Marek Bazan
Wrocław University of Science and Technology, Faculty of Electronics, Department of Computer Engineering, ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland
marek.bazan@pwr.edu.pl

Janiczek Tomasz
Wrocław University of Science and Technology, Faculty of Electronics, Department of Control Systems and Mechatronics ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland
tomasz.janiczek@pwr.edu.pl

s. 7-21

Kurda Rafał
Wrocław University of Science and Technology, Faculty of Electronics, Scientific Circle CyberTech, ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland

Matusiak Kamil
Wrocław University of Science and Technology, Faculty of Electronics, Scientific Circle CyberTech, ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland

Sak Łukasz
Wrocław University of Science and Technology, Faculty of Electronics, Scientific Circle CyberTech, ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland

AN ALGORITHM FOR VEHICLE IDENTIFICATION BY ON-BOARD BLUETOOTH DEVICES EXPLOITING BIG-DATA TOOLS

ABSTRACT

Nowadays, vehicles are equipped with various on-board devices that work in Bluetooth technology and log on to the ITS infrastructure whenever passing by Bluetooth readers. The location of Bluetooth readers is an important issue for travel time prediction in urban areas. Bluetooth technology is used to enhance travel time prediction accuracy and is additional to vehicle license number identification. The algorithms for travel time prediction are used by such technologies e.g., TRAX to offer the road user an alternative route to traverse the most congested regions of the city in the most efficient way. In this paper we present the implementation of the algorithm that enables us to match Bluetooth on-board devices, and also cell phones that are mounted or are just in vehicles of road users. Since

the ITS is a source of an enormous and increasing amount of data for this purpose we engage Big Data tools such as Apache HaDoop and Apache Spark. To build Map-Reduce tasks we use Hive-SQL. The algorithm is tested on ITS data from the city of Wroclaw. The results of the algorithm may be used to locate stolen vehicles.

KEYWORDS

Automatic number plate recognition, Bluetooth devices, HaDoop, Spark, user identification

INTRODUCTION

Automatic number plate recognition (ANPR) systems are the main component of the travel time estimation systems within intelligent transportation systems. The most effective methods of recognizing license plate numbers consist of the following three stages:
1) license plate location within the image,
2) license plate image normalization,
3) optical character recognition.
The broad survey of the mathematical algorithms exploited in the above stages can be found in [1] and references therein.
The travel time estimation methods that are employed in intelligent transportation systems are based on the appropriate placements of the ANPR cameras in the city along its main arteries c.f. [2] . That allows us to propose the alternative routes for the quickest traversing of the city c.f. [3] – for the alternative routes travel time estimation quality.
Apart from travel time estimation, ANPR cameras in the ITS system can also be used for congested traffic prediction as presented in [4] .
Bluetooth is one of the most popular technologies that nowadays allows us to establish a vehicle to infrastructure ad-hoc networks (VANETs) [5] that enhance travel time estimation and prediction capabilities of intelligent transportation systems c.f. [10] . In depth analysis of the error that can be encountered in travel time estimation may be found in [7],[9] and references therein.
VANETs connected to the GPU servers within the infrastructure servers may also be used for a fastest path search, based on congestion measurement by an ad-hoc network Bluetooth gate software c.f. [6] .
Optimal locations of Bluetooth readers in an urban environment so that travel time prediction is as accurate as possible were recently studied in [8] .
In this paper we present an off-line data-mining algorithm of license plate numbers matching with the MAC addresses of on-board Bluetooth devices exploiting Big Data tools. To our knowledge this is the first publication of such an algorithm, not only in Big Data, but in the general data-mining field. With the exception of the presentation of the algorithm the purpose of this paper is also to show a track of the development of its implementation in Big Data ecosystems. Big Data analysis is nowadays increasing in popularity since its tools allow the processing of an enormous amount of data both in real time [11], [12] as well as off-line analytics [13], [14]. Since transportation systems are sources of a quite large amount of data, it is normal that they are a potential field of Big Data analysis tools applications – for an extensive review of what has already been researched in transportation related to Big Data, see [15].
Summarizing a contribution of this paper to the field of transportation telematics: in the context of the literature of the subject of vehicle identification up to our knowledge it is a first paper that considers deanonimization of Bluetooth devices and matching them with license plate numbers in the ITS systems containing Bluetooth gates and ANPR systems.

