Online Hotspot-based Predictions

for Aalborg City Bike

SW701E14

Internet Technology

7th Semester, Fall 2014

AALBORG UNIVERSITET
STUDENT REPORT

Department of Computer Science
Selma Lagerløfs Vej 300
9220 Aalborg Ø
Phone (+45) 9940 9940
Fax (+45) 9940 9798
http://www.cs.aau.dk

Title:
Online Hotspot-based Predictions for Aalborg City Bike (OHPAC)

Subject:
Internet Technology

Project period:
02-09-2014 – 19-12-2014

Project group:
sw701e14

Participants:
Alexander Drægert
Bruno Thalmann
Mikael E. Christensen
Mikkel S. Larsen
Stefan M. G. Micheelsen

Supervisor:
Giovanni Bacci

Printings: 7

Pages: 54

Appendices: 5

Total pages: 70

Abstract:

The purpose of this project was to improve on the already existing Aalborg City Bike system, by implementing a software solution.
In order to improve on the already existing Aalborg City Bike system, it was first analyzed, identifying and defining problems to be improved upon. Based on these problems, along with information gained from an interview with Aalborg Kommune, a solution was formed. This solution built on top of the current Aalborg City Bike system, with the addition of GPS receivers and the removal of bike stations.
Instead of bike stations, hotspots were introduced. These hotspots were found by using the clustering algorithm DBSCAN. Hotspot representation was then simplified by obtaining its convex hull.
Time-homogenous Markov chains, represented as a matrix, was then introduced in order to form a model of the average bike usage. Based on this model, predictions were made about when to expect bikes at the different hotspots.
In order to make the data available to the users, a web service was implemented.
A simulation of the real world was created in order to ensure that the implementation was successful. The results of the tests suggest that the implementation was successful.

Source code:
https://github.com/sw701e14/code/tree/d0909818e25db18b90b7111363979e8bd4c61540

Preface

This report has been prepared by 7th semester Software Engineering students at Aalborg University, during the fall-semester of 2014. It is expected of the reader to have a background in IT/software, due to the technical content.

References and citations are done by the use of numeric notation, e.g. [1], which refers to the first item in the bibliography.

In the report, this project will henceforth be referred to as OHPAC. Whenever the report refers to 'we', it is to be understood as the members of the project group sw701e14.

Finally, we would like to thank our lecturers, and an especially big thanks to our supervisor Giovanni Bacci, for his excellent supervision throughout the semester.

Contents

Introduction

In 2009, Aalborg Kommune implemented the CIVITAS-ARCHIMEDES project[1]. As part of this project, the Aalborg City Bike[1] project was launched, making communal bikes available throughout the city[2]. In order to maximize the availability of the bikes, a simple system was chosen; practically free and very easy to use. To use a bike one needs only to give a 20DKK deposit, which will be returned at the end of the ride. Despite the bike usage being free, the bikes are well equipped and of good quality [3].

To further increase the likelihood of people using the bike as primary transportation, Aalborg strives to become a 'bicycle city'. A bicycle city is a city which makes an effort to better its cycling infrastructure [4].

Aalborg Kommune has over the years made several efforts into achieving the aforementioned goal, amongst others they have:

- created a bike highway from the city to the university[4].
- built pedestrian safe zones between bus stops and bike lane[5].
- organized campaigns to increase awareness about cyclists[6].

This increased focus on bikes as a primary source of transportation, is further emphasized by a publicly available bike system. However, the system as it is now, exhibits some problems that prevents it from completely fulfilling its goal. This will be the focus of this project, with a basis in the following initial problem:

What are the problems with the current Aalborg City Bike system, and how can they be improved by implementing a software solution?

[1] In Danish: Aalborg Bycyklen

Chapter 1

Analysis

In order to explore the problems and shortcomings of the current Aalborg City Bike system, a short analysis was carried out. The first step was establishing contact with Aalborg Kommune, in order to learn more about the current system, and what is to come of the system. This was done through an interview. Based on the interview, and other sources, the current system was described and analysed. The problems and shortcomings were identified and defined. Then other, already existing, systems were examined, in order to draw inspiration. Finally, based on this analysis, a problem statement was compiled.

1.1 Interview with Aalborg Kommune

The Aalborg City Bike project is maintained by Aalborg Kommune. In order to get information about Aalborg City Bike, an interview was conducted with three people involved with the project.

The interview was carried out October 10th, 2014, as a semi-structured interview. The interviewees were:

- Brian Høj; Responsible for the Aalborg City Bike project.
- Jesper; Member of the Aalborg City Bike team.
- Anne Mette; Member of the original team which implemented the CIVITAS-ARCHIMEDES project[2].

There were two overall objectives of the interview:

1. To gather information about the current system.
    - How is the system used?

- Who uses the system?
- What are the problems and shortcomings of the system?

2. To determine if the project members themselves already had improvements in mind.

1.1.1 Results

The following section is based on the Danish summary of the interview, found in Appendix A.

Intended use The intended use of the bikes are single, short trips. This means that the bikes are not meant to be used for longer periods of time, nor to completely replace a personal bike.

Users and usage There is no target group for the usage of the bikes. The bikes are currently not tracked so they were not able to say anything about the actual usage.

Future plans The interviewees have thought about installing GPS receivers in the bikes before, to receive data about their movement. They would really like to possess an overview of the usage of the bikes and usage patterns.

Specifically, they would like some information about:

- What routes are travelled?
- How long, both in distance and time, are the trips?
- When are bikes used?
- Who uses the bikes?

Rent/Booking It has never been the intention of the Aalborg City Bike project to be able to rent or book a bike, because it would hamper the spontaneous use of the bikes. If all the problems with booking could be solved in a reasonable manner, they would not rule out the future of such a system.

Missing bikes According to Traffic & Roads, Aalborg Kommune[3] there was a 11% loss of city bikes from 2009 to 2011 (237 to 210 bikes). According to the interviewees there now, fall 2014, remain 200 of the original 237 bikes from summer 2009. The interviewees see this as such a small reduction in bikes, that missing and stolen bikes are not considered a problem.

1.1.2 Conclusion

The interview yielded the following points which will be used in deciding which problems to focus on.

No renting/booking system This was deemed a bad idea, as it opposes the intention of keeping the system simple and available.

No specific target user group The interviewees stated that there is no target group.

Statistics about usage The interviewees would like some statistics about how the bikes are used; e.g. which routes are traveled. They also liked the idea of adding GPS receivers to the bikes in order to achieve this.

Short period usage The intended use of the system is short trips.

1.1.2.1 Further contact

After the interview we tried to contact Aalborg Kommune again, but despite several attempts, it was not possible to come into contact with them.

1.2 Aalborg City Bike

Aalborg City Bike is a communal bike service and therefore available to anyone living in or visiting the city of Aalborg. The service is available from the 1st of May until the end of October [2].
The service consists of:

- 200 bikes (140 active) and
- 21 bike stations, with room for 170 bikes in total.

Each bike is provided with a lock, that enables locking a bike to a station (similar to shopping cart systems). The bike is unlocked by depositing a coin (20 DKK) into the lock, after which it the bike is freely available for the user. By returning the bike to a station, and re-applying the lock at a free stand, the deposited coin is returned.

Missing and broken bikes are reported through a form on the Aalborg City Bike website: `http://www.aalborgbycyklen.dk/`.

The general utilization of the service can be summarized as follows:

1. A user unlocks a bike at a station, by inserting 20 DKK into a locked bike.
2. The user can freely use the bike around the city.
3. The user can return the bike at any station by locking the bike.

1.2.1 Problems in the Current System

This section will consider the challenges associated with Aalborg City Bike. The challenges described are devised from our own experience and perspective, as there is no target group according to the domain experts (Aalborg Kommune).

Bikes left outside stations The low deposit fee enables users to leave bikes, away from stations, without consequence other than the limited loss of 20 DKK. It is possible to ride straight to another destination, outside a station, and leave the bike there. The problem with this is that when someone else needs a bike, they might not be able to find one at a station, and without the possibility of knowing where they might be, outside of the stations. Additionally, Aalborg City Bike has to find the unused bikes outside stations, collect them, and return them to the stations.

Too few stations A ride is supposed to start and end at a station. This, however, can be a problem when the user's actual start or destination is not close to a station. Strictly speaking this shouldn't be possible, as it is not allowed to park a bike outside a station. Practically speaking it is possible to do, and when done the problem as described in **Bikes left outside stations** emerges.

Making short stops The bikes can only be locked at stations. This means that when a user, during a ride, wants to make a stop where there is no bike station, he has to leave his bike unlocked. This entails a risk that a new user comes along and takes the bike (which now can be done without 20 DKK deposit, as it is not locked at a station), leaving the original user without a bike. The original user then has to go to a station to acquire a new bike, hope to get lucky and find another bike nearby, or find another source of transportation from this point on.

No bikes at a station When users want to use a bike, they have to go to a station and see if there are any available bikes. Otherwise they have no way of knowing whether there is a bike at a station or not.

No way of knowing when a bike will arrive If a user needs a bike at a specific time in a specific area, he can not be sure that there will be a bike at that exact time and area. Nor can he know when a bike will arrive.

Broken bikes Aalborg Kommune does not know when a bike is in need of repair and rely solely on the users to report the whereabouts of broken bikes. Potentially there could be bikes left outside the common-usage-area in need of repair. These bikes would be out of service for a long time.

1.3 Existing Systems

Already existing systems are explored in order to draw inspiration from them. They will also serve as guidelines to what should and shouldn't be repeated in the new adaption of the system.

There are many public bike systems around the world, mainly in bigger cities. The systems differ a lot in both how and where bikes are acquired and how the use of the bikes is paid. The two overall methods lie in either a rent-system or an grab-if-available-system. The rent-systems usually have very few stations, and bikes are rented for a longer period of time (days), similar to a car rental service. Aalborg Bycyklen is an example of a grab-if-available-system.

Chosen for comparison are two systems; Bycyklen[1], located in Copenhagen and Citi Bike NYC, located in New York City. These systems are similar to Aalborg Bycyklen, in that the bikes are publicly available, and at many different locations.

[1]Translation from Danish: 'The City Bike'

These two systems were also chosen because of their social and cultural similarities of the cities with respect to Aalborg. Rent-systems will not be considered, as these are too different from the more publicly available systems.

1.3.1 Copenhagen: Bycyklen

The overall purpose of Bycyklen[7][8] is to provide bikes for both single and repeated use.

A bike is rented at an hourly rate of 25 DKK/hour. In exchange for a fixed subscription of 70 DKK/month, the hourly rate is reduced to 6 DKK. In both cases, an account must be created first. This can be done using an internet browser or the bike-mounted tablet.

To use the service the user has to sign in with an existing account. The bike is then unlocked from the rack and the hourly charges begin. Once the bike is no longer in use and placed in a rack again, the user is logged out and the bike is locked.

If a bike is left outside a bike station for longer than 2 hours, fees are charged (200 DKK in Copenhagen/Frederiksberg, 800 DKK outside), or if it has been in use for longer than 10 hours (500 DKK). For short stops during a trip, the lock can be locked with an analogue lock, which is unlocked the same way as at a station, but only for the currently active user.

It is possible to reserve a bike by using their website, which in all cases cost 10 DKK. This will hold a bike, by locking it, up to 45 minutes before it is needed. Half an hour before the reservation time, a text message is sent to inform whether a bike was successfully held or not. Repeated reservations are allowed.

Currently, there is a total of 20 stations and an estimated average of 15 bikes per station, giving a total of approximately 300 bikes. The bike itself is electrically powered and equipped with a GPS receiver and a handlebar-mounted tablet. The tablet is used for registration and signing in/out. Additionally the tablet, with use of the GPS receiver, can provide navigational guidance.

