means in our case there should be at least four other points within 4 meters, to label a point as a core point. This is the default value of the sci-kit learn implementation of the algorithm [sci]. More experimenting would be needed to find the optimal value. However this is outside the scope of this research.
The *metric* parameter determines the metric to use when calculating distances. Since we are working with geographic coordinates, we use the haversine metric. This is the metric used when calculating distances between geographic coordinates [Wik21b].

3.4 Baseline Algorithm for detecting Staypoints

3.4.1 How does it work?

For every measurement the algorithm looks ahead a predefined amount of time. For example 30 seconds. It then checks the distance between these two points. If the distance is smaller than a predefined threshold it takes the average of these two points and saves this as a staypoint. The staypoints are clustered into POIs using the DBSCAN algorithm.
Algorithm 2 shows the pseudo-code for the baseline algorithm, where *point* is the current measured

location of the simulated walker. The predefined threshold values for the time between measurements and the distance between these points, are defined as *threshold_time* and *threshold_distance* respectively. The $staypoint_list$ contains all staypoints found in the current park.

Algorithm 2 Baseline algorithm:

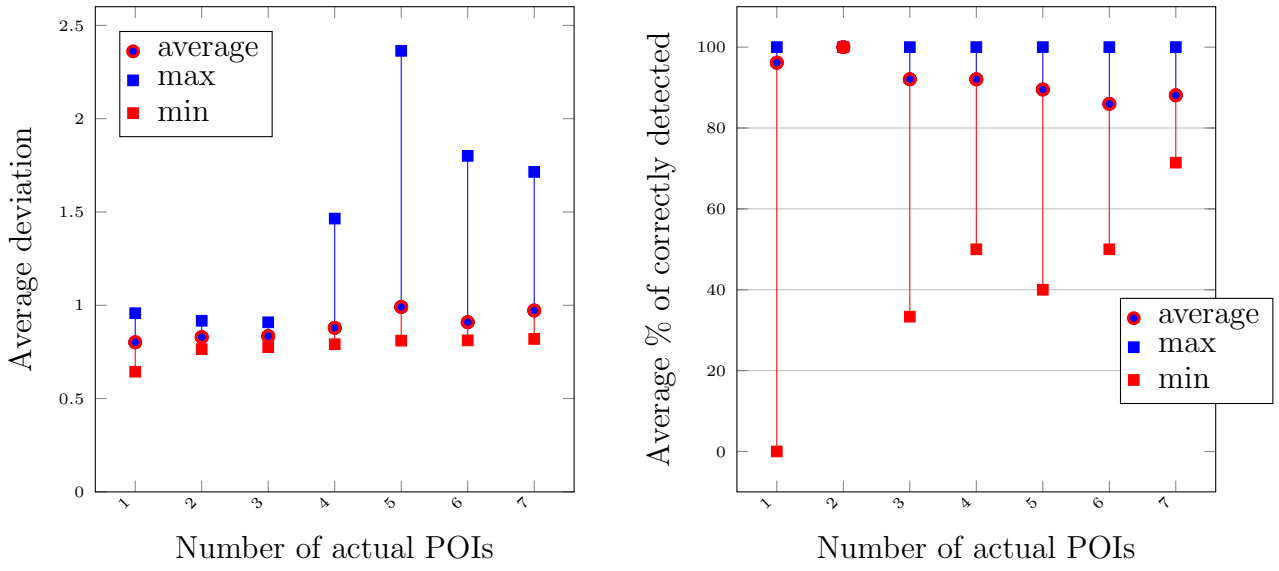
```
for each park do
    staypoint_list = [ ] // Create an empty list for the current park.
    for each walker do
        point = 0 // Start at the first measurement.
        while point + threshold_time < measurement_time do
            if getDistance(point, point + threshold_time) < threshold_distance then
                staypoint_list = getAverage(point, point + threshold_time)
            point += 1 // Go to the next measurement.
    DBSCAN(staypoint_list) // Cluster staypoints to get POIs for the current park.
```

3.4.2 Measuring performance

In this section we will look at the performance of the baseline method, using the performance indicators. We used two different threshold values for the baseline algorithm. One where the threshold was set to 35 seconds, and one with the threshold set to 20 seconds. Since the results with the 35 seconds threshold were better, we will use these scores moving forward. Average scores of the baseline method with the threshold set to 20 seconds can be found in Table 2.