The remainder of the paper is of the following structure. In the second section we present a general idea of the matching algorithm together with the data formats that are needed. In the third section four implementations of the algorithm are presented. Codes of the first three implementations are in python and in Transact SQL whereas the third one is in Apache Spark. The first two implementations are characterized by different execution performance and are tools for checking if there is a necessary dependence in the data sets that we analyzed for tests. In this section the implementation process of the corresponding Map Reduce algorithm is also discussed. In the fourth section we present results of the execution of the above implementations on ITS data from the city of Wroclaw (Poland). In this section we show the placements of ANPR cameras as well as Bluetooth readers in the city, and establish the conditions necessary for the algorithm to work. As results we sketch the number of matches as well as execution times. Finally we conclude the paper depicting possible areas of the application of the algorithm.

1. GENERAL IDEA OF A MATCHING ALGORITHM

An algorithm relies on the fact that there are two separate tables in the ITS system that store result data from ANPR cameras and Bluetooth gate readers located in various places of the city. The location of ANPR cameras and Bluetooth readers are designed so that travel time estimation for VMS is possible [8]. The crucial requirement is that there are points in space where ANPR cameras are close to Bluetooth gates and are placed on main arteries, so that the vehicles driving the same way are caught on both of them. Location of the cameras and Bluetooth gates in the city of Wrocław can be seen in Fig. 1.

2. IMPLEMENTATIONS

2.1 DATA PREPARATION

Data preparation is a very important stage in the processing of big data. It allows us to save time performing algorithms especially for sequence implementations. In this section we present a preprocessing approach in the form of the discussion on the python implementation.

Steps of the algorithm:
I. Read data
II. Sort data:
- first by Bluetooth MAC address
- next for every MAC address by registration scanning time

III. Main function:
1. Initialize variable
2. Main for loop iterate all elements in data plus one for the last iteration. Next, create a condition to the last iteration (line 52-55).

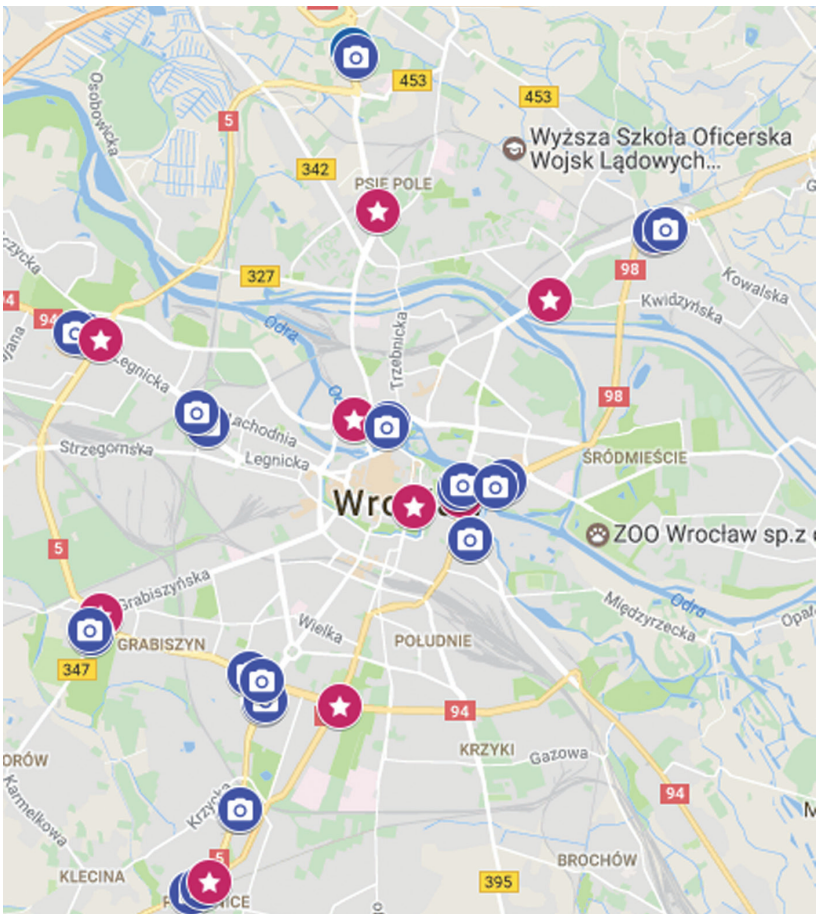

Fig. 1. Locations of cameras and Bluetooth scanners in the city of Wrocław. Purple Star – position of Bluetooth scanner (11), Blue Camera – point camera (26). In Wroclaw there are more points with cameras, but only those are chosen which are near Bluetooth scanner locations.