1.3.2 New York City: Citi Bike NYC

Citi Bike NYC[9] is a system which makes bikes available for short trips through renting. Unlike most other systems, you do however not rent a single bike. There are three methods of renting bikes:

- 24-hour pass: $9.95

- 7-day pass: $25
- Annual membership: $95

With the 24-hour and 7-day passes, you can use a bike for an interval of 30 minutes at a time. With annual membership this interval is extended to 45 minutes. However, after each interval, you can just acquire another bike. If you do not return the bike after the interval, overcharge fees occur.

In order to acquire a bike with a pass, a payment is made with a credit card at a bike station, which provides a single-use code. This code is entered on the bike itself, which is equipped with a tablet solely for this purpose, which will unlock the bike from the bike rack. Along with the pass fee, a $101 security hold is placed. When a bike is returned, and another bike is to be acquired, the credit card must be used again. The credit card will not be charged again if a pass is already active, instead a new code is provided. For annual memberships a key is used to unlock bikes and no interaction with the bike station is needed.

The exact number of bikes and stations is unknown. The official website states that there are ”100s of stations” and ”1000s of bikes”[9]. The bikes are very simple, with no motor or GPS receiver, and the tablet in front is only used for unlocking bikes with either code or membership key.

1.3.3 Evaluation

Based on the problems described in Section 1.2.1, the existing systems will be evaluated in regards to whether they solve these problems.

Bikes left outside stations In Bycyklen, this is partially solved by the hourly charges until the bike is returned, along with the additional fees if the bike is immobile for 2 hours or used for more than 10 hours. Presumably, the GPS receiver in the bikes can also be used to locating misplaced bikes (this is based on the fact that users agree to let Bycyklen collect anonymous data about start/end destination, and route usage).

In Citi Bike NYC overcharge fees are continuously charged, after exceeding the allowed half hour trip, until it is returned to a bike station. Nothing can be found about whether this goes on forever or not, however this ensures the user returns the bike to avoid fees.

Too few stations Bycyklen has 20 stations in Copenhagen making the solution difficult to use for trips with a destination away from a station.

Citi Bike NYC tries to solve this problem by placing so many stations in the city that you are never far from a station. Whether or not this solution solves the problem is not known, but it seems like an expensive solution.

Making short stops Bycyklen provides a manual lock, for short stops (2 hours) during use. However, when done using a bike it must be returned to a station.

In Citi Bike NYC, the multitude of stations and the placement throughout the city should make sure that there is always a station close-by, no matter where you want to end your trip, including shorter stops.

No bikes at station Both Bycyklen and Citi Bike NYC have web applications with a map showing stations, along with an indication of how many bikes are available at the different stations. Citi Bike NYC also has native mobile applications, while Bycyklen only has the web application.

No way of knowing when a bike will arrive Neither Bycyklen nor Citi Bike NYC has a prediction system, but they have solutions that attempt to address the problem.

Bycyklen tries to solve the problem with reservations but they cannot ensure that a bike will be available, as it will only attempt to lock a bike 45 minutes before the reservation time if a bike is available.

Citi Bike NYC has no reservation system, one must consult an app for available bikes at stations, but the multitude of available bikes should ensure that there's always a bike, and the multitude of stations should ensure that you will not have to go far if no bike was at the first visited station.

Broken bikes Before using a Bycyklen bike, all faults on the bike must be reported, or else the user can be charged for faults found by the next user. Faults must also be reported during use, and if a fault hinders a trip, a refund can be made.

Using the mounted tablet on a Citi Bike NYC bike, one can report faults/damages by pressing a wrench-icon button. If faults are found during use, the bike is simply returned at a station and the aforementioned button is pressed.

1.3.4 Conclusion

Both systems do well in handling all the problems described in Section 1.2.1

They are, however, in conflict with the results from the interview with Aalborg Kommune, as stated in Section 1.1.1. Specifically, the simplicity of the system, where Aalborg Kommune did not like the idea of booking/reservations, as it would complicate the use of the bikes.

Another problem is the lack of usage statistics in the current system, meaning that it is not easy to find out where to place stations (which both existing systems require).

1.4 Problem Statement

This section contains the main problem for this project and the generic description of the required solution.

1.4.1 Analysis Summary

The analysis was based on the initial problem: *How can the current Aalborg City Bike system be improved, in order to make it more usable?*. From the analysis, it was found that the current Aalborg City Bike has several problems, limiting the overall usability (as described in Section 1.2.1).

The problems identified were:

1. Bikes left outside stations
2. Too few stations
3. Making short stops
4. No bikes at station
5. No way of knowing when a bike will arrive
6. Broken bikes

Additionally, certain limitations and requests were made by Aalborg Kommune, collected through the interview:

1. No renting/booking system
2. No specific target user group
3. Interested in statistics about usage
4. Short period usage

1.4.2 Scope

Due to the limited time and project members restrictions will be set, to remove focus from certain problems.

Making short stops is not solved directly, as the only solution we see can directly addresses the problem, is the use of a locking mechanism.

Statistics about usage is something Aalborg Kommune also requested. This, however, is deemed too far from the initial problem, as we won't be able to get any significant data without testing it on the actual bikes.

1.4.3 Solution

We have chosen the solution with the same bikes as the current system, with the addition of a GPS receiver on each individual bike, and completely removing the stations and bike locks. This is due to the restrictions set by Aalborg Kommune, requesting a simple system, in order to accommodate the unspecified target user group.

GPS receivers and Hotspots The greatest changes to the current system is the addition of the GPS receivers and the introduction of hotspots. As we do not have access to real GPS receivers, we will base the rest of the report on the following assumptions about the GPS receivers on the bikes: Each bike is equipped with a GPS receiver which uses GPS satellites to calculate its location.

The GPS receiver is presumed to send the location of the receiver, an accuracy stated in meters[2] and a time stamp indicating the time of the reading. This collection of data will be referred to as a *GPS location*. The GPS receiver will send a GPS location to our system at a fixed interval, but it is not guaranteed that the interval is exact.

Hotspots will serve as informal stations; areas of the map where bikes are often parked.

By combining these two things, we want to improve on the problems with the current system.

1: Bikes left outside stations is handled by the GPS receivers, as bikes and their location now can be found, no matter if it is at a hotspot or not.

[2]the definition of accuracy is based on the output from FollowMee [10]

2: Too few stations is automatically handled by the generation of hotspots. Whenever an area is a popular place to be going from/to, a hotspot should emerge and reflect this.

3: Making short stops is improved upon but not solved. If a user stops in a hotspot he can use the system to estimate how long he will have to wait before a new bike arrives at that particular hotspot. From this information he can decide whether to leave the bike or not (or possibly how long he should spend in a shop etc.). This is an improvement over the current system that provides no such information, however it does not solve the problem entirely.

4: No bikes at station is handled by locating the exact location of an available bike, or looking up nearby hotspots.

5: No way of knowing when a bike will arrive will be handled by modeling bike behavior and predicting when a bike will be available at certain hotspot(s).

6: Broken bikes will not be directly handled, but it is possible to infer from looking at bike locations and detecting when a bike has been immobile for a very long time, which could indicate that it either is too far away for anyone to pick it up, or broken.

Software A software solution is to be developed, consisting of 3 main software components, and a shared database. The structure of the solution can be seen in Figure 1.1.

The *Location Service* continuously updates the locations of the bikes. The *Model Updater* uses the stored location data to generate a model used for predictions. The *Web Service* makes an API available, in order to make platform implementations to make the data available as specified in the requirements. A web service was chosen as to make the system as available as possible, due to the unspecific target user group.

1.4.4 New Problems

In order to develop this system, with the addition of GPS receivers and hotspots, new problems emerge. The following chapters will focus on these problems:

- Defining and identifying hotspots
- Modeling the usage and using the model for predictions

- Making the data available through a web service

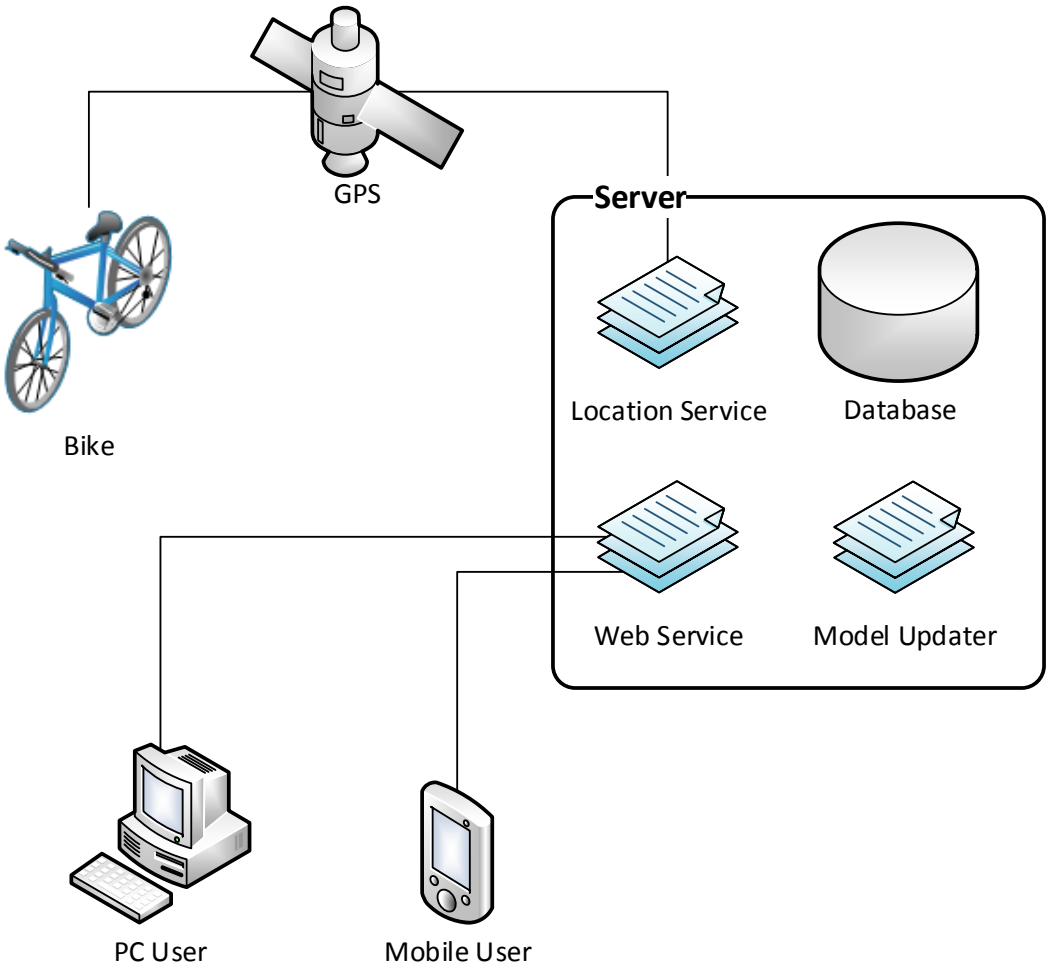


Figure 1.1: The overall structure of our solution

Chapter 2

Finding Hotspots

The data collected through the GPS receivers installed on the bikes can be used to find out where the users of the system most frequently start and stop their journeys. The point can be used to make the hotspots that OHPAC will use for predicting the behavior of the bikes. This chapter will explain how we will use the GPS locations to create hotspots that describe the most frequently visited places in the city.

These frequently visited places will be referred to as *Hotspots*. Hotspots are defined as confined areas in the city where users of the system start and arrive more frequently, and therefore the GPS locations are more frequent in this area.

For the purpose of making hotspots we need a definition of when a bike is standing still. The GPS receiver described in 1.4.3 will not tell if the bike is moving or not. We will instead consider a GPS location as standing still if the next GPS location received by the system is at the same spot taking inaccuracy into account. Such a point will be marked as standing still and will be referred in the rest of the report as a *static GPS location*.