Average deviation from actual POI

Figure 4a shows the deviation of the predicted POI from the actual POIs. The lower the score the closer the predicted POIs are to the actual ones. The deviation is measured in meters. We assign predicted POIs to an actual POI by looking at the distance between the two. If a predicted POI lies within 5 meters of an actual one, we will check the deviation. We consider these POIs 'found'. We can only look at predictions that found an actual POI, because the other predictions are false positives. There is no good way to assign these false positives to an actual POI. If the algorithm considered some noise to be a POI, we could assign this detected point to the closest actual point. But this would be incorrect, since the noise cluster and closest actual point might not be related at all. This means that these results should only be considered in combination with the average percentage of correctly found POIs and false positives. Otherwise it might be that only one point was correctly detected, while 5 others were missed. In this case the deviation could still show a good score. While the algorithm did not perform well.



(a) Average deviation, using baseline method, from actual POI in meters per group of parks.

(b) Average percentage of correctly found POIs, using baseline method, per group of parks.

Figure 4: Average deviation and average percentage of correctly detected POIs for baseline method. Where parks are grouped by number of POIs.

When the number of POIs in a park increases, the average deviation also increases slightly. This could be due to the park being busier with POIs. Which could cause the algorithm to struggle. A bigger difference can be seen between the maximum extremes. The first three groups have a relatively small maximum. While the maximums of the last three groups are much larger. Again this could be caused by the parks being busier with POIs in the last three groups. The worst prediction happened in the group with five POIs. The prediction was around 2.4 meters off.
The average deviation over all groups is 0.87 meters. This means that on average when a POI has been correctly detected, it has been detected with an accuracy of 0.87. We assume that this accuracy is good enough for most practical implementations. Therefore we consider this to be a good score.

Percentage of correctly found POIs

Figure 4b shows the average percentage of correctly found POIs. A correctly detected POI is one that lies within five meters of an actual POI. Furthermore, there can only be one detected POI per actual POI. A second detected POI inside the same 5 meter radius will be marked as false positive. In this case the min-max range also shows the reliability of the algorithm.
The figure shows that the average performance of the baseline method slightly decreases when a park contains more POIs. In the group with one POI there has been at least one case in which the algorithm did not find the POI correctly. Therefore we have a minimum score of zero percent in this group. If there is only one POI to find, the chances of finding zero percent are larger than when there are seven POIs in the park. Because if you miss the only chance of finding the POI, you have found zero percent.
The best performing group is the group with two POIs per park, with an average score of 100 percent. The worst performing group is the group with 6 POIs per park, with a score of 86.84

percent. This is a difference of 13.16 percentage points. The average score over all groups is 92.38 percent. This means that on average 92 percent of the actual POIs will be found using the baseline method.

False positives

The third and final performance indicator is the number of false positives. False positives are the points that have been incorrectly detected. It is important to take these into consideration. The average percentage of correctly found POIs might be very good, but with a lot of false positives. This could mean that we just predicted so much POIs, we found the actual POIs by chance. When doing POI detecting in a real scenario, we only want to find real POIs. Therefore the number of false positives gives valuable information about the performance of the method.
In short, a false positive is a point that was detected, but was not correctly detected. Meaning it was not inside a 5 meter range of an actual POI that had not been detected yet.
Figure 5 shows the number of false positives the baseline method produced on the simulated dataset.