2.1. MAIN IF. (LINES 58-75)

This condition is fulfilled when the MAC address and Bluetooth ID was the same and it is not the last iteration. It is in a situation if between the next two rows in the base, the scanning time is longer than 15 minutes and the current cumulative time is longer than 30 minutes (this was probably a static transmitter close to a Bluetooth scanner or a very long traffic jam) then save indexes to delete data.

2.2. MAIN ELSE: (LINES 77-89)

If MAC address or Bluetooth ID are changed (new MAC address to check or if a car was at another Bluetooth scanner), but the previous cumulative time was longer than 30 minutes, then save indexes to delete data.
IV) Print how many elements will deleted from data (line 91-92)
V) Return list with indexes to deleted data. (line 94)
VI) Removing unnecessary data (indexes to delete are in return list form Main Function)
VII) Save data after preparation

```
    #for every element in list
    for i in range(1,len(data)+1):
        #if for last iteration
        if i == len(data):
            BT_mac = 'if last iteration :) Go to else'
            #set too big index to last
            i = -1

        #if we have the same BT_ID and BT_mac and it is not last element
        if data[i][0] == BT_id and data[i][3] == BT_mac and i != -1:
            #compute cumulative time
            cumulative_time = datetime.strptime(data[i][1],TIME_FORMAT) - BT_time
            #set variable last element to delete on i
            last_element = i
            #time between two row
            time_change = datetime.strptime(data[i][1],TIME_FORMAT) - datetime.strptime(data[i-1][1],TIME_FORMAT)
            if time_change > time2:
                if cumulative_time > time1:
                    #count how many elements we need delete from this condision
                    c1 = c1 + last_element - first_element
                    #add to list first and last element to delete from data
                    delete_indexes.append([first_element, last_element])
                #reset cumulative time
                cumulative_time = timedelta(minutes=1)
                #set new value to first element and new start BT_time
                first_element = i
                BT_time = datetime.strptime(data[i][1], TIME_FORMAT)
        #if BT_id or BT_mac are change or it is last iteration
        else:
            if cumulative_time > time1:
                #count how many element we need to delete
                c2 = c2 + last_element - first_element
                # add to list first and last element to delete from data
                delete_indexes.append([first_element, last_element])
            # reset cumulative time
            cumulative_time = timedelta(minutes=1)
            #set new value
            BT_id = data[i][0]
            BT_mac = data[i][3]
            BT_time = datetime.strptime(data[i][1], TIME_FORMAT)
            first_element = i
    #print how many elements we will delete
    print(c1)
    print(c2)
    #return list with indexes to delete from data
    return delete_indexes
```

Code snippet 1.: Data preprocessing algorithm in python. A procedure to delete data related with a local Bluetooth transmitter or if parked vehicles transmit a Bluetooth signal more than 30 minutes near one Bluetooth scanner (a very long traffic jam or vehicle parked near a Bluetooth point).

Results:
All elements in base: 10 653 923
Delete elements:
c1 = 4 092 735
c2 = 925 91
Saved elements: 5 635 271
The algorithm deleted more than 45% of the data from the file.
Execute algorithm time: 26 minutes 44 secs (algorithm working on one thread)
Processor: Intel(R) Core(TM) i7-4702MQ CPU @ 2.20GHz (8 CPUs),
Memory: 8192MB RAM.