In order to group all the static GPS locations, we explore clustering techniques, before choosing an appropriate one to implement.

Finally, in order to simplify hotspot representations, we represent its set of locations by means of its convex hull, this creates a polygon representation of a hotspot that will be easy to query for point location.

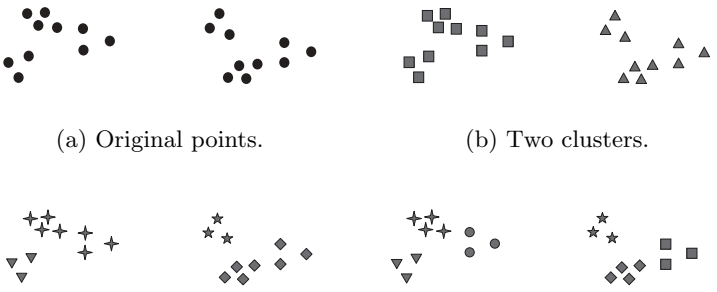
2.1 Clustering

A clustering is a collection of data object groupings (clusters). In order to identify hotspots we consider different clustering techniques. This section will describe the concepts of clustering as well as the possible algorithms.

This section is based on Pang-Ning et al. [11].

2.1.1 Cluster Analysis

Cluster analysis is a technique used to group data objects into clusters based only on the information the data itself contains. The requirement for membership in a certain cluster is often vague, as several clusterings can be made on the same dataset. An example of this can be seen in Figure 2.1, where the same dataset has been clustered in three radically different ways, even though all three could be regarded as correct, depending on the purpose of the clustering. The clustering therefore depends on the dataset and its purpose.



(a) Original points. (b) Two clusters.

(c) Four clusters. (d) Six clusters.

Figure 2.1: Different clusterings on the same dataset. From Pang-Ning et al. [11].

2.1.2 Types of Clusterings

This section will introduce the terminology used to describe the various types of clusterings.

Hierarchical versus partitional A *partitional clustering* is a division of data into pairwise disjoint subsets, where each object belongs to exactly one subset. If clusters are allowed to have subclusters, the clustering is said to be *hierarchical*. A hierarchical clustering is represented as a set of nested clusters organized as a tree, where each node is the union of its children.

Exclusive, overlapping and fuzzy clusterings A clustering is *exclusive* if an object is assigned to exactly one cluster. If an object can belong to

more than one group, the clustering is said to be *overlapping*. If a weight is used to describe the membership of sets, the clustering is said to be *fuzzy*.

Complete versus partial A *complete* clustering has every object assigned to a cluster, while a *partial* clustering can have outliers; objects that do not belong to any cluster.

2.1.3 Types of Clusters

The purpose of a cluster depends on the kind of data set it is applied on. In this section, the different notions of a cluster will be presented.

Well separated A cluster is a set of objects, where the objects that are similar are grouped in a cluster. Sometimes a threshold is used to define a minimum similarity. All objects in a cluster need to be at least as similar to all other objects in the cluster, for an object to be in the cluster.

Prototype based Objects are placed in clusters based on the prototypes that define the clusters. An object is placed in the cluster where the object is more similar to the prototype of that cluster, than to the prototype of all other clusters.

These prototypes can be either the average value of a cluster, or the most representative object of a cluster.

Graph based If the data can be represented as a graph, with the objects as nodes, clusters can be defined as connected components in the graph.

Density based A cluster is defined by the density of the data objects. A cluster is a dense region surrounded by a low density region.

2.1.4 Our Data and Purpose

The data we need to cluster is all the static GPS locations. We expect the points to be distributed in a wide area, but clustered around places where people place the bikes (hotspots). These clusters may have different size, depending on the physical properties of the places the bikes are left.

The purpose of the clustering is to find the areas where bikes are left most often, in order to make predictions on when a bike will arrive in those areas. These areas can then be considered our hotspots.

Based on this description we need a partitional, exclusive, partial and density based clustering.

Partitional, because we do not need hierarchies for our problem, we only need the select clusters that meet our requirements for a cluster. It needs to be exclusive because we want the clusters to be separated from each other so any point on the map is either in one cluster or not in any cluster. We want to find out where the bikes spend most of their time and therefore we want a density based partial clustering.

2.1.5 Techniques

This section will explore the techniques that exist in cluster analysis, as well as evaluate them, based on the requirements stated in the previous section.

2.1.5.1 K-means

K-means is a technique that creates a partitioning from prototypes, where the prototypes are the centers of clusters.

The basic algorithm takes a number K and generates K initial center points. Each data object is then assigned to the nearest center point. The center points are now updated to be the center of the created clusters. These steps are repeated until no point changes cluster, or the center points do not change.

Because we do not have a way of finding K before running the algorithm, this approach is not applicable to our problem.

2.1.5.2 Hierarchical clustering

Hierarchical clustering arranges the points in a hierarchy, dependent on the distance between points. Hierarchical clustering can be performed either by starting at the top of the tree or at the bottom. *Agglomerative hierarchical clustering* starts by considering all points as being in individual clusters, then generating the hierarchy by merging the closest clusters. *Divisive hierarchical clustering* starts by considering all points as being in a single cluster, then continues by splitting the clusters at each step.

We would like our clustering to be partitional, which makes this algorithm inappropriate for hour purposes. It should also be noted that hierarchical clustering is a complete clustering. For bikes used by many different users one would expect some noise in the data. Therefore it seems reasonable to only implement a partial clustering.

2.1.5.3 DBSCAN

DBSCAN is a density based clustering algorithm that uses the notion of a number $MinPts$ and a distance eps to determine if points are in the same cluster. If there are $MinPts$ in a radius of eps of a point this point will be regarded as a cluster. The rest of the cluster will be determined by expanding the cluster using the same definition on the neighboring points.

This algorithm fits our need for a an exclusive, partitional and density-based clustering algorithm. It therefore seems to be the best candidate for our purpose.

2.1.5.4 Choice of clustering algorithm

Of the three presented algorithms, only the DBSCAN algorithm fits our needs as stated in the description of the algorithm. Thus we have chosen to implement this algorithm for our clustering.
The DBSCAN algorithm will be described in more detail in the following.

The DBSCAN algorithm There exists different methods of determining density in a set of data points. One approach is to use a point as a center and calculate the number of points that are within a certain radius of the point. Using this measure it can be determined whether a point is placed in a dense region (a core point), at the edge of a dense region (a border point), or is in a region with sparse density (a noise point). A visual representation can be seen on Figure 2.2 and the exact definition of the three types of points are as follows[11]:

Core point These points have at least $MinPts$ points in a radius of eps from it. Here, $MinPts$ is the minimal number of points in the vicinity of a point, for it to be regarded as dense by the user, and eps is the radius in which to look for these points.

Border points A point that is not a core point, but is in the neighbourhood of a core point. One point can be a border point to several core points.

Noise point Any point that is neither a core point or a border point.

Given these definitions, the algorithm is described in Algorithm 1.

2.2 Convex Hull

In order to check whether a location is within a hotspot, we need to represent hotspots as something different than a cluster of locations. Instead of doing

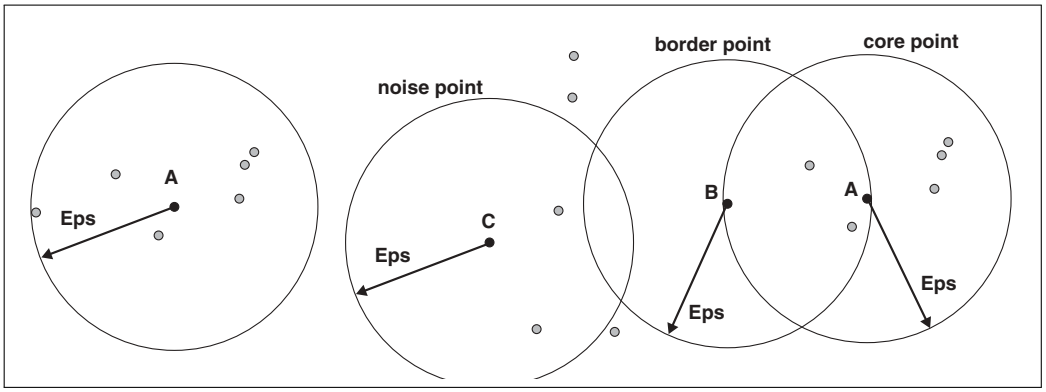


Figure 2.2: The three types of points in the clustering algorithm DBSCAN. Taken from [11, Page 528].

this with a simple geometrical shape, as this could misrepresent complexly shaped hotspots, we represent a hotspot as a polygon by applying convex hull to the clustered locations.

A convex hull could also be a misrepresentation of a complexly shaped hotspot, but as hotspots have no inherent shape we deem the convex hull to be a close representation of what we associate with hotspots. This way we reduce the amount of locations needed to represent a hotspot without much loss of information.

The following definition of convex hull is based on Cormen et al. [13, Section 33.3].

Given a set of points, the smallest convex polygon, for which each of the points in the set are either on the border of or inside the polygon, is called the convex hull of the polygon[13, Section 33.1-5]. A convex hull is represented by a set of points. We used an implementation of the Graham-Scan[13, Page 1031] algorithm to identify convex hulls for hotspots.

```
DBSCAN(D, eps, MinPts)
C ← 0
foreach unvisited point P in dataset D do
    mark P as visited
    NeighborPts ← regionQuery(D, P, eps)
    if sizeof(NeighborPts) < MinPts then
        mark P as NOISE
    else
        C ← next cluster
        expandCluster(D, P, NeighborPts, C, eps, MinPts)

expandCluster(D, P, NeighborPts, C, eps, MinPts)
add P to cluster C
foreach point Pn in NeighborPts do
    if Pn is not visited then
        mark Pn as visited
        NeighborPtsn ← regionQuery(D, Pn, eps)
        if sizeof(NeighborPtsn) ≥ MinPts then
            NeighborPts ← NeighborPts ∪ NeighborPtsn
    if Pn is not yet member of any cluster then
        add Pn to cluster C

regionQuery(D, P, eps)
return all points from D within the eps-neighborhood of P (including P)
```

Algorithm 1: The DBSCAN clustering algorithm[12]

Chapter 3

Predicting Behaviour

3.1 Modeling Behavior

As the purpose of OHPAC is to be able to predict the movement of bikes at a given time (see Section 1.4, Problem 5), it must be able to answer questions such as:

- How much time will it take before a bike will be in my vicinity?

The locations of the bikes are abstracted by creating hotspots, as described in the previous chapter. Thus the above question can be rephrased as:

- How much time will it take before a bike will be *in a given hotspot*?

To answer this question we need to model the behavior of an *average* bike. In other words, we need to provide a formal description of the movement of a single bike, representing the movement, on average, of all bikes.

3.1.1 Abstraction Model

As the system is based on anonymous usage, no user sensitive information is collected. Because of this, some uncertainty is expected in the model, as all bikes (and thus all users) are modeled as a single *average bike*. Modeling the movement of all bikes will be done by modeling the movement of this average bike as a stochastic process [14].

We must also consider that our model will be working on discrete-time location data. The GPS receivers that the bikes will be equipped with will only report the location of a bike at a *fixed* interval as described in 1.4.3

Additionally the accuracy of each of these measurements can be low, resulting in even more imprecision. As we are modeling the average bike, we can represent the state of the process as the location of the bike. In order to reduce the number of locations to a sensible number, we will use the generated hotspots instead. Using this definition, we can describe the movement of the bike as an infinite route $h_1, h_2, h_3, \ldots$ where $h_i \in H \mid i \in \mathbb{N}$ and H is the set of hotspots in the system.

