Traffic Analysis of Intersections Based on GPS Data

BY
Group d403a
Algimantas Bačkys, Kim Sø Pedersen, Mindaugas Remeika & Casper Nicolaj Sparre
Supervisor - Kristian Torp

1. Abstract

In this paper we develop methods for analysing traffic congestion in light-controlled intersections, based on GPS data collected from more than a 100 vehicles.

One of the challenges consist of extracting relevant information from the data, which is stored in a database. This is done through selective filtering of the data, before we start analysing it.

Finally we analyse the filtered data, to estimate congestion in a specific intersection in Aalborg. We present the findings through the use of graphs and diagrams, to highlight the relevant facts, in a way which should quickly identify any problems in the intersection.

Our goal is to show that GPS data can be applied to light-controlled intersection analysis. In conclusion we show that it can, and the results corresponds very well with outcome that we would expect, based on our own knowledge of traffic in Aalborg.
2. Introduction

Traffic analysis and optimisation of traffic flow are major concerns in many countries. Loss of time due to traffic jams, can mean a loss of revenue for companies, taxes for the state, and added fuel costs for drivers. For example: “Texas Transportation Institute (TTI) estimates that congestion is costing Americans more than $78 billion a year. Urban travelers are delayed in rush hour traffic nearly 40 hours a year.” [6]

This paper looks at integrating GPS trip data, gathered by GPS devices placed in vehicles, into the analysis of traffic flow. Specifically we will focus on analysing traffic light-controlled intersections, in an urban environment. This choice is made because such intersections have the highest traffic flow and the highest overall impact on traffic flow is based on the alignment of these intersections.

The actual analysis will revolve around one specific intersection in Aalborg (Denmark), that of Kong Christians Allé, Óstre Allé, Hobrovej and Vesterbro, in order to keep the work focused. However we develop techniques which easily could be generalisable to a broader scope. We create an automatic method (software) which would be able to analyse the input (GPS data and intersection segment numbers) and produce results both in text and a graphical way. The main interest of this project is 4-legged intersections, which are very common in urban areas.

![Figure 1. Chosen Intersection viewed from Google Maps](image)

This intersection (Figure 1) is chosen to be our focus as we develop the analysis tools. The main reason we chose this one, is because it is one the most travelled intersections within our data set. Additionally there are four incoming lanes in each direction, two of these lead straight through the intersection, and the remaining two are dedicated for right and left turns, which means it is a good example of a symmetric and common high traffic-flow intersection.

We are going to calculate the vehicles’ driving directions so we can determine the most busy directions of this intersection. Furthermore, we will find out average time to get through the
intersection in the different directions, average speed crossing it and the average waiting
time in the intersection, to evaluate congestion. A graphical representation of results will be
included in the intersection analysis.

The remaining parts of this paper, is structured as follows. Related Work - here we briefly
introduce a few related projects, who in similar fashions have used GPS data to analyse
traffic.

Development - Here we present some of the initial work done, to form the basis of this
project. That includes database setup, and data processing/filtering, as well as an
introduction to KML.

Implementation - This section presents the actual implementation. It states the actual
software used, discussed some problems involved in working with real-world data, and lays
out the algorithms the produces the analysis results, in the form of pseudo code.

Results - Here we presents the results of the intersection analysis, and briefly discusses the
outcome, which generally corresponds with our initial expectations for the particular
intersection.

Finally we present our conclusions. Additionally you will find acknowledgements and
references at the end of the paper.

3. Related Work

A number of studies have looked at how to incorporate GPS data from vehicles into traffic
analysis and digital map construction. Common for these, are that they often focus on one
specific stretch of road, like a freeway or highway leading into a city or some other highly
congested road. Furthermore the data used for these projects, often originates from fairly
limited pools of vehicles. Either it is collected by one or a small group of vehicles, specifically
for a project.

Tong et. al. [1] mainly focus on studying potential errors in GPS data, but also on analysing
congestion on one specific stretch of highway, through the use of GPS data. In their analysis
they only focus on peak-hour traffic, and make congestion analyses based on average
speeds. The data collected is from a single vehicle, following a specified route, a few
weekdays during peak hours.