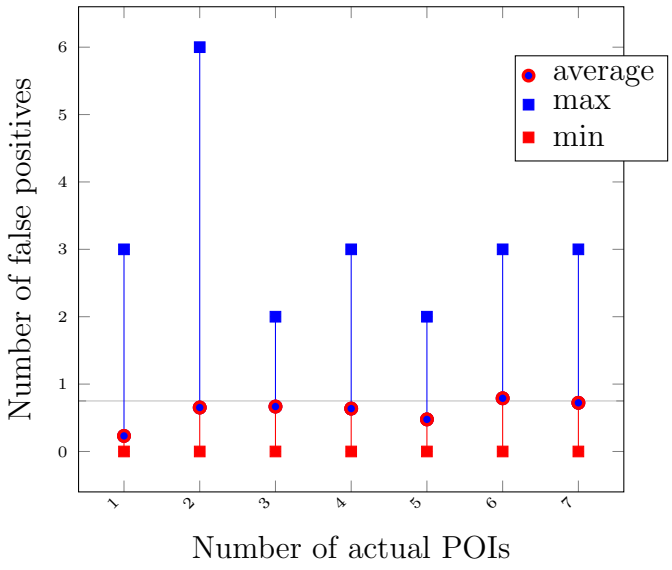


Figure 5: Average number of false positives, using baseline method, per group of parks. Where parks are grouped by number of POIs.

The figure shows that the group with one POI per park has the lowest number of false positives. This can be explained by the fact that there is only one actual POI. This means that there is less possibility for the detection of false positives. Thus leading to a lower number of false positives. We can see the opposite for the last two groups. Here the number of false positives is highest. These are the groups with six or seven POIs per park. Therefore there is are more opportunities to incorrectly detect POIs. The group with 5 POIs is an outlier. Since there is no information to suggest otherwise, we think this is just purely because of chance.
The overall average of falsely detected POIs by the baseline method is 0.58. We would consider this to be a pretty good score. Since this score means that on average the baseline method detects less than one point incorrectly.

We have to keep in mind however, that the scores for the groups of one and five POIs bring this average down considerably. If we ignore this outlier, we would see a better score. Also, we could question if it is realistic to have parks with only one POI.

3.5 C-DBSCAN Algorithm for detecting staypoints

3.5.1 How does it work?

C-DBSCAN adds two constraints to the original DBSCAN algorithm to avoid potential errors in staypoint detection. The first constraint is all points in a cluster should be temporally sequential. This means the sequential order should increase one by one and no 'sudden increase' is allowed in the cluster. If such a 'sudden increase' is found, the cluster will be divided into two potential clusters at the point of sudden increase and each one will be tested to see if it satisfies the condition of minimum number of points in one cluster. The second constraint is that the percentage (PCT) of abnormal points in a cluster should not exceed a given threshold. Where $PCT = |AP|/|C|$, $|AP|$ is the number of abnormal points in the cluster and $|C|$ is the total number of points in the same cluster.

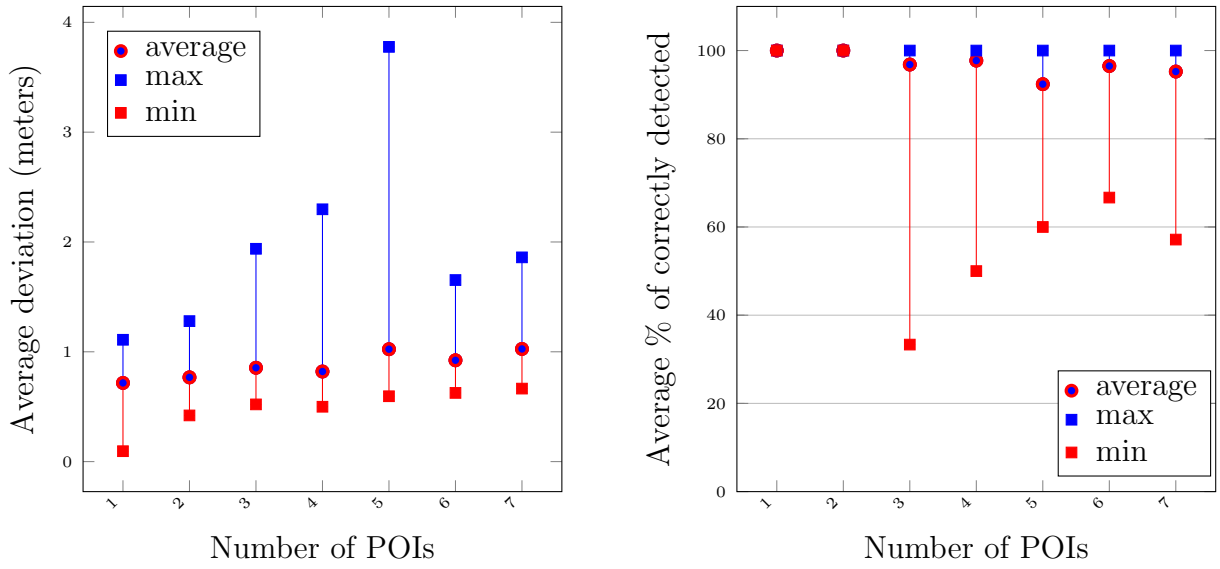
A point is marked as abnormal when the direction change coefficient (DCC) is close to or equal to 1. To find the DCC, we look at the direction changes of points in a cluster. Suppose we have three consequent points. We form two rays. The first one between the first and second point. The second one between the second and third point. The direction change of the middle point is the angle between these two rays. This angle is $\Delta\alpha$. To get the DCC we take the cosine value of $\Delta\alpha$. Because we are using the cosine value it does not matter whether $\Delta\alpha$ is positive or negative.
Within the range of a POI we expect the points to be scattered all across the area. This would lead to an even distribution of direction changes within the cluster. This means that the DCC of these points should differ from 1. Points with a DCC of 1, might represent a point along a straight line. We only expect straight lines when a walker is moving between POIs. Therefore points with a DCC close or equal to 1 are marked as abnormal points. Noted below is the formal definition of an abnormal point. Where DCC_{AP} is used to denote the approximation to 1. [GSY+15]