Using this representation of the average bike, we see that the average bike must move from one hotspot h_i to another h_{i+1} in one *timestep*. Note that it could be that $h_i = h_{i+1}$, meaning that the bike has moved to the same hotspot as it departed from. This is also used to indicate that the bike has not moved. In Section 3.1.3 we will describe the modeling of the average bike in greater detail.

Taking all of the above into account, it is reasonable to suggest the abstraction model for the behavior of the average bike, to be a discrete-time Markov chain, where the states are used to represent hotspots.

3.1.2 Discrete-Time Markov Chains

The state of a discrete-time Markov chain model changes in a process at discrete time intervals. It can be viewed as a sequence (chain) of random variables $X_1, X_2, X_3, \ldots$ where each such variable is a function $X_i : S \to x$. Here, S is a countable set of states, representing the possible states (hotspots) in the process described by the Markov chain, and x is a *random* state from S. The subscript describes the time associated with each of the variables. Thus, X_i represents the possible states of the process at time i. This notation allows us to establish a timespan, as each step refers to a fixed interval.

This randomness will be biased towards certain states, given its current state (and possibly time), resulting in a set of probabilities for each state given a random variable. Let $\mathbb{P}(X_i = x)$ be the probability of X_i being in state x. The above will then allow us to express the probability of moving to an arbitrary state y, given the path we have traversed.

In other words, knowing which state we are in, and how we got to it, allows us to determine the probability of the next state being y:

$$\mathbb{P}(X_{n+1} = y \mid X_1 = x_1, X_2 = x_2, \ldots X_n = x)$$

Markov property For the sequence to describe a Markov chain, it must additionally have the Markov property. The Markov property states that for any state x we can determine the probability of the next state being y, without any knowledge of the path that led to x. We can formalize this as:

$$\begin{aligned} &\mathbb{P}(X_{n+1} = y \mid X_1 = x_1, X_2 = x_2, \dots X_n = x) \\ &= \mathbb{P}(X_{n+1} = y \mid X_n = x) \end{aligned} \tag{3.1}$$

Time-homogeneous Markov chains In an effort to simplify the problem of predicting the behavior of the average bike, we will be using time-homogeneous Markov chains. A Time-homogeneous Markov chain can also be thought of as a time-independent chain. In other words, predicting the next state of the chain can be done without knowledge of the time:

$$\mathbb{P}(X_{m+1} = y \mid X_m = x) = \mathbb{P}(X_{n+1} = y \mid X_n = x) \tag{3.2}$$

Time-homogeneous Markov chains provides us with a much simpler method of modeling the average bike. A time-homogeneous Markov chain is very simple to represent (as is described below) and easy to use when predicting the state of the average bike multiple steps into the future. For these reasons we have chosen to simplify our Markov chain models.

This simplification will mean that we are unable to determine a difference in behavior at different times in the process. Thus we cannot, using a single model, represent different behaviors of the average bike in the morning/afternoon, or in different days of the week. To allow for this distinction, multiple models could be generated, representing different datasets. The system would then simply select the model that corresponds to the current time-of-day/day-of-week.

Mathematical representation From the above definitions, we have that a Markov chain $\mathcal{M}$ can be described as a function $\tau_M : S \times S \to [0; 1]$, such that $\tau_M(x, y) = \mathbb{P}(X_{n+1} = y \mid X_n = x)$. As the chain is independent of time, we can represent it using a single matrix, containing all its transitional probabilities. Here is a generic representation of such a matrix:

$$\begin{bmatrix} \tau_M(s_0, s_0) & \tau_M(s_0, s_1) & \cdots & \tau_M(s_0, s_n) \\ \tau_M(s_1, s_0) & \tau_M(s_1, s_1) & \cdots & \tau_M(s_1, s_n) \\ \vdots & \vdots & \ddots & \vdots \\ \tau_M(s_n, s_0) & \tau_M(s_n, s_1) & \cdots & \tau_M(s_n, s_n) \end{bmatrix} \tag{3.3}$$

This is a simple representation, which will allow us to predict probabilities multiple steps into the future, by applying the same matrix to some initial probability distribution multiple times.

Visual representation Markov chains are typically represented using a DAG[1] with weights on the edges. In such a graph, nodes represent states and the weight of edges represent the probability of transitioning between two states. This is a direct mapping of the matrix described above.

In Figure 3.1, we give an example of a Markov chain that models an average bike, travelling back and forth between two hotspots $\{h_1, h_2\}$. From the weights on the diagram edges, we can read the time-homogeneous probability of moving from one state to another (or staying in the same state). For instance, a bike currently in h_1 has probability 0.7 of transitioning to h_1 (staying) and probability 0.3 of transitioning to h_2.

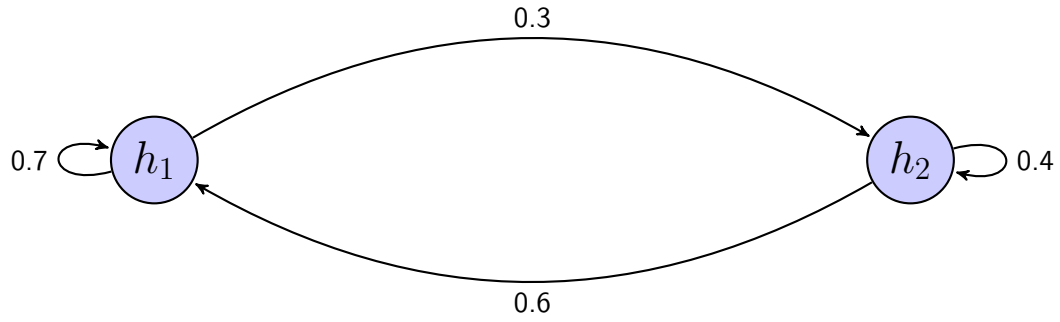


Figure 3.1: A Markov chain model with two hotspots

In Figure 3.1, there are no edges with weight zero. Any such edges would represent transitions with zero probability (*"impossible"* transitions). Had there been edges with weight zero, they could have been removed to simplify the graph. This also means that any *missing* edge between two states s_1 and s_2 implies that $\tau(s_1, s_2) = 0$. An example of this can be seen in Figure 3.2, where states d_1 and d_2 are not connected.

3.1.3 Modeling the Average Bike

Throughout this section, the average bike has been modeled using a state set, consisting of hotspots and an associated transition function (represented using a probability matrix). We presented a simple example of this, using only two hotspots, in Figure 3.1. Using this model, we will be able to make predictions on the path of the bike in the system. To do this we require an initial state of the system.

[1]Directed Acyclic Graph

This state should represent the current (at the time of querying) location of the bikes in the system. Using this representation, we will be able to compute the probability that the bikes will be in some hotspot after a number of time steps.

Insufficient states As we are looking to use the Markov chains to predict whether or not a bike is available at a hotspot, we must also be able to represent whether they are in use or not. The modeling described so far only allows for a bike to be at a hotspot, and thus we are not able to give a direct representation of a bike in transit.

To accommodate this restriction, we introduce an additional state for each of the hotspots. These states are called *departure states* and the departure state associated with hotspot h_i is denoted as d_i. The simple two-hotspots model, described above, can then be expanded with departure states, as seen in Figure 3.2.

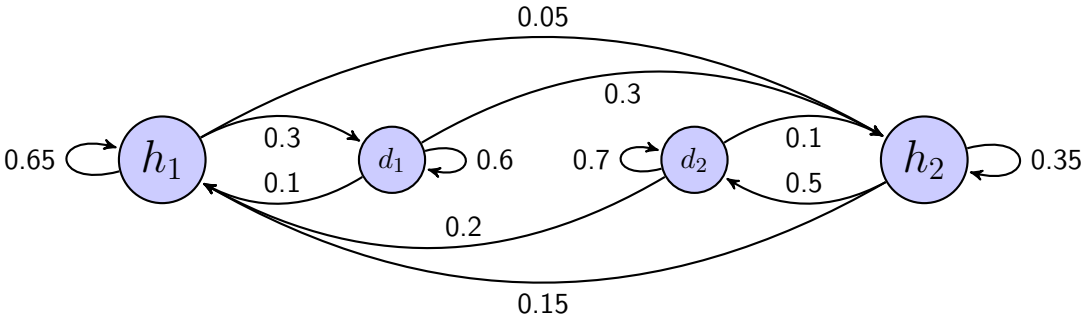


Figure 3.2: A Markov chain model with two hotspots and departure states

These states allow us to represent bikes that are not currently located at a hotspot. Note that, in this model, some transitions are not possible. These are $d_i \to d_j$ and $h_i \to d_j$ for $i \neq j$ transitions. This describes the fact that a bike must transit through h_j before it can arrive at its departure state d_j.

This model allows us to initialize a Markov chain, using some initial state representing the current (the time of querying) location of the bikes, and then predict where the bikes will be located after a number of time steps. From this prediction it is then possible to predict how many bikes should be available, at a given hotspot, in a given timespan, according to our model.

Ordered matrix To ensure that the various parts of this section, as well as the implementation, works consistently, we introduce a set ordering of states

in the matrix. In this representation of the matrix, (x, y) is used to denote $\tau_M(x, y)$. This is done solely to simplify the matrix.

$$\begin{bmatrix} (h_0,h_0) & (h_0,d_0) & (h_0,h_1) & (h_0,d_1) & \cdots & (h_0,h_n) & (h_0,d_n) \\ (d_0,h_0) & (d_0,d_0) & (d_0,h_1) & (d_0,d_1) & \cdots & (d_0,h_n) & (d_0,d_n) \\ (h_1,h_0) & (h_1,d_0) & (h_1,h_1) & (h_1,d_1) & \cdots & (h_1,h_n) & (h_1,d_n) \\ (d_1,h_0) & (d_1,d_0) & (d_1,h_1) & (d_1,d_1) & \cdots & (d_1,h_n) & (d_1,d_n) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ (h_n,h_0) & (h_n,d_0) & (h_n,h_1) & (h_n,d_1) & \cdots & (h_n,h_n) & (h_n,d_n) \\ (d_n,h_0) & (d_n,d_0) & (d_n,h_1) & (d_n,d_1) & \cdots & (d_n,h_n) & (d_n,d_n) \end{bmatrix} \tag{3.4}$$

3.2 Generating the Average Bike Model

Matrices that describe the transition probabilities of Markov chains, should be generated from the GPS locations collected from the bikes in the system. This process is handled by Algorithm 2. The algorithm takes a set of hotspots H and the set of GPS locations G, from which the matrix should be built. For each hotspot $h \in H$ there exists a departure state $d \in D$. Additionally, B represents the set of bikes that is being modeled.

It is presumed that G is implemented using a data structure that allows the retrieval of a single element using a chronological ordering. This will allow us to retrieve the first/last element in the set and to compare elements using the $=$, $<$ and $>$ operators.

Functions We let $g \in G$ represent a single GPS location and say that $G(b) \mid b \in B$ represents the set of GPS locations assocaited with bike b. Both G and $G(b)$ can then be seen as ordered sets of GPS locations.

From this we can provide a definition for the function $G(b, d)$ where d is a representation of a date. Let $G(b) = \{g_0, g_1, \ldots, g_n\}$, then we have that:

$$G(b,d) = \begin{cases} g_0 & \text{if } g_0.date > d \\ g_n & \text{if } g_n.date \leq d \\ g_k & \text{where } g_k.date \leq d \wedge g_{k+1}.date > d \end{cases} \tag{3.5}$$

$L(b)$ represents a collection of $bike \rightarrow state$ references and is used to maintain a list of the last known location of each bike.