Mathew et. al. [3] focuses on identifying single trips (trip start and ends), and subsequently
analysing these trips. A very few number of vehicles are used.

Wenhuan Shi et. al. [2] describes a system with GPS fitted vehicles, to create a spatial map
(GIS) of a the downtown Shanghai area, followed by a proposal to use the GPS data to
make traffic flow models. The main difference from other work we have seen, is that they a
very large pool of vehicles included, up to 8000 taxis fitted with GPS devices, which they use
to create a real time state of the traffic flow and congestion, intended for use in traffic
coordination and navigation.
In this paper we focus on analysis of light-controlled intersections, based on data from a pool of more than a hundred vehicles, from a period of 3 months, which is in excess of 20 million data points. We will present a solution to filter through the large amount of data, followed by an analysis of a light-controlled intersection. The analysis will calculate relevant information about congestion from the data, such as average speeds and times of vehicles going through the intersection.

We focus directly on light-controlled intersections, which none of these other projects has done. They are often the source of congestion, because cars have to slow down or stop.

4. Development

In this part of the paper we describe initial methods for creating a useful output, from raw data. The GPS data used, originates from a the project, Spar På Farten [7], designed to monitor a vehicle’s speed. Furthermore the irrelevant data for this analysis has been filtered out for performance reasons.

Database Setup

The table diagram (Figure 2) depicted above, represents how our database is setup. The tables are as follows:

- **gps_data** - Contains a subset of the available GPS data, which has been filtered out for performance reasons. It consists of a point recorded every second for each vehicle, whenever the engine of the vehicle is turned on. Each data point’s primary key is a composite of vehicle_id and date_time. The data associated with each data point are latitude and longitude (in UTM format [11]), speed_gps, which is the speed of the vehicle, road_segment, which identifies a map-matched road segment in roads. Finally, we have spatial_point, which is a point calculated from the GPS coordinates, and the only column which we have added to the data our self. The raw data does contain more information, but it is not used for this project.
- **trips** - This table contains all the trips, the vehicles have driven. The start of a trip is defined by the engine being turned on, and it ends when the engine is turned off, this is derived from the `gps_data` table, by searching for time gaps in the data points. Data points from a trip, can then be found as a range from one vehicle (`vehicle_id`), by selecting all data points between `start_time` and `end_time` (`date_time` in `gps_data`).

- **roads** - This is spatial information, obtained from a shapefile [15] provided with the data for this project, which all the `gps_data` has been map matched too. Each road segment is numbered with `road_segment`, which corresponds to numbers in `road_segment` from `gps_data`.

- **spatial_lines** - Each trip stored in `trips`, is also stored as a spatial line in this table. Each line consists of the appropriate `spatial_points` in `gps_data`, identified by the times recorded in `trips`. Trips and `spatial_lines` are separate, because the `spatial_lines` column named `spatial_line` (each line is made up of hundreds of points) has allocated big amount of memory on the physical hard disk. Therefore to avoid performance issues, it is split into two tables and creates the 1 to 1 relation.

- **intersection_data** - information about the intersection, which is analyzed. It contains the columns `id_intersection`, `spatial point` - which represents the intersection center point in geometry data type and the `name` of intersection. It has no direct relations to the rest of the database, as it mainly used to store particular intersection data, this table is the key to analysing more than one intersection at a time.

**Data processing**

Our intersection analysis started by filtering useless data, as not all the data is useful for our analysis. First of all, there are GPS points which have not been map matched to any segments of the map, and which should be filtered out, to avoid corrupting the results of the analysis.

Remaining filtered points are allocated to separate trips, by incrementally checking the time stamps for each of the vehicle. Furthermore the GPS units, which generated the data, would only generate data whenever the vehicle’s engine was turned on. Therefore if there was a time gap for more than 30 seconds, we considered it a new trip. The 30 seconds time period is chosen, because it is big enough for driver to restart his vehicle engine if it e.g. stalls, and small enough to separate trip if driver went to the gas station or shopping mall.