2.3. SQL

SQL implementation presented in this section was designed for a Microsoft SQL Server environment using Transact-SQL. The main part of the algorithm is presented in *Code snippet 1.*

```
1  SELECT z1.Car_Registration, z1.Bluetooth_Address, COUNT(*)
2  FROM (
3     SELECT t1.Car_Registration, t3.Bluetooth_Address, t1.Time_Reg
4     FROM dbo.Registrations t1
5     JOIN dbo.Reg_Bl t2
6        ON t1.Camera_Id = t2.Camera_Id
7     JOIN dbo.Bluetooths t3
8        ON (t2.Detector_Id = t3.Detector_Id)
9        AND (t1.Time_Reg BETWEEN DATEADD(minute, -5, t3.Time_Bl)
10       AND DATEADD(minute, 5, t3.Time_Bl))
11    GROUP BY t1.Car_Registration, t3.Bluetooth_Address, t1.Time_Reg
12 ) z1
13 JOIN (
14    SELECT t1.Car_Registration, t3.Bluetooth_Address, t1.Time_Reg
15    FROM dbo.Registrations t1
16    JOIN dbo.Reg_Bl t2
17       ON t1.Camera_Id = t2.Camera_Id
18    JOIN dbo.Bluetooths t3
19       ON (t2.Detector_Id = t3.Detector_Id)
20       AND (t1.Time_Reg BETWEEN DATEADD(minute, -5, t3.Time_Bl)
21       AND DATEADD(minute, 5, t3.Time_Bl))
22    GROUP BY t1.Car_Registration, t3.Bluetooth_Address, t1.Time_Reg
23 ) z2
24    ON (z1.Car_Registration = z2.Car_Registration)
25    AND (z1.Bluetooth_Address = z2.Bluetooth_Address)
26    AND (z1.Time_Reg NOT BETWEEN DATEADD(minute, -360, z2.Time_Reg)
27    AND DATEADD(minute, 360, z2.Time_Reg))
28 GROUP BY z1.Car_Registration, z1.Bluetooth_Address
29 HAVING COUNT(*) > 2;
```

Code snippet 2: Core fragment of SQL implementation. [own study]

Implementation is based on three tables:

1. Registrations – contains data from cameras capturing car registrations.
2. Bluetooths – contains data from Bluetooth device detectors.
3. Reg_Bl – reference table, contains camera identifiers? matched with Bluetooth detectors located in the same area. Created specifically to solve the problem described in this paper.

Relations amongst tables mentioned above are shown in Fig. 2.
Zero or many relations between the tables arise from a few facts:

- not all cameras, same as not all Bluetooth devices detectors, take part in the calculations, because not all of them have their pair – camera and Bluetooth devices detector become a pair only if they are in close proximity to one another;
- it is not always the case that Registrations or Bluetooth tables must have records for all cameras or Bluetooth devices detectors specified in Reg_Bl table;

The first step of the algorithm, represented in *Code snippet 1* inserted above by subqueries in FROM clause (lines 3-11), is to get all distinct sets of vehicle license plate, Bluetooth adapter MAC address and the time of license plate number capture. Each set must meet the following conditions:
The camera that captures a license plate number and detector that captures Bluetooth devices must match each other (Camera_Id and Detector_Id creates a pair, which is saved in Reg_Bl table).
The time when the license plate number was captured (Time_Reg) and the time when the Bluetooth device was recorded (Time_Bl) must meet the specific condition presented in *Code snippet 1* in lines 9-10 and 20-21. In the described implementation the

condition is that the difference between the Time_Reg and Time_Bl cannot be greater than 5 minutes. This condition should be adjusted to a given location taking into account variables like traffic or average distance between the cameras and Bluetooth detectors. The last part of this step is the GROUP BY clause used to deduplication of the results, in case the same Bluetooth device address was captured many times in a given period .

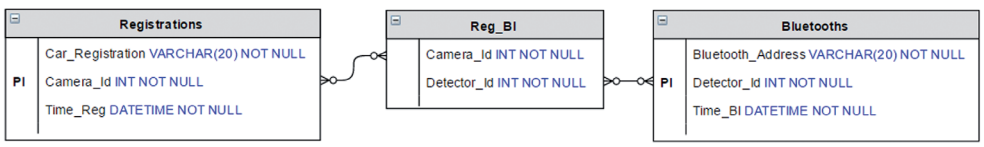

Fig. 2. A fragment of database infrastructure, identifiers important for the implementation of the algorithm. The descriptio covers only attributes required to demonstrate the implementa-tion. [own study].