$$Abnormal\ point = \{x \in Cluster \mid DCC(x) \leq DCC_{AP}\}.$$

A benefit of this more complex C-DBSCAN algorithm comes from the small scale nature of our data. Since we are working on a small scale it is more likely that two different POIs are very close to each other. The first, time sequence, constraint of the C-DBSCAN algorithm helps to differentiate between POIs when they are close together.
The second constraint helps us avoid detecting POIs when in reality a walker is moving from POI to POI. Usually a walker would be walking in a relatively straight path towards their next destination/POI. We do not want to detect any POIs during this time. Here the second constraint comes in. When the walker is walking in a relatively straight path, the DCC of these points will be close to 1. Therefore this straight path will not be considered a POI.

3.5.2 Measuring performance

Since we want to have an equal comparison between the two methods for finding staypoints, we will use the same plots as we used when measuring the performance of the baseline method.



(a) Average deviation, using C-DBSCAN, from actual POI in meters per group of parks.

(b) Average percentage of correctly found POIs, using C-DBSCAN method, per group of parks.

Figure 6: Average deviation and average percentage of correctly detected POIs for C-DBSCAN method. Where parks are grouped by number of POIs.

Average deviation from actual POI

Figure 6a shows that as there are more points in the park the deviation increases. This increase is not large however. The best performing group, the group with 1 POI, has an average deviation of 0.72 meters. The worst performing group, the group with 7 POIs, has an average deviation of 1.03 meters. The difference between these groups is only 0.31 meters. This difference would probably be barely noticeable in any practical implementation of the algorithm.
The average deviation over all groups is 0.87 meters. This is the same score as the baseline method. As discussed before, in the baseline section, we consider this a good score.

Percentage of correctly found POIs

Figure 6b shows that the performance seems to get slightly worse when there are more POIs in the park. In parks with 1 or 2 POI all POIs were found correctly. However the performance decrease is very small. The group of parks with 5 POIs shows the worst average performance with 92.38 percent.
The average over all groups is 96.31 percent. This means that on average less than 4 percent of POIs are missed by the C-DBSCAN algorithm.

False positives

Figure 7 shows the average number of false positives per group that the C-DBSCAN method produces.