3.3 Making Predictions

As a Markov chain $\mathcal{M}$ can be represented with a matrix $\mathbf{M}$ (constructed in Section 3.2), we can use $\mathbf{M}$ to give an estimate of the behavior of the

```
1  CreateModel(H, G, step)
2  Let M be an empty n × n matrix with n = 2|H|
3  foreach b ∈ B do
4  |  g ← G(b).first
5  |  L(b) ← NearestState(H, g)
6  Let d ← G.first.date
7  Let l ← G.last.date
8  while d ≤ l do
9  |  foreach b ∈ B do
10 |  |  g ← G(b, d)
11 |  |  n ← NearestState(H, g)
12 |  |  if n ∈ D ∧ L(b) ∈ D then
13 |  |  |  n ← L(b)
14 |  |  else if n ∈ D ∧ L(b) ∈ H then
15 |  |  |  n ← L(b).departure
16 |  d ← Add(d, step)
17 Normalize M
18 return M
```

Algorithm 2: Creating the model.

```
1  NearestState(H, g)
2  n ← nil
3  mindist = ∞
4  foreach h ∈ H do
5  |  d ← distance(h, g)
6  |  if d < mindist then
7  |  |  n ← h
8  |  |  mindist = d
9  if ¬contains(n, g) then
10 |  n ← n.departure
11 return n
```

Algorithm 3: Creating the model.

average bike in future time-steps. For this we require an initial probability distribution for the average bike. We can then apply $\mathbf{M}$ to this distribution to determine the probability of the average bike being in various states after a number to time-steps. In this section, we will describe how this initial state is established and how we use it to handle predictions.

It should be noted, that the use of the term *"prediction"* in this section refers to determining possible future states of a system, estimated by the Markov chain model.

Notation We will introduce some notation in order to describe the various probability distributions of Markov chains. Let

- X_0 be the random variable representing the possible initial states.
- $H = \{h_0, h_1, \ldots, h_n\}$ be the set of hotspots.
- $D = \{d_0, d_1, \ldots, d_n\}$ be the set of departure states.
- $S = H \cup D$ be the set of all states.
- $P_j(h_i) = \mathbb{P}(X_j = h_i)$ and $P_j(d_i) = \mathbb{P}(X_j = d_i)$

Additionally we let P_j represent a vector with the probability distribution of the random variable X_j. To formalize this definition, we use the following:

$$P_j = \Big(P_j(h_0),\ P_j(d_0),\ P_j(h_1),\ P_j(d_1),\ \ldots,\ P_j(h_n),\ P_j(d_n)\Big) \tag{3.6}$$

3.3.1 Initial Distribution

The initial probability distribution represents the probability of the *average* bike being in each of the states (hotspots or departure states) in the system. Thus we can simply examine all the bikes in the system and identify their current location.

The current location of a bike $L(b)$ can be either a hotspot or a departure state; $L(b) \in S$. The probability of the average bike being in state s will then be equal to the fraction of bikes that are actually in s.

Let B be the set of all active bikes in the system and $B_j(s) \in B$ be the set of bikes in state s at time j, as described by the model. $B_0(s)$ will then be the initial set of bikes in state s and we can represent the probability distribution as the following:

$$P_0 = \left(\frac{|B_0(h_0)|}{|B|},\ \frac{|B_0(d_0)|}{|B|},\ \frac{|B_0(h_1)|}{|B|},\ \frac{|B_0(d_1)|}{|B|},\ \ldots,\ \frac{|B_0(h_n)|}{|B|},\ \frac{|B_0(d_n)|}{|B|}\right) \tag{3.7}$$

Location of a bike Looking at the coordinates of a single bike, we are able to determine if it is inside a hotspot or not. If it is not, we will iterate backwards through the set of known locations of a bike until, we find it located in a hotspot h_i. We then say that the bike is currently in state d_i. If there is no data placing a bike within a hotspot, the bike will instead be said to be in the departure state of the nearest hotspot. In this context, nearest refers to the hotspot that has the lowest minimum distance to the last known location of the bike.

An example With a set of states, such as the one in Figure 3.2, and a total of 16 bikes, we could have a distribution of bikes as follows: 8 bikes are currently located in hotspot h_1. 2 bikes have left it h_1, but have not arrived in another hotspot; they are in departure state d_1. Finally, 6 bikes are located in hotspot h_2, while none are in its departure state d_2.

Such a distribution of bikes would result in an initial probability distribution of the average bike as such:

$$P_0 = \left(\frac{8}{16}, \frac{2}{16}, \frac{6}{16}, \frac{0}{16}\right) = (0.5, 0.125, 0.375, 0)$$

This distribution will be used to determine future probability distributions of the system. In the following section, this example will be revisited, providing such information.

3.3.2 Predicting from the Initial Distribution

Having represented the probability distributions as vectors, we can multiply them with the transitional matrices described by eq. (3.4) on page 26. Such a multiplication would result in a vector with the probability distribution of the individual states after one time-step. In other words:

$$P_{t+1} = MP_t \tag{3.8}$$

Using this definition, we can define P_1 from P_0. To do this, we look at a single value in the vector P_1; $P_1(h_0)$. The remaining values in the vector are defined in a similar manor.

Using the probability matrix from Equation (3.4), page 26, we can determine $P_1(h_0)$. Additionally, we can use Equation (3.7) to further specify the value:

$$P_1(h_0) = \frac{|B_0(h_0)|}{|B|} \cdot \tau_M(h_0, h_0) + \cdots + \frac{|B_0(d_n)|}{|B|} \cdot \tau_M(d_n, h_0)$$

This will then represent the probability, that the average bike is in state h_0 after one time-step. As this models a single average bike, we can multiply the probability with the number of bikes, in order to determine the expected number of bikes in h_0 at time 1:

$$B_1(h_0) = P_1(h_0) \cdot |B|$$

This definition of $B_1(h_0)$ can be simplified by removing the $|B|$ from each term. This gives a simple definition of the value, that can be easily shown to be true for all $B_1(s) \mid s \in S$. Thus we can use this process to retrieve B_1 as a vector. Using a similar argument, we can also show that the same rule applies to all B_j.

$$B_1(h_0) = |B_0(h_0)| \cdot \tau_M(h_0, h_0) + \cdots + |B_0(d_n)| \cdot \tau_M(d_n, h_0)$$

Because of this, predictions can be handled by working directly from the initial distribution of bikes, instead of the distribution of probabilities. We can then directly calculate the expected distribution of bikes in future steps of the Markov chain.

An example Given the distribution of bikes, page 29, and the transition probabilities given in Figure 3.2, page 25, we can estimate the location of the 16 bikes in the example system. The initial distribution of bikes $(8, 2, 6, 0)$ multiplied by the matrix yield the following results:

$$(8, 2, 6, 0) \begin{bmatrix} 0.65 & 0.3 & 0.05 & 0 \\ 0.1 & 0.6 & 0.3 & 0 \\ 0.15 & 0 & 0.35 & 0.5 \\ 0.2 & 0 & 0.1 & 0.7 \end{bmatrix} = (6.3, 3.6, 3.1, 3)$$

3.3.3 Accumulating Probability

The previous section described how to determine the probabilities of bikes being in a specific state. However, the user of OHPAC does not seek to know what the probability of a bike being at a station at a certain time is. Instead the user should be informed about how long he should wait at a hotspot before a bike arrives.

Because of this, we wish to alter the model to reflect a user waiting at a hotspot. We achieve this by altering the transition probabilities of the model. Let $\tau'_M(x, y)$ describe the probability of the average bike transitioning from state $x \in S$ to state $y \in S$, when the user is waiting for a bike at hotspot

$z \in H$. Then we have that:

$$\tau'_M(x,y) = \begin{cases} 1 & \text{if } x = z \wedge y = z \\ 0 & \text{if } x = z \wedge y \neq z \\ \tau_M(x,y) & \end{cases}$$

Using this function, bikes will never leave hotspot z, *while the user is waiting*. This better mimics the process of waiting for a bike; rather than the probability that a bike is present, as bikes continuously leave the hotspot.

Chapter 4

Web Service

The following section will describe the possibilities when designing a web service, arguing for our choice of web service. After the chosen architecture has been presented, the actual implementation will be presented.

4.1 Choosing an Architecture

We want to use a webservice to make the content of our model available to the users of the system. When a user wants to know if a bike is available in their vicinity they will need a way to retrieve information about the bikes in the model.

REST[1] is an architecture that exposes resources and as a bike can be represented as a resource REST will be our choice of web service architecture.

The remainder of this section will describe REST web services, specifically Resource-Oriented Architecture, based on Richardson and Ruby [15], Fielding [16], and HTTP/1.1 Specification[17].

4.1.1 Representational State Transfer(REST)

REST is an architectural style, which provides design criteria for network-based software architectures. Web services adhering to these design criteria are referred to as RESTful web services. These criteria were defined in Fielding [16, Chapter 5] and are listed here:

1. Client-Server
2. Stateless

[1] Representational State Transfter

3. Cache
4. Uniform Interface
5. Layered System
6. Code-On-Demand (optional)

Since REST is only an architectural style, making it possible to implement various correct architectures, we will be following one specific architecture implementation: Resource-Oriented Architecture (ROA), as described in Richardson and Ruby [15].

4.1.2 Resource-Oriented Architecture(ROA)

ROA is an architecture, upholding the REST design criteria. This is done by organizing and accessing resources in a standardized manner, and by following the HTTP/1.1 specification.

In order to ensure that a ROA implementation will comply with the REST design criteria, it will have to adhere to the four concepts[15, Chapter 4]:

- Resources
- URIs
- Representing resources
- Linking resources

along with the four properties:

- Addressability
- Statelessness
- Connectedness
- Uniform interface

.

Resources are a subset of the resource state. A single resource is an entity important enough to be a thing in itself. Each resource should have at least one *URI* (Unique Resource Identifier). URIs ensure that each resource can be uniquely identified, directly handling the *addressability* property.

Representations of resources depends on the desired use. A representation of a specific resource is a collection of any useful information about that resource's state, defined by properties and sub-resources.

Connectedness and linking resources are highly dependent on URIs. As each resource is uniquely identified, any resource containing sub-resources or needing to refer to other resources, can do so by linking to its URI.

Statelessness is handled by providing, if any, state-relevant parameters on each request. This makes the server independent of clients and previous requests, as it is up to each individual client to handle its own application state (not to be confused with the server's resource state) and providing it when needed.

Uniform interface is where the HTTP/1.1 specifications come in. HTTP provides a set of methods for retrieving and modifying resources. As the service provided by OHPAC will only allow users to retrieve data we will on be employing the GET method. All resources can then be retrieved using a GET request (see Section 4.2.2 for a specification). Any feedback needed to be given to clients from the server is handled through HTTP status codes (e.g. '200 OK' and '404 Not Found').

4.2 Web Service API

The web service API was implemented using ASP.NET Web API 2.3[18], which is an implementation of the Model-View-Controller architecture. In the architecture of OHPAC, the web service is defined as a component of its own (see Section 5.5).

As described in Section 4.1.2, ROA sets the ground rules for designing the API. These criteria are therefore upheld by the API.

4.2.1 Model-View-Controller[19]

This is an architectural template that is used to separate responsibility of an application into three parts; the model, the view, and the controller. A diagram, displaying the pattern and its internal dependencies, can be seen in Figure 4.1.

Model contains the model of the application domain and takes care of fetching data from the underlying layers and making it available to the view and the controller.

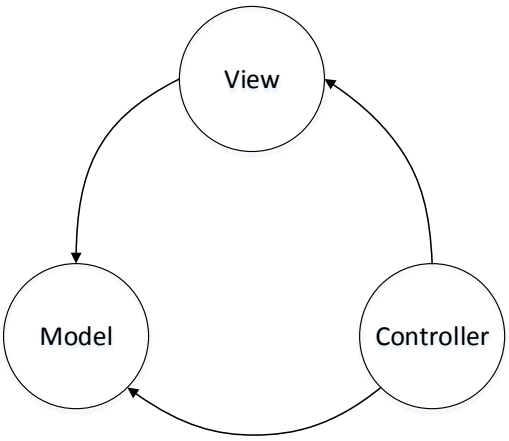


Figure 4.1: The MVC design pattern.