The second step of the algorithm is to exclude the results received only during one way through many measurement points. This approach is necessary in case when the same group of vehicles drives through many measurement points in similar time. Cases like this can cause the same set of results for more than one vehicle, which is an undesirable situation. To achieve this, results from step 1 are joined with eachother, which is represented in *Code snippet 1* the above inserted by the JOIN clause (lines 13-27), and get into further calculations only these results that meet the following condition:

1. vehicle license plate number and Bluetooth device MAC address is the same,
2. time between captures of subsequent results is greater than 360 minutes.

The third and final step of the algorithm is to group the results received for each pair of vehicle license plate number and Bluetooth device MAC address together. This part is represented in the inserted above *Code snippet 1* by lines 1, 28, and 29. As the final result of the algorithm we get a car registration with a corresponding Bluetooth device MAC address and with a count of occurrences. An important issue in this step is the clause HAVING, located in line 29. The number at the end of the line is responsible for the number of occurrences that specific set of results must have to be in the final results. The higher the number the more the results are dependable, but there is one thing that must be borne in mind - results with a lower number of occurrences than this will never get to the final results, e.g., when the HAVING COUNT is set to greater than 20, there will not be in the final results cases where a vehicle license plate number occurred 19 times and always for these 19 times a specific Bluetooth device for this car registration was found, but there will be in the final results a case where a vehicle license plate number occurred 1000 times and only 20 times for all these occurrences a specific Bluetooth was found.

Possible ways of optimization of the presented algorithm are:

1. Reducing the amount of time and memory required for processing by reducing the source volume of data in the Registrations and Bluetooths tables to only those that Camera_Id / Detector_Id which match any Camera_Id / Detector_Id specified in Reg_Bl table.
2. Reducing the amount of memory but increase the amount of time required for processing by utilization of cursors to store all distinct vehicle license plate numbers and process them row by row.

2.4. PYTHON IMPLEMENTATION

Python implementation presented in this section was written in Python 2.7.
Before using the algorithm we need to sort registration numbers and Bluetooth data by time.

First step in the algorithm is to prepare data:

1. Choose start and end time (year-month-day hour:minutes:seconds).
2. Sort data:
   - License plate number data by a registration number
   - Bluetooth data by Global_ID (location of the Bluetooth scanner)
3. Load into list():
   - Registration file - Registration numbers, Global Camera ID, acquisition time
   - Bluetooth file – MAC Bluetooth address, Global Bluetooth ID, acquisition time
4. Delete duplicated registrations using set (registration)
5. Create a dictionary where the key is Global Camera ID and value is list of a near Bluetooth scanner (maximum 2 km).

The main part of the algorithm, matching a vehicle license plate number with a Bluetooth device is presented in *Code snippet 3*.

```
for rej in rejestracje_niepowtarz:
    #count how many registration is in list
    counter = rejestracje_powtarz.count(rej)
    #the first occurring index
    indeks = rejestracje_powtarz.index(rej)
    device_list = list()
    #if only one instance registration in list then omin it
    if counter <= 1:
        continue

    previous_time = datetime.strptime(registration_time[indeks], TIME_FORMAT)
    #for index one registration
    for i in range(indeks, counter+indeks):
        current_time = datetime.strptime(registration_time[i], TIME_FORMAT)
        if i != indeks and current_time < time_stamp(previous_time, 6):
            continue
        previous_time = current_time
        #window time +- 5 minutes
        mini, maxi = min_max_time(current_time, 10)
        list_for_one_registration = set()
        #for error data in file
        try:
            for cam_blue in cam_to_blue[rejestracje_ID[i]]:
                indeksB = bluetooth_MAC_adress.index(cam_blue)
                counterB = bluetooth_MAC_adress.count(cam_blue)
                #binary search for time interval
                start = bisect_right(bluetooth_czasy[indeksB:indeksB+counterB],mini)
                end = bisect_left(bluetooth_czasy[indeksB:indeksB+counterB],maxi)
                start = start + indeksB
                end = end + indeksB
                list_for_one_registration.update(set(bluetooth_id[start:end]))

            device_list.append(list_for_one_registration)
        except KeyError:
            pass
        except ValueError:
            pass
    #for intersection
    intersection_set = set()
    #if only one set
    if len(device_list) > 1:
        intersection_set = device_list[0]
    else:
        continue
    for i in range(1, len(device_list)):
        intersection_set = intersection_set.intersection(device_list[i])
```