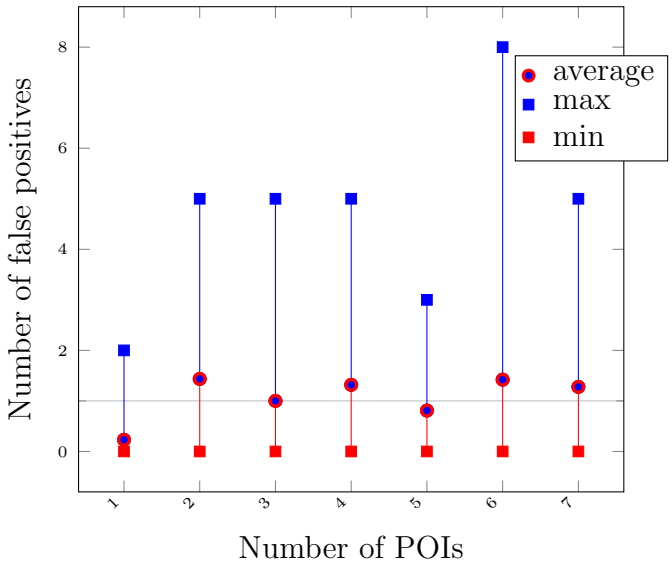


Figure 7: Average number of false positives, using C-DBSCAN method, per group of parks. Where parks are grouped by number of POIs.

The plot does not seem to show any trend. This suggests the performance is unrelated to the number of POIs that a park contains. The only exception is the group with one POI. This group has a considerably lower average detection of false positives. This is similar to what we see when using the baseline method. Therefore we again think this has to do with the number of opportunities to find incorrect points. The algorithm probably has an easier time with one POI in the park since there is less noise coming from other actual POIs.
We also again see that the group with five POIs is a bit of an outlier. Apart from the first group it is the only group to have a average score below one. This group is an outlier using both methods. Therefore we assume that the dataset for this group is 'easier'. Meaning the noise has generated in such a way that it is easier to find the actual POIs. Where to other groups have a less optimal generation of noise.
The average number of falsely detected points over the entire dataset is: 1.04. This means that, on average, this method produces just over one falsely detected POI.

3.6 Comparison between baseline and C-DBSCAN

In this section we will compare the two staypoint detection methods. Firstly we will look at the performance indicators. Secondly we will compare the Davies-Bouldin indexes. We also perform statistical tests to confirm that the differences are significant.

3.6.1 Performance comparison

Now that we have measured the performance of both methods, we can compare the results. We can ask the question if reliability is more important than mean accuracy or vice versa. A summary of the performance indicator can be found in Table 2, where T is the threshold value used for the baseline algorithm. This threshold determines how much time is between the two measurements

the algorithm looks at.

	% Correctly Found	Deviation	False Positives
Baseline with $T = 20$	66.38	0.79	16.67
Baseline with $T = 35$	92.38	0.87	0.58
C-DBSCAN	96.31	0.87	1.04

Table 2: Summary of performance indicators for baseline and C-DBSCAN methods.

The baseline algorithm with $T = 20$ scores worse than with $T = 35$. Therefore we will disregard the scores of the baseline method with $T = 20$ from now on.

Number of POIs per park
The graphs do not show a clear trend when it comes to the number of POIs per park. Parks with different number of POIs seem to result in the same performance. Therefore we will not look further into these different groups from now on.

Percentage of correctly found POIs
If we look at all groups together, the overall average of correctly found POIs using the baseline method is 92.38 percent. The C-DBSCAN method has an average score of 96.31 percent. So using this performance indicator, the C-DBSCAN method performs the best. Meaning that the C-DBSCAN method will, on average, find more of the actual POIs.

Average deviation from actual POI
The average deviation over all groups for the baseline method is 0.87 meters. For the C-DBSCAN method the average is also 0.87 meters. There is no difference between the two methods for this performance indicator. Meaning they are both equally accurate when a POI has been correctly found. As discussed before, we consider an accuracy of 0.87 meters a good score. In practice an accuracy of less than 1 meter should be good enough for most implementations.

False positives
When looking at the overall score, the comparison on false positives is quite clear. The baseline method outperforms the C-DBSCAN method. The average for the baseline method is 0.58 compared to the 1.04 score of the C-DBSCAN method. Where the baseline method produces a false positive roughly every two parks, the C-DBSCAN produces one roughly every park. The C-DBSCAN also has larger maximum values. Meaning that a bad extreme is worse for C-DBSCAN predictions than it is for predictions made with the baseline method. Therefore the baseline method seems to be the clear winner for this performance indicator.