View displays the model to the user. In our case the view is the serialized resource representation, that the user obtains by performing requests against the API. Per default, this is `text/xml`, however, by adding the `Accept` header[17, Section 14], this can be set to `application/json`.

Controller handles the interaction with users. Based on user input, the controller works on the model and selects which view is needed to display the data.

4.2.2 Resources

This section contains the resources designed and implemented in the web service. Resources in the implementation are objects, containing named properties and sub-resources. Sub-resources are represented as simple objects, containing only its relative URL. Each resource is described following this structure:

- **URL:** the URL of the resource
- **Properties:** a property which can be accessed from this resource
- **Resources:** the resource(s) which can be accessed from this resource
- **Methods:** HTTP methods which can be used on that resource
- **Responses:** HTTP status codes which could be returned by this resource

4.2.3 The Resources Implemented:

URL: `/` **Properties:** `version`

Resources: `availablebikes, hotspots, predictions`

Methods: `GET` **Responses:** `200 OK`

The root path serves as the entry point for the API. It has a single property; version, to be used for asserting that the API has changes that implementations should react to. More importantly, it contains a sub-resource for each main resource.

URL: `/availablebikes` **Properties:** `count`

Resources: `List of availablebikes`

Methods: `GET` **Responses:** `200 OK`

This resource contains a list of all `availablebike` resources, along with a count of how many there are.

URL: `/availablebikes/{bikeId}` **Properties:** `latitude, longitude`

Resources:

Methods: `GET` **Responses:** `200 OK, 404 Not Found`

This is the actual `availablebike` resource. In case a client tries to request a bike that is not available, it will be met with a `404 Not Found` response.

URL: `/hotspots` **Properties:** `hotspots`

Resources:

Methods: `GET` **Responses:** `200 OK`

This resource returns a list of all the hotspot resources in the system.

URL: `/hotspots/{hotspotId}`	**Properties:** `coordinates`
Resources:	
Methods: `GET`	**Responses:** `200 OK, 404 Not Found`

This resource returns the points the make up the convex hull of a single hotspot. In case a client tries to request a hotspot that is not available, it will be met with a `404 Not Found` response.

URL: `/prediction`	**Properties:**
Resources:	
Methods: `GET`	**Responses:** `200 OK`

This is the prediction resource. When a client wants to know when a bike arrives at a hotspot this resource can be used.

Chapter 5

Architecture

This chapter illustrates the system architecture and describes the role of each component. It also contains an analysis of why the architecture is scalable.

The system architecture is introduced to ease maintenance and ensure scalability. The architecture consists of four macro components and a database, as can be seen in Figure 5.1. The macro components are: Shared, Location Service, Model Agent, and Web Service.

5.1 Database

The database used is a MySQL database, and is where all our data is stored. The design of the database can be found in Appendix B. It is not included in the report as its structure is rather trivial and only serves the purpose of having a simple and fast method of storing/retrieving large amounts of data.

5.2 Shared

The task of the shared macro component is to provide extended capabilities to all other macro components. The component provides a shared structural representation of data and a uniform set of database connection methods.

Data Access All SQL statements and communication with the database are encapsulated here providing the upper layers a communication protocol to the database. Additionally this layer provides an abstraction of the actual communication with the database as well as an abstraction of the database design.

Data Transfer Object This layer provides a collection of shared data types, making it possible for all the macro components to treat the data uniformly. The types within this layer provides methods for retrieving data from the database and converting it into elements of the types in the Data Transfer Object layer.

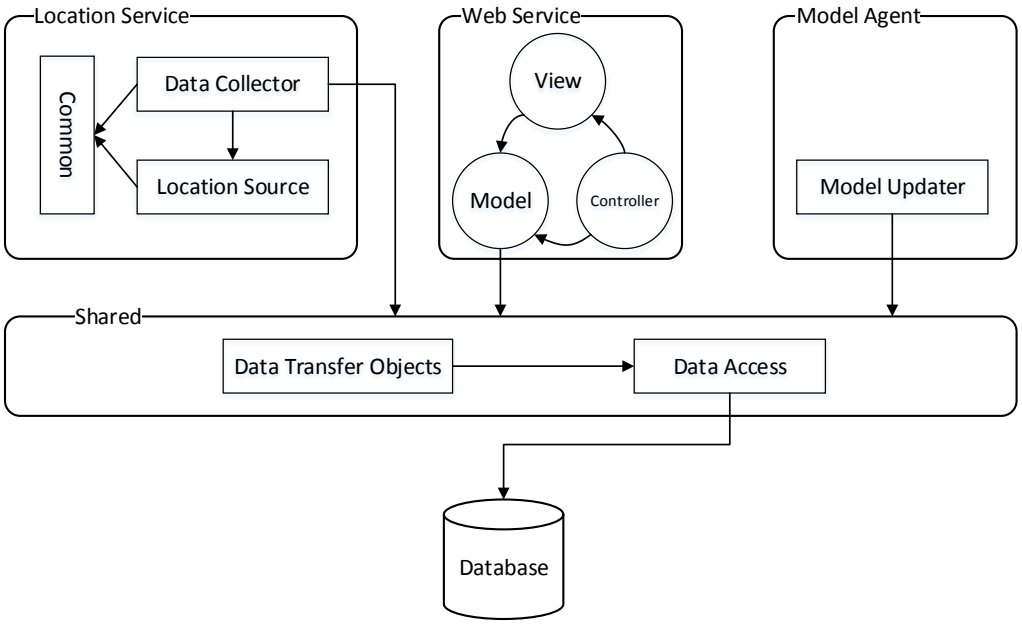


Figure 5.1: The architecture of the system

5.3 Location Service

The location service component processes and stores location data directly from a data source. In other words it serves the purpose of loading gps location data for the other components (through the shared macro component).

Location Source Provides an implementation of a data-source from which data can be retrieved. The common layer in provides a shared structure for communication between the location layers.

The layer is structured such that the Location Source layer can easily be replaced with another source of data[1].

[1]This is used for providing data when testing the system, as will be described in section 6.1.

Data Collector Using a data source defined in the location source layer, this layer gathers the gps location data and stores it in the database using the shared macro component.

5.4 Model Agent[2]

The task of the model agent macro component is to generate and update the hotspot and Markov chain model in the database (through the shared macro component).

Model Updater layer This is where the generation of the model is handled, which works as follows:
Clusters are created based on the GPS locations in the database. This is done using DBSCAN, as described in Section 2.1.5.4. The convex hull of each cluster is determined, by using Graham Scan, and these are saved to be used as hotspots. The Markov chain matrix is now created with these hotspots, and their departures, as was described in Section 3.2.

Finally, since we have no further use of it, all already modeled GPS locations are truncated from the database, with exception of the single latest location of all bikes.

5.5 Web Service

Web Service is the macro component containing a Model, View, and Controller. The task of the macro component is to provide a web service, in order for users to interact with the system. The model, view, and controller of the web service were introduced in Section 4.2.1.

5.6 Scalability

Scalability is an important factor when designing a web service. In this project the main potential growing assets are users and bikes.

Users are important, to take into account, since there is a high uncertainty in regards to the number of users, which could lead to performance issues if not addressed.

[2] An agent is defined as a program that invoke a given task, when a condition is satisfied, then returning to a sleeping state, until the condition is satisfied again.[20]

Additionally, the number of bikes must be increasable without causing problems in the system. If not addressed, it could lead to storage issues due to the increase in number of GPS locations and increase in the time required to build the models. A high number of resulting hotspots will also result in slower response times for the users due to the increased number of matrix multiplication required.

The easiest way to address these issues is to scale out (adding new hardware) or scale up (enhance existing hardware) [21].

The system is scalable due to the way the macro components are divided. By dividing the system into macro components with their own tasks, it would be easy to add more hardware. It should further be easy to implement additional functionality due to the Shared macro component.

Since each macro component only needs a connection to the Shared macro component, it should be possible to migrate or add new macro components to other machines - as long as it can connect to the Shared macro component. Thus it is possible to scale out which increases the performance of the system.

Chapter 6

Test

In order to test how well our system works we have chosen to perform an integration test of some of the integral parts of the system, the clustering and the predictions. Because we do not have access to bikes with GPS receivers mounted we have created a simulation that will mimic the real world as close as possible. This chapter will first explain how we have simulated the real system and will afterwards present the performed integration test.

6.1 Data Input Simulation

In the running system, locations will be retrieved from a GPS receiver attached on the bikes. Because we do not have bikes with GPS receivers at our disposal, we need to create a realistic simulation of the behavior of the system.

6.1.1 Point Generation

Because we do not have access to a running system with X amount of bikes sending locations, we have created a simulation of the real system. This system uses Google Directions to create pseudo-realistic routes and returns the locations at an interval. In order to generate points that are as realistic as possible we have used the following approach for generating GPS points:

Create routes with Google directions The Google Directions API [22] has been used to create realistic looking bike routes. The API has a variety of settings to restrict the results, including a setting to only get bike routes, which is set when generating routes to be used for simulation. The result

is a route, structured by a number of "steps".[1] Steps are the points where the route changes direction and are the points that would be used to print a directions instruction for the route. The result of a query to the Google directions API is a route that can be used by a human to find his way to his destination.

Generate intervals The route generated by Google Directions only has points at the steps, as described above. Our assumptions about how GPS receiver updates are obtained, is with a fixed 5-minute interval. In order to make the generated routes fit with the data obtained by a GPS receiver, the simulated points must be at the corresponding 5-minute intervals. To achieve this, points are added onto the generated route, separated by a 5-minute interval. This way, the final route is consisting of 5-minute interval points, instead of the generated step points.

To further the realisticness, the generated GPS points query time is also delayed randomly, by up to 30 seconds. Lost GPS locations, which could also be a possibility in an actual system, are not taken into account.

Perturb points to simulate precision As one of our assumptions about GPS receivers are that they are not 100 % accurate, and they will therefore not provide accurate locations, we will have to take this into account when generating points with Google Directions. To simulate this inaccuracy, every generated point will be perturbed by a random value, corresponding to an expected inaccuracy. This expected inaccuracy should correspond to GPS inaccuracy, which is up to 15 m according to Garmin [23].

6.2 Integration Test

In order to test that our components interact together according to the specifications, we have chosen to perform integration testing of the system.

Integration testing has the purpose of testing that two units work together. Traditionally, integration testing is done by integrating two modules at a time, either from the top of the dependency tree or from the bottom. There exists a variant, in which the whole system is tested together, instead of pair-wise, called big bang integration testing. We have chosen to perform our test with this method, due to time limitations [24].

[1] Actually a route contains a number of "legs" which contains a number "steps". A leg is a part of a journey and a new leg will be created when changing means of transport or at waypoints. For our purposes, all routes will always contain only one leg.

6.2.1 Test Setup

The purpose of the tests we have designed is to check if the system can produce sensible predictions for the bikes in the system.

In order to generate test data, we have created a method for simulating real world data. The simulator gets a list of destinations (in the form of addresses) and a list of probabilities as input, making it possible to adjust the probability of the system deliberately choosing certain destinations. This makes it possible to create a data set, where it is possible to predict the expected result.

What to test We want to test if the system works as specified in the problem statement. The problems we want to focus on in testing are 2: *Too few stations* and 5: *No way of knowing when a bike will arrive*. The full problem statement can be found in Section 1.4. We want to test the following parts of our system in order to test the two problem statement items:

- Finding hotspots
- Markov Chains

We test the hotspot creation in order to check if hotspots are generated where there is bike activity. This will check if the created hotspots will be able work as a stand in for the old stations. This corresponds to item 2 in the problem statement.

We test the Markov chains in order to figure out if our method of generating Markov chains give proper values for using when predicting the behavior of the bikes. This corresponds to item 5 in the problem statement.