Vehicle traveling routes - trips, are determined by checking if there are points in the trip which were map-matched to two different map segments of the intersection. This is a safeguard to separate a trip by a vehicle, which traveled through the intersection.
Filtered data is used by spatial queries to calculate the results. One of the results is traffic flow [16] in the different directions of the intersection. This is gathered by calculating trip count through the intersection for each direction. Travel time is calculated by getting time when a vehicle entered and exited the intersection boundaries. Intersection boundaries are the radius of a circle around the center of intersection.

Red lines in the Figure 4 represent the intersection boundaries and red dots represent GPS data points which are in the intersection boundaries for one trip. The speed of a vehicle, crossing the intersection as shown Figure 4, is estimated by querying the speed for each point within the radius of a circle, and summing them all together and dividing by number of points, which are within the intersection boundaries.
A vehicle which was within intersection boundaries and its speed equals two kilometers per hour or less is treated as waiting in queue in the intersection. Travel time in the intersection was calculated by counting all points for each trip, which were within the intersection boundaries and the vehicle speed is higher than two. The 2 km/h speed check was chosen, because when you are driving at 2 km/h, we could still classify it as being in a queue, and also there could be an inaccuracy in the speed measurement.

Data gathered during data processing will be processed and presented in more convenient ways such as tables, charts, and other types of graphical representations.

**Keyhole Markup Language graphical representation**

The KML (Keyhole Markup Language) [5] tool was developed in order to represent trips on a map. This tool takes the data from the database, converts it to a KML file which can be used in e.g. Google Earth [12]. The color of the line indicates the time when the trip took place. For example, our chosen intersection with three months’ worth of traffic:

![Figure 5. Representation of GPS trips in Google Maps [10] KML format](image)

The purpose of the KML output and representation is mainly for discovering anomalous data and to ease error checking. As visible in Figure 5, our GPS data does contain some trips, which is too far away from the road and is not map matched properly (good example is a white line on the left bottom corner). There may be several reasons for this, for one the western road is surrounded by tall trees, which might interfere with the GPS signal. Furthermore the initial purpose of the project Spar På Farten was not to provide GPS data for analyzing intersections; therefore some inaccuracies are not avoidable.
By creating a graphical representation, it is possible to visually check the data, which can help to confirm or reject the results of an analysis. Figure 5 is obviously extremely cluttered, but it is also possible to extract single trips.

5. Implementation

In this section we will look through the tools and techniques we used to achieve the results. It starts with description of the software we use, then we have an explanation of data and its characteristics, and finally a description of our algorithm with its pseudo code.

Software

For this project we have used PostgreSQL 9.0, combined with PostGIS 1.5, as our database system, and pgAdmin III to manage that. For the implementation Java 6 was utilized, and Postgres’ own JDBC (Java Database Connectivity) driver was used to connect to the database. As mentioned earlier, Google Earth was also used to visually represent data on a map.

These are all free tools, available online to everyone. They have suited this project excellently, however one should be aware of a few limitations in pgAdmin III. It’s SQL Query editor and executor, has a limit on the number of characters it is able to display, and it does not display an error when this limit is exceeded, but just displays blank space. That meant that the spatial lines of trips could in some cases not be displayed as strings in the SQL editor. Also the SQL execution in this tool is substantially slower, than the JDBC driver, therefore pgAdmin III should not be used to evaluate speed and efficiency of queries or the database.

Data

The amount of raw data we received from Spar På Farten project is huge. There are 23 millions of GPS data rows, 105 unique vehicles driving and over 6,300 hours of driving time in total.

In the beginning of our research we thought that the best place to install a database was to a dedicated server. However this was not an option, even though the data processing power of a server might have been greater, we found that downloading the results from the server was taking too long, because of bandwidth limitations. Taking in consideration that 23 million points of data allocate more than 2 Gbytes of memory and with average connection of 5 Mbits/s to the database server, it will take approximately 50 minutes just to retrieve all GPS data points from database. This problem was solved by installing the database with the same data into all our personal computers, which we were using for this project. A database installed locally gives us read speed from the hard disk in the range of 50 Mbytes/s (400 Mbits/s) (tested with Disk Benchmark 2.6) [14].