Statistical tests
To determine if the differences between the two methods are significant, we use a statistical test. Our null hypothesis will be: The baseline and C-DBSCAN methods will perform equally on the two performance indicators.
To choose the correct statistical test we need to know what kind of variables we are dealing with.

We want to answer the question: What is the effect of using two different algorithms on the average accuracy score using the same dataset. In this case the predictor variable is the two different algorithms. Which is a categorical variable. The outcome variable is the average accuracy score. This is a quantitative variable. [Bev21] This means we could use a paired t-test as our statistical test. We will use the paired t-test for both performance indicators. It is important to note that we do not make a distinction between parks with a different number of POIs like we did in the plots shown earlier. We look at the accuracy over the entire dataset.
The paired t-test does not need a specific minimum number of samples. A rule of thumb is that a minimum of 30 samples is used. One 'park' is one sample. We have 150 parks, so our sample size is large enough.
For the statistical test we will define α to be 0.05. Our null hypotheses are that there is no difference between the scores. We used SPSS to calculate the outcomes of the paired t-test. Results are shown in Table 3. The t-distribution value is the value found in the Student T-table where, $\alpha = 0.05$ and $df = 149$.

	t	df	Two-sided p	t-distribution
% correctly found	3.28	149	0.001	1.98
False positives	3.72	149	< 0.001	1.98

Table 3: Test statistic and t-value for both statistical tests.

In both cases t is larger than the t-distribution value. This means the differences are significant. Therefore we can reject our null hypothesis. Meaning the C-DBSCAN algorithm outperforms the baseline method on the percentage of correctly found POIs performance indicator. While the baseline method outperforms C-DBSCAN when it comes to avoiding false positives.

Performance indicators conclusion
Looking at the three performance indicators, we can disregard the deviation indicator. Both method produce score the same on this indicator. We are left with the percentage of correctly found POIs and the number of false positives. The C-DBSCAN method has a better percentage of correctly found POIs, while the baseline generates less false positives. This essentially means that, on average, the C-DBSCAN method finds more actual POIs at the cost of producing more false positives.
The performance indicators do not give us a clear winner. Both methods could be useful depending on the task at hand. If it is very important that all POIs are found, the C-DBSCAN method might be more useful. Vice versa, if accuracy is more important and you do not want to find false positives, the baseline method is the best.