6.2.2 Finding Hotspots

All predictions are calculated based on the generated hotspots. Misplaced hotspots can therefore skew the results, decreasing the precision of the predictions. Thus, we want to test whether or not the hotspots are generated as expected.

Test data For this test, we have constructed a data set, which consists of two addresses:

- Nytorv 1, 9000 Aalborg
- Nytorv 15, 9000 Aalborg

and their corresponding GPS coordinates. We will then generate hotspots based on this data, and compare the center of these hotspots, with the addresses from the constructed data set. Because the addresses are the only place where the bikes stand still, and the data is generated with noise there should form a cloud of GPS locations around the address which will then be clustered.

DBSCAN parameters The clustering algorithm chosen[2] expects two parameters, *eps* and $MinPts$. Because real data is not available the chosen values for the parameters are based on the following.

The $MinPts$ parameter was set to 4 as suggested by Pang-Ning et al. [11, Page 529]. For the other parameter, *eps*, 15 meters was chosen. This value was chosen because of the perturbation, as described in Section 6.1.1, which was added to all simulated points. When clustering, we are ensured that the bike locations which are based on the same address will be grouped, as the perturbation does not exceed *eps*.

Result The result has been visualized and can be seen in Figure 6.1. The figure shows as expected, two hotspots and a path of GPS locations inbetween. The test verifies that all GPS locations that should be in a hotspot, actually are (as illustrated by the circles in Figure 6.1). This means that the clustering is accurate and works as expected. One should keep in mind that the DBSCAN parameters, which worked with the constructed data set, may need to be calibrated when used on real data.

6.2.3 Predictions

The users rely on the predictions of bike usage, as stated in Section 1.4.3, item 5. It is therefore important to test how well the predictions work.

Test data To test this we create a number of controlled data sets. The data sets are controlled by assigning probabilities of choosing the destinations. We then know where to expect the probabilities to be high.

The data is constructed in such a way that every time a bike arrives at a hotspot it will wait for 12 minutes, before going to the next destination. This ensures that there is registered at least two points within a hotspot.

The addresses are placed in such a way that it should not be possible to go from one hotspot to another in a single update interval. This means that it should only be possible to get to a hotspot from a departure state, and

[2]DBSCAN explained in Section 2.1.5.4

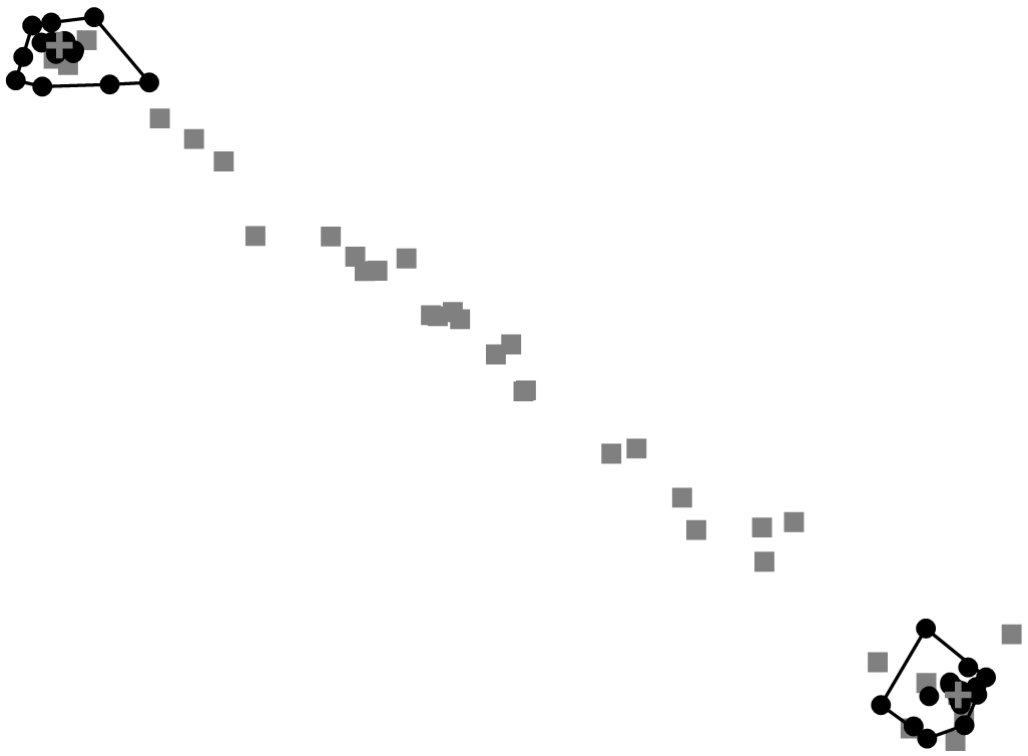


Figure 6.1: The clustering test results. The crosses indicating the coordinates of the two given addresses. The squares representing GPS points received while a bike is moving. The circles representing when a bike stands still. The line connecting the circles is the cluster represented by a polygon(a hotspot). Keep in mind that the squares inside a cluster are not part of a cluster.

not directly. This means that every hotspot row in the result should contain only two numbers, the transition to itself and the transition to its departure state.

In order to display results, tables depicting the generated Markov Chain matrix are displayed. h indicates a hotspot while d indicates a departure state. The number in (d_1, h_2) indicates the probability in percentages of transitioning from departure state 1 to hotspot 2.

6.2.3.1 Even distribution

The simplest test case is when the generated data has equal probability of choosing every destination. In this case we would expect the probabilities of a bike arriving at every destination to be the same.

The data was generated to simulate a 15 hour period with 100 bikes and 4 possible addresses:

- Nørresundby Torv, 9400 Nørresundby
- Nytorv, 9000 Aalborg
- Borgmester Jørgensens Vej 5, 9000 Aalborg
- Selma Lagerløfs Vej 300, 9220 Aalborg Ø

The results for the test can be seen in Figure 6.2.

In the result it is seen that some inaccuracies have occurred since there are some entries with values that, although very low, are not zero as expected. An example of this inaccuracy is (h_1, h_2) which is 0.3 even though it should be impossible to get from h_1 to h_2 in only one step.

The important numbers are those that tell how a bike act when in a departure state. These are the numbers that indicate how the routes are expected to be chosen. These numbers are setwise marked with a color in the table.

Each of these sets of numbers indicate the distribution between destinations. As the distribution of the generated data was set to be equal, we would expect these numbers to be equal as well.

The numbers are normalized such that they add up to 100 % and can be seen on fig. 6.3 where the expected results are in the left table and the actual results are in the right table.

As it can be seen from the results, the numbers look very reasonable. One would expect the numbers to converge to 33.3% as the set of test data is increased.

%	h_1	d_1	h_2	d_2	h_3	d_3	h_4	d_4
h_1	59,6	40,1	0,3					
d_1	0,1	74,6	8,6		8,3		8,5	
h_2	0,2		59,6	40,1				
d_2	9,8		0,1	68,4	11,4		10,4	
h_3			0,1		59,2	40,7		
d_3	12,4		13,4		0,2	60,4	13,7	
h_4							60,0	40,0
d_4	7,7		6,8		6,9		0,0	78,5

Figure 6.2: Results of the test with an even distribution. A colored set of numbers indicates the probabilities of a departure state going to each of the other hotspots.

	h_1	h_2	h_3	h_4
d_1		33.33	33.33	33.33
d_2	33.33		33.33	33.33
d_3	33.33	33.33		33.33
d_4	33.33	33.33	33.33	

	h_1	h_2	h_3	h_4
d_1		33.85	32.68	33.46
d_2	31.01		36.08	32.9
d_3	31.39	33.92		34.68
d_4	35.98	31.78	32.24	

Figure 6.3: The expected proportions to the left and the resulting proportions to the right

6.2.3.2 Uneven distribution

Another test case is to weight one destination higher than the others. In this case we would expect the probability of a bike arriving at this particular destination to be higher than all the other probabilities.

The data for this test was created in the exact same way as the data for the even distribution, a 15 hour period with 100 bikes and the same 4 possible addresses. The only difference was that the distribution was changed such that one address had 50 % chance of being chosen, another had 20 % and the two remaining had 15 %. The resulting Markov chain after this simulation can be seen on Figure 6.4.

The order of the hotspots does not correspond to the order in which the percentages were presented above. However we have determined the association by examining the GPS locations associated with the generated hotspots. The ordering of percentages, using the hotspot order h_1, h_2, h_3, h_4, is $15\%, 50\%, 15\%, 20\%$.

In Figure 6.5 we present two tables that describe these values in relation to transitions. The expected results are in the left table and the actual results are in the right table. The values are normalized so it adds up to 100 %.

These results are a little more difficult to judge based on the tables, but the numerical difference are at most 2.45 percentage points (from d_4 to h_1) which is regarded as a reasonable result.

%	h_1	d_1	h_2	d_2	h_3	d_3	h_4	d_4
h_1	60,4	39,6	0,1					
d_1	0,1	59,4	24,3		6,9		9,4	
h_2	0,2		59,3	40,4				
d_2	7,4		0,2	74,8	7,6		10,1	
h_3					59,9	40,1		
d_3	4,0		11,4			80,1	4,5	
h_4	0,1						59,0	40,9
d_4	7,4		22,8		5,7		0,1	64,1

Figure 6.4: Results of the test with an even distribution. A colored set of numbers indicates the probabilities of a departure state going to each of the other hotspots.

	h_1	h_2	h_3	h_4
d_1		58.82	17.65	23.53
d_2	30		30	40
d_3	17.65	58.82		23.53
d_4	18.75	62.5	18.75	

	h_1	h_2	h_3	h_4
d_1		59.85	17.00	23.15
d_2	29.48		30.28	40.24
d_3	20.10	57.29		22.61
d_4	20.61	63.51	15.88	

Figure 6.5: The expected proportions to the left and the resulting proportions to the right

Chapter 7

Conclusion

This project had the goal of improving the current Aalborg City Bike system. Our solution involved equipping all bikes with a GPS receiver. We could then develop a software system that helped solve the problems, stated in 1.2.1, as well as the new problems that arose, stated in 1.4.4.

We have in this report described how we successfully solved these problems and, as stated in Chapter 6, both our clustering and prediction methods works as intended. Thus the project is considered to fulfill the problem statement.

It should be noted that we have only tested on simulated data. Even though the simulated data is carefully crafted to mimic real world data, we cannot ensure with a 100% certainty, that real world data always will fall inside the simulated parameters. Thus we cannot conclude that the system would work as expected in a real world setting.

Chapter 8

Future Work

This chapter will reflect on how the developed system could be improved.

8.1 Optimized Data Structure

In Section 2.1.4 we mention that we would like a partitional clustering because we do not need hierarchies for hour purpose. While this is true it has since come to our attention that a hierarchical clustering would have made it easier to check whether bike is within a hotspot. The current implementation uses a list, which means that every element is accessed in worst case. A hierarchical clustering would not have this problem because it would be possible to traverse the hierarchy when looking for a point.

Using a data structure on top of the DBSCAN clustering algorithm would be a possible improvement to the current implementation. R-trees[25, Section 25.3.5.3] is a data structure that could be chosen for that purpose.

8.2 Model Updating

There are some issues regarding the process of updating the model and how often it should be updated. There are many possibilities regarding this, because it is a rather complex matter. Below is a list trying to show the factors that influence the model:

- Time of day
- Weekday
- What kind of day (workday, holiday, city events, etc.)

- Weather conditions
- Etc.

A simple model would be to update it every week, and don't take the different factors into account. An example of a more complex model is to partition the day into mornings, afternoons and nights. And for each of these have a model for holidays and working days.

To find the best model one needs a collection of real data for testing which model gives the best results.