The intersection analysis is made for the particular intersection in Figure 1, and to further ease the computer intensive calculations and to handle the big amount of data, the GPS
data (which belongs to the trips that crossed the intersection) were copied to a separate table. It reduces the amount of data that has to be processed each time the analysis is executed and it increased the performance. We use this trick, because none of us have computer that is powerful enough to run through the whole data.

Working with real world data has its own peculiarities. There is a thing called traffic uncertainty. It means that trips might not just go from point A to point B, but also include some problems in the intersections, like stopping near it or going through the intersection and returning back at a later time. You have to take everything into consideration, because our experience shows that if you forget even a small thing you can get a forty minute trip in the intersection on a left turn and that would corrupt the results enormously.

To counter these problems we use different techniques. If a vehicle has an emergency stop near the intersection (within radius we are calculating results), if it starts to move again after 30 seconds, we mark it as a new trip. Furthermore, we find it difficult to analyse trips of the intersection that goes through it more than once. After a discussion on how we should handle it, our choice is to check for a time gap in a trip during the analysis. In case this kind of occurrence is found it will split this trip. It is done after the data filtering; however GPS points are not stored in a database as a new set of trips.

Algorithm

The following section will show some parts of the algorithm used to calculate the different results.

The algorithm can calculate three different statistics. The first is the average time it takes to go through an intersection, the second is the average speed and the last is the average time spent queuing in the intersection. The algorithm has two parameters, a radius limit and an optional direction limit.

The average time is calculated by taking all the GPS points for a trip that is within the radius limit and adding together the time differences between the points. That number is then added to the numbers for each of the other trips. When all trips have been calculated the total time is divided by the number of trips. The average time is calculated for each direction separately. Figure 6 shows pseudo code for the average time calculation.
Figure 6. Pseudo code for calculating average time it takes to go through intersection.

```c
    // Used to calculate the average time it takes to go through the intersection.
    CalcAverageTime(trips)
    {
        time, count = 0
        foreach (trip in trips)
        {
            val = 0
            foreach (GPSPoint in trip)
            {
                val += 'time diff. between this point and the previous'
            }
            time += val
            count++
        }
        return (time / count)
    }
```

In figure 6 the foreach loop from line 5 - 14 loops through all the trips. Within that loop is another foreach statement (lines 8 - 11) that loops through all the GPS points for the trip and adds together the time difference between the points. Finally it returns the result of time divided by count.

The average speed is calculated by taking all the GPS points for a trip, which is within the radius limit. The speed for each point is added together and after is has finished adding, the total is divided by the number of points. The average speed is calculated for each direction separately. Figure 7 shows pseudo code for calculating average speed.

```c
    // Used to calculate the average speed for a single trip.
    CalcAverageSpeed(trip)
    {
        speed, count = 0
        foreach (GPSPoint in trip)
        {
            speed += 'Speed from GPS Point'
            count++
        }
        return (speed / count)
    }
```

Figure 7. Pseudo code for calculating average speed

In Figure 8 the foreach statement (lines 5 - 9) loops through all GPS points of a single trip, and adds together the speed. The function returns the result of dividing speed with count.

The average queue time is also calculated by taking all the GPS points for a trip that is within the radius limit. The speed of each point is checked, and if it is less than or equal to 2 km/h, the time spent in queue for this trip is incremented. After checking all the points for all the trips, the average is found by dividing the total time by the number of trips. As with the two other parts of the algorithm, the average queue time is also calculated for each direction separately. Figure 8 shows pseudo code for calculating queue time.
Figure 8. Pseudo code for calculating average queue time

In figure 8 the first foreach statement (lines 5 - 17) loops through all the trips. The second foreach statement (lines 8 - 14) loops through all the GPS points of the trip, and if the speed is less or equal to 2 km/h val is incremented. After the second loop val is added to time. The function returns the result of dividing time with count.