3.6.2 Davies-Bouldin Index

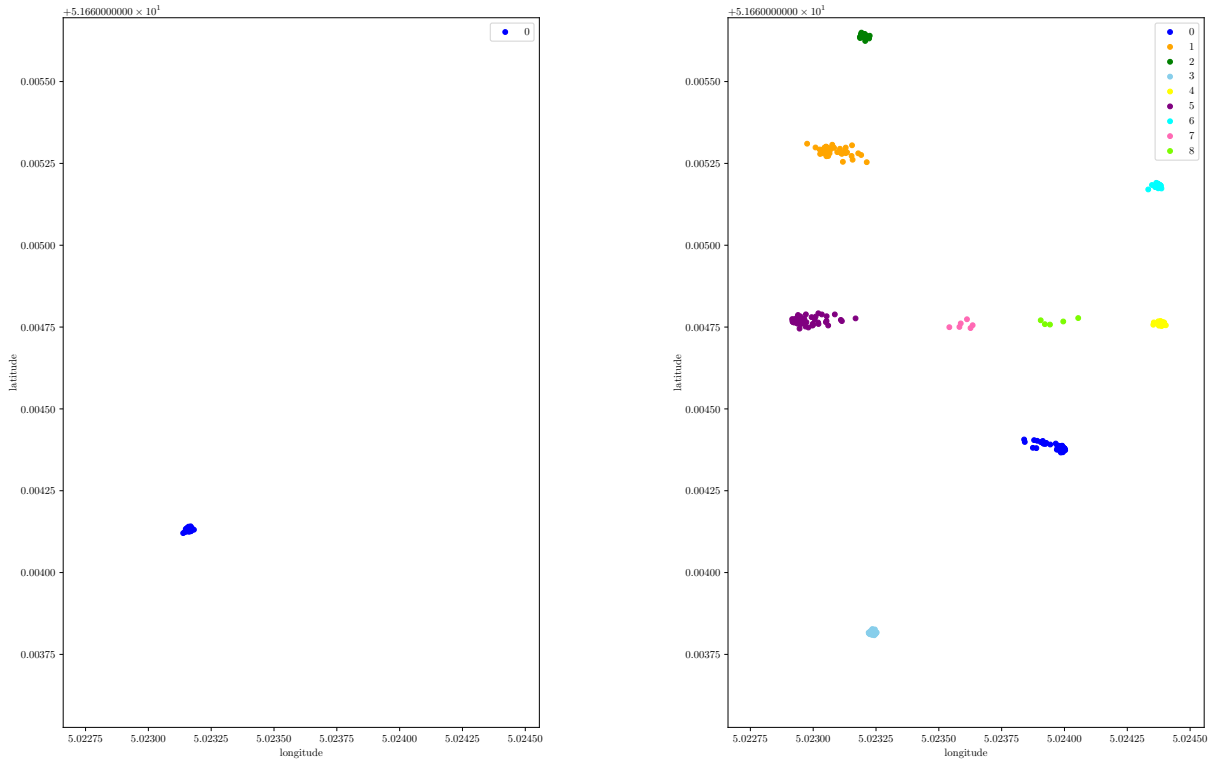
The Davies-Bouldin (DB) index captures the intuition that clusters that are well-spaced from each other and themselves very dense are likely a good clustering [Sci18]. This metric would be useful since we expect a cluster of staypoints to be densely concentrated around an actual point of interest. We also expect the POIs to have some distance between them. So an algorithm that finds POIs that are well-spaced from each other is likely a good algorithm.
The minimum score of the Davies-Bouldin Index is zero. The lower the score the better. Keep in

mind that the index does not take the ground truth in consideration. It only looks at the structure of the clusters and their relative position.

	Davies-Bouldin Index
Baseline	0.8290296727267201
C-DBSCAN	1.176628263618832

Table 4: Davies-Bouldin Index scores for baseline and C-DBSCAN methods.

Table 4 shows the baseline method scores the best. Meaning the clusters formed by the baseline method are denser and better spaced out. These clusters consist of staypoints. Meaning the staypoints generated by the baseline method are denser and spaced out better. This suggests that the baseline method has a higher accuracy, since there are fewer points that are further away from the actual POI. However, we found earlier that the percentage of correctly found POIs is lower using the baseline method. Meaning that if the baseline detects an actual POI, this detection should have a lower deviation to the center of the actual POI than when using the C-DBSCAN method. However we have already seen that the baseline and C-DBSCAN perform equally in this deviation test.
We can conclude that this index shows us that the baseline method should be more tidy. Therefore we would expect a greater accuracy. However, the index is not based on ground truth. Its outcome is also not explainable using earlier observations. Therefore we should value this index less than the performance indicators.
Figure 8 shows a comparison between the best and worst scoring predictions using the C-DBSCAN method according to the Davies-Bouldin index. The best scoring prediction shows one very compact cluster. The worst scoring prediction shows multiple less compact clusters. These two extremes give a little insight in how the Davies-Bouldin Index works. It is interesting to note that the best performing clustering only contains one cluster, while the worst performing clustering has the maximum number of clusters. It seems to be easier to get a good score with fewer clusters. Probably because there are less potential clusters that can worsen the score. The chances of getting one good cluster out of one, is larger than getting 7 good clusters out of 7. This suggests that the number of clusters has an effect on the Davies-Bouldin Index.



(a) Best scoring prediction.

(b) Worst scoring prediction.

Figure 8: Comparison between best and worst staypoint clustering using the C-DBSCAN method according to the Davies-Bouldin Index.