8.3 Usage statistics for Aalborg Kommune

One of the requests from the interview with Aalborg Kommune (see Section 1.1.1) was that they wanted certain statistics about bike usage.

Some of the data is already available in the current implementation, as bike locations are already being continuously collected. We would add a data-warehouse to extract the newest bike location updates and form its own data, so that it could be easy to query on it, queries such as those mentioned in Section 1.1.1.

These statistics could be used for improving the Aalborg City Bike system, as well as the software system. The statistics could improve/help the following processes:

- Where to add more bike lanes
- Improve cycle infrastructure
- When to update the model(s)
- etc.

8.4 API Access

Adding different queries for Aalborg Kommune, that should not be accessible to normal users, would require the API to be re-designed considering security measures. Therefore it would be rather practically to introduce different user access levels. For instance there could be the following access levels:

- Administrator (facilitator of the API - access to everything)
- User (standard access)

- Bike maintainer (access to e.g. all bikes location)

This change would be added in the `Controller` layer in the `Web Service` component.

Appendix A

Interview 10-10-2014

A.1 Interviewees

- Brian Høj - Responsible for Aalborg City Bike.
- Jesper - Employed with Aalborg City Bike.
- Anne Mette - Part of launching the ACHIMEDES project.

A.2 Interview Summary (in Danish):

- Hvilket overblik har I over cyklerne? lokation, forbrug, antal, osv. Hvordan finder I ud af om der skal flere cykler i brug, og hvor cyklerne skal placeres? Der er i øjeblikket ingen kontrol med cyklerne og dermed heller ingen data omkring brugen af cyklerne. Den eneste information er gehør, altså hvad de selv observerer ude i byen, og de opringninger de modtager fra borgerne der melder en cykel forsvundet.

- Får I nogle klager? Hvis ja, hvad lyder klagerne på? De klager de oftest får er at der ikke er nok cykler tilgængelig. De observerer også selv at der er helt tomme stationer.

- Har I nogle statistikker/data om systemet vi kan få adgang til? Der er ingen data overhovedet. Kun anelser om hvordan cyklerne bliver brugt. En af de interviewede observerede at der holdt mange bycykler ved skoler og universiteter, hvilket de fortolker som om at der er mange der bruger cyklerne som privatcykler. Hvad de fandt interessant er at få mere data om brugen af cyklerne. Dette kunne fås v.h.a. GPS, ruter, strækning, stationer, tid, stilstand, samme bruger?

- Kan vi bruge det samme layout som http://www.aalborgbycyklen.dk m.h.t. Copyright? De mener at det er dem der ejer siden, og vil lige undersøge om ikke også de har rettighederne.

- Hvad er jeres egentlige behov? Hvilke mål har i med projektet og hvilke restriktioner er blevet påsat jer for at opnå disse mål? De er usikre på om det er turister eller borgere der er målgruppen for brugen af cyklerne, men de ligger vægt på at den ikke bliver brugt som privatcykler. For eksempel hvis en person ønsker at bruge den resten af året ud. Den større politiske opmærksomhed på bycykler sammen med tendensen til at der bliver afsat flere penge af til sådanne projekter leder dem til at konkludere at det er et system der skal udvides på, og gøres mere moderne.

- Har i nogle udvidelser/videreudviklings planer til projektet? De er interesseret i at tracke cyklerne med GPS og indsamle data om cyklernes brug, men de har ikke nogle konkrete planer om hvad der skal laves og hvordan man sikrer at GPS'en ikke bliver stjålet fra cyklen.

- Har i haft nogle tanker om reservering af bycyklerne? Der var ingen tanker i den retning med systemet, men idéen om at booke en cykel virkede interessant for dem. De var åbne overfor muligheden, men kunne også ulemperne ved et sådant system, da det kan blive for restriktivt i forhold til brugeren. For eksempel, hvad sker der hvis brugeren slet ikke dukker op til sin booking? Hvor længe i forvejen skal man kunne booke en cykel? Hvornår må systemet konstatere at brugeren ikke dukker op? De ser brugen af cyklerne som en spontan handling fra brugeren og måske ikke så meget planlagt.

- Har i lyst til at være vores kunde? Højst 3 møder mere i løbet af semester. Ja, de var villige til at lave en forsøgsperiode på en uge eller mere, hvor eksempelvis ti cykler blev udstyret med GPS, så man kunne studere cyklernes færden. Aftalen lyder, at vi tre grupper snakker sammen om vores videre arbejde og kontakt med dem som kunder.

A.3 Interview notes (in Danish):

- Oprindeligt 200 cykler men kun 140 cykler ude nu. Resten er i depot.
- Et privat firma der står for driften. En dyr ordning. Der mangles sponsor indtægter.

- Ingen GPS på cyklerne. Simpelt system.
- Har overvejet at se på noget mere avanceret. Måske nogle nye cykler? tænker de Århus, har andre cykler hvor de ikke har problemer med sponsorer (store reklamer på hjulene). Færre driftsomkostninger
- Der forsvinder ikke mange cykler. (ikke noget særligt)
- Cyklerne samles ind i oktober. De finder allesammen, men de kan godt finde mærkelige steder. De kommer ud igen først i april.
- De har problemer med at cyklerne ikke bliver fundet af sig selv men rapporteret forsvundet af brugerne.
- De kunne godt tænke sig at have kontrol af cyklerne (med GPS har de overvejet)
- De vil også gerne vide mønstrene af brug af cyklerne.
- Der findes ingen data for brug af cyklerne. Det eneste der sker ved cyklerne er vedligeholdelse.
- De vil gerne have turister til at bruge cyklerne
- Målgruppen er: turist eller borger (ikke så meget hvem der er, men at den ikke bliver brugt som sin private cykel)
- GPS ideen lyder spændende for dem.
- De er bevidste om at det er en dårlig ordning.
- De er interesseret i data og statistikker om hvordan det bliver brugt.
- Booking er også interessant, men GPS delen er særlig interessant.
- Hvor længe holder de stille? Hvor meget er de i brug?
- Deres egne observationer er at der holder mange ved skoler og universiteter og kan måske konkludere ud fra det at de bliver brugt til privat brug.
- De tror mere på de spontane brug af bycyklerne, og mener at det bliver meget restriktivt med booking.

Appendix B

Database Design

To store the data we use a MySQL database.

When location data is collected and processed, it is stored in the `gps_data` table. This table contains all information necessary for storing a GPS point as well as the field `hasNotMoved`, which indicates the bike is standing still, as the location has not changed since the previous point was inserted into the database. This has been done in order to reduce redundant data in the database. The `gps_data` table can be seen in Code example B.1.

```
1 CREATE TABLE gps_data (
2     id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
        PRIMARY KEY,
3     bikeId INT UNSIGNED NOT NULL,
4     latitude DECIMAL(10,8) NOT NULL,
5     longitude DECIMAL(11,8) NOT NULL,
6     accuracy TINYINT UNSIGNED NOT NULL,
7     queried DATETIME NOT NULL,
8     hasNotMoved BOOL NOT NULL DEFAULT FALSE
9 );
```

Code example B.1: Table for GPS_data

To query information about a single bike, and keep track of the bikes currently available in the system, we use the `bikes` table. This table is very simple and can be seen in Code example B.2.

```
1 CREATE TABLE bikes (
2  id INT UNSIGNED NOT NULL UNIQUE PRIMARY KEY
3 );
```

Code example B.2: Table for bikes

The hotspots created by the DBSCAN algorithm are also stored in the database. A hotspot is described by a convex hull of the points that make up the area of the hotspot. This convex hull is stored as binary data (a blob) in the `hotspot` table, which can be seen in Code example B.3.

```
1 CREATE TABLE hotspots (
2     id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
        PRIMARY KEY,
3     convex_hull BLOB NOT NULL
4 );
```

Code example B.3: Table for hotspots

The Markov chain matrix is also stored as binary data, as seen in Code example B.4.

```
1 CREATE TABLE markov_chains (
2   id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
     PRIMARY KEY,
3   mc MEDIUMBLOB NOT NULL
4 );
```

Code example B.4: Table for Markov chains

Bibliography

[1] CIVITAS. Archimedes. http://www.civitas.eu/content/archimedes, 2014. Online; Accessed 12/12/2014.

[2] Aalborg Bycyklen. Hvem står bag bycyklen? http://www.aalborgbycyklen.dk/default.aspx?m=2&i=47, 2014. Online; Accessed 23/09/2014.

[3] Anne Marie Lautrup Nielsen. Integreret cykelplanlægning i aalborg for pendlere, turister og børn. *Artikler fra Trafikdage på Aalborg Universitet*, page 9, 2012. Online; Accessed 23/09/2014.

[4] Aalborg Kommune. Cykelhandlingsplan 2013. http://www.e-pages.dk/aalborgkommune/793/, 2013. [Online; Accessed 21-10-2014].

[5] Aalborg Kommune. Vurdering af højklasset pendlerrute på hobrovej. http://referater.aalborgkommune.dk/Pdf.aspx?pdfnavn=16956990.PDF&type=bilag&id=9271, 23-08-2011. [Online; Accessed 21-10-2014].

[6] Aalborg Cykelby. Kampagner & events. http://www.aalborgcykelby.dk/aalborg-cykelby/kampagner-events, 11-09-2014. [Online; Accessed 21-10-2014].

[7] Bycyklen. Bycyklen. http://bycyklen.dk/en/, 2014. [Online; Accessed 26-09-2014].

[8] Bycyklen. Gobike betingelser. http://bycyklen.dk/media/26398/Gobike-betingelser-010814-DK.pdf, 2014. [Online; Accessed 20-10-2014].

[9] Citi Bike NYC. Citi bike nyc. https://www.citibikenyc.com/, 2014. [Online; Accessed 26-09-2014].

[10] FollowMee. Followmee gps tracker. https://www.followmee.com/. [Online; Accessed 18-12-2014].

[11] Tan Pang-Ning, Michael Steinbach, Vipin Kumar, et al. Introduction to data mining. In *Library of Congress*, 2006.

[12] Wikipedia. Dbscan. http://en.wikipedia.org/wiki/DBSCAN#Algorithm, 2014. [Online; Accessed 18-12-2014].

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

[14] Gordan Žitković. Introduction to stochastic processes - lecture notes. http://www.ma.utexas.edu/users/gordanz/notes/introduction_to_stochastic_processes.pdf, 2010. [Online; Accessed 12-12-2014].

[15] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Series. O'Reilly Media, Incorporated, 2007. ISBN 9780596529260.

[16] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, 2000. [Online; Accessed 10-10-2014].

[17] The Internet Engineering Task Force (IETF). Hypertext transfer protocol – http/1.1. https://tools.ietf.org/html/rfc2616, 1999. [Online; Accessed 21-10-2014].

[18] ASP.NET. Asp.net web api. http://www.asp.net/web-api, 2014. [Online; Accessed 24-11-2014].

[19] Asp.net mvc. msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx. Accessed: 2014-11-27.

[20] Definition of software agent. http://www.techopedia.com/definition/24595/software-agent. Accessed: 2014-11-27.

[21] Maged Michael, Jose E Moreira, Doron Shiloach, and Robert W Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[22] The google direcitons api. https://developers.google.com/maps/documentation/directions/. Accessed: 2014-11-24.

[23] Garmin. Garmin gps. http://www8.garmin.com/aboutGPS/. Accessed: 2014-11-27.

[24] Paul C. Jorgensen and Carl Erickson. Object-oriented integration testing. *Commun. ACM*, 37(9):30–38, September 1994. ISSN 0001-0782. doi: 10.1145/182987.182989. URL `http://doi.acm.org/10.1145/182987.182989`.

[25] S. Sudarshan Abraham Silberschatz, Henry F. Korth. *Database System Concepts, SKS 6th edition:*. McGraw-Hill, 2011. ISBN 9780073523323.