In practice the three examples could be put into the same function. However to help make it more clear and readable the calculations have been put in separate function.

Our solution has been implemented using approx. 950 lines of code. The analysis, at its current configuration where results are calculated both for a radius of 50 meters and a radius of 100 meters, takes around 40 seconds to execute on a modern laptop.

6. Results

Results of this analysis represent traffic flow and congestion in different directions of the intersection. There are a total of twelve different directions in this 4-leg intersection. There are 3 different directions from each side of intersection - left turn, right turn and forward. We do not analyse “U-turn” travel types, because the percentage of this type of turns is very low and our method used for all others crossing types does not produce correct results for this type of turn, and therefore it can corrupt all the other results.

We used cardinal directions to represent each direction, e.g. North, South, etc.. Each leg of the intersection has been assigned to the closest cardinal direction as can be seen in Figure 9. For example, the direction from Vesterbro to Hobrovej been assigned to a direction called North-to-South (NS).
We analysed the intersection using 1036 different trips which goes through the intersection. It is illustrated in Figure 10. The main part of traffic flow - more than 40% (424 trips) in this intersection, goes from North-to-South (from/to city centre and the shopping mall in southern Aalborg) or South-to-North. There are 27% (285 trips) of traffic flow from East-to-West and West-to-East. In conclusion, there is 48.8% more traffic going from North-to-South and backwards then East-to-West and backwards.
Average speed going through the intersection, compared with average time in queue within 50 meters radius around the intersection, is represented in Figure 11. The analysis shows that the longest average time to turn left is when you are going from West-to-North (from Kong Christians Allé towards the city centre). This is because this is a left turn and it crosses the direction with the highest traffic flow (North-to-South) so it takes time to make the turn. Looking at these results we see that they live up to our expectations, which e.g. left turns take longer than right turns.

![Average times within 50 meter radius comparison graph](image)

Figure 11. Average times within 50 meter radius comparison graph

Comparing the traffic flow in different directions, with average time spent in a queue in Figure 11, we discovered the following. The traffic which are coming from West and North takes a longer time to go through the intersection, than the traffic coming from East and South (21 & 20 seconds, compared to 9 & 14 seconds).

Waiting time for going North-to-South (the highest traffic flow direction) is not the highest, and it is even lower than other straight crossing, West-to-East, that means that the traffic lights was configured with consideration to the high density of traffic flow.
Figure 12. Average time within 100 metres radius comparison graph

Figure 12 indicates the increased results mention in Figure 11, but with a larger radius around the intersection. This gives us better understanding of queue length in the intersection. Comparing with Figure 11, the average time spent in queue has increased by approximate 20-25% for all directions.

Figure 13. Average speed comparison graph
Figure 13 shows that speed depends on the kind of turn vehicle have to take. In addition, the average speed is higher with a bigger radius, with one exception, going North-to-South. We think the cause of it is a small 3-way intersection just north of the one we are analysing.

7. Conclusion
All things considered, working with huge amounts of data, changes the way you think about database management, working with, and querying data. Query optimization and data separation to different sets is crucial, if you want to get the results fast enough. Speaking about the former, a good advice is to try queries on smaller sets of data. Furthermore, while working with different sets of familiar data, redundancy is also a thing you have to be aware of.

Results of the analysis revealed traffic tendencies in this particular intersection. Such as more drivers come from/to the city centre and to/from the shopping centre at City Syd than from/to the Eastern part of the city and to/from the Western part. It is very small number of drivers who goes from Western part of the city to the city centre using this intersection. In general after doing the analysis we can, by looking at the time needed to drive through the intersection, conclude that it is properly configured, when considering the traffic flow in it.

The algorithm itself needs some work before it can be generalized and the software we created is not yet ready to be used by people without knowledge in computer science. On the other hand, it is enough for us to make an analysis and further development can be done if needed.

8. Acknowledgements
We would like to thank the people that worked on the “Spar På Farten”-project and Aalborg University, who shared the GPS data with us.
9. References


All the online reference links was last checked 2011-05-18