4 Conclusions and Further Research

4.1 Discussion & further research

In some graphs we see a slight trend. If the number of POIs go up, the algorithms start to struggle. The trend in the graphs is so small, we considered it insignificant. However, we do not know what happens when parks can generate with more than 7 POIs. The trend might continue and become significant.

In this thesis we have not experimented with the hyper-parameters of the DBSCAN and C-DBSCAN algorithms. Perhaps another configuration of these parameters gives us different results. It might be possible to minimize the number of false positives, making the algorithm more viable.

Since we did not have real world data for our experiments, we used simulated data. We tried to create a simulation that is close to reality. But is still a simulation nonetheless. Visitors of the park might show different behaviours as to what we expected when creating the simulation. Therefore running the experiments with real world data might give different insights.

4.2 Conclusion

We wanted to find an algorithm that can detect POIs from GPS data. We have found out that although the DBSCAN algorithm is an older algorithm, it still holds up today. As pointed out by Schubert et al [SSE+17]. DBSCAN is not specifically focused on GPS data, therefore we used an extension called C-DBSCAN. This algorithm was specifically designed to be used with GPS data as described by Gong et al [GSY+15].
So to answer our first research question: The C-DBSCAN is a state of the art clustering algorithm for GPS data.

Secondly, we wanted to find out if the algorithm is suited for use in the Vossenberg park. To test the performance of the C-DBSCAN algorithm on small scale GPS data, we compared it to a baseline method. We found out that the C-DBSCAN method is very similar in performance to the baseline method. We concluded that both methods can have its uses. With the C-DBSCAN method finding more POIs at the cost of producing more false positives.
So is the C-DBSCAN algorithm suited for the type of data collected in Kaatsheuvel? That depends on the situation and requirements of the user. It found most of the POIs with great accuracy. The only drawback is the frequent detection of false positives. The C-DBSCAN algortihm produces an average of 1.04 false positive per park. This means you get at least one false positive every time you run the algorithm. In the case of the Vossenberg Park this might lead to misinformation. Areas might be labeled as POIs while they are not. If they a park is designed based on this misinformation, investments might end up in the wrong area. This would be unfortunate since the funds invested in this area of the park could have been invested in another area. If detecting false positives is not a problem, or if a human expert is available to filter out the false positives, the algorithm works great. Otherwise you might be better off looking for another solution.

References

[Bev21] Rebecca Bevans. Statistical tests: which one should you use?, Sep 2021.

[BK07] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data Knowledge Engineering*, 60(1):208–221, 2007. Intelligent Data Mining.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[GSY+15] Lei Gong, Hitomi Sato, Toshiyuki Yamamoto, Tomio Miwa, and Takayuki Morikawa. Identification of activity stop locations in gps trajectories by density-based clustering method combined with support vector machines. *Journal of Modern Transportation*, 23, 07 2015.

[GT15] Junhao Gan and Yufei Tao. Dbscan revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 519–530, New York, NY, USA, 2015. Association for Computing Machinery.

[KCG+11] Sonia Khetarpaul, Rashmi Chauhan, S. Gupta, L.V. Subramaniam, and Ullas Nambiar. Mining gps data to determine interesting locations. *Proc. WWW*, 01 2011.

[sci] sklearn.cluster.dbscan.

[Sci18] ODSC Open Data Science. Assessment metrics for clustering algorithms, Nov 2018.

[SSE+17] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3), jul 2017.

[vDE15] Frank van Diggelen and Per Enge. The world's first gps mooc and worldwide laboratory using smartphones. *Proceedings of the 28th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2015)*, pages 361–369, September 2015.

[Wb21] Wikipedia-bijdragers. Nagoya — wikipedia, de vrije encyclopedie, 2021. [Online; accessed 9-augustus-2021].

[Wik21a] Wikipedia contributors. Dbscan — Wikipedia, the free encyclopedia, 2021. [Online; accessed 1-December-2021].

[Wik21b] Wikipedia contributors. Haversine formula — Wikipedia, the free encyclopedia, 2021. [Online; accessed 1-December-2021].

[Wik21c] Wikipedia contributors. Preferred walking speed — Wikipedia, the free encyclopedia, 2021. [Online; accessed 1-December-2021].