Code snippet 3: Python implementation of the matching algorithm,

1. Iterate over license plate numbers. (line129)
2. Count number of occurrences of every registration number. (line131)
3. If registration number occurred only once, omit it. (line136-137)
4. If the detected time is greater than the exact value given as a parameter then omit it. This solution is for a situation whereby two or more cars are going in the same direction at the same time. (line139-145)
5. Calculate the window time +/- a few minutes (Camera and scanner Bluetooth are in different places) (line147)
6. Load Bluetooth data and match it with the registration number:
   - Start Loop: For each specified camera get all Bluetooth points instances:
   - + Get a first index MAC address which is appropriate to a Global Bluetooth ID (line152)
   - + Count how many MAC addresses are in the appropriate Global Bluetooth ID (line153)
   - + Execute binary search algorithm (data was sorted by time) for the time window (lines155-158)
   - + Save data to "set" structure (line159)
   - End loop
   - Append list of devices from one checkpoint to the list of "set" structures (line161)
   - Calculating common part of all checkpoints using intersection on "set" structure (lines 173-174)

Advantages:
- The algorithm is adapted to use multiprocessing/multithreading - it allows usage of parallel computing which speeds up execution time
- Calculating the common part of all records is expected to provide high accuracy

Disadvantages:
- If Bluetooth is turned off once or more while a car is passing the Bluetooth checkpoint, or if a car doesn't pass by the nearest Bluetooth scanner, then this algorithm won't work.

Python vs. SQL algorithms

Result:
Compare results based on data from 14-15.05.2015.
Data source: ITS Wrocław

Before comparing results the following command was executed:
uniq -u - only print unique lines,

If one Bluetooth device was matched to many different registration numbers, then all of the results with this Bluetooth device are removed – the algorithm cannot decide which result is the correct one. This situation may occur if some Bluetooth device is constantly detected at one of the Bluetooth stations.

SQL:
- Execute time: 38h 31m 25s
- Number of registration: 1385
- Number of MAC address Bluetooth: 1784

Python:
- Execute time: 3h 40m 31s
- Number of registration: 2062
- Number of MAC address Bluetooth: 2819

2.5. APACHE SPARK IMPLEMENTATION

There are a few dozen cameras and Bluetooth receivers stationed by the roadside in Wroclaw.
These cameras are capable of detecting a car and reading its registration plate number. Receivers try to identify numbers of devices passing under.
Data is collected and stored by a centralized system administered by the municipality of Wroclaw. The authors of this publication received a slice of data that was gathered within several days in 2014. Provided records were saved in CSV files that contained a few million rows. Camera data contained only fully registered journeys from point A to B. Therefore it contained two places, two timestamps and id number, and additional information like input time. Bluetooth data contains timestamp, id, type, and input time. The task was to find cars and Bluetooth devices that occur. I have implemented the following algorithm:
- Select cameras that are near Bluetooth receivers
- For every capture of car select all Bluetooth devices that were present at the nearby location and time.
- For every vehicle create a table that contains Bluetooth devices and number of times it was present near a vehicle.
- Similarly, create a table for every Bluetooth device that contains car numbers that occurred with it, and store the number of times it happened.
- For every vehicle license plate number select from its table the most often appearing Bluetooth. Check if the current car number is also the most often appearing in the table created for this Bluetooth address.

Such an algorithm is resistant to all imperfections of real world data and fulfils its task specification:
Sometimes a vehicle plate number or wireless device may not be detected when the other is. Especially if the vehicle is traveling very fast, license plates are dirty, or the phone is turned off. Periodically the vehicle owner may switch to a different vehicle or walk. When a person uses public transport, we cannot claim that there is a connection.
We have implemented the above algorithm in a python and Hadoop Map-Reduce framework. However, for both implementations we had to slightly modify the above method of proceeding. First of all it does not fit the Map-Reduce framework.

Modified algorithm
Map phase:
For every record create key, value pair. The Key should be a union of timestamp, and generalized place of occurrence. The Value should be identification number of the car or device.
At this stage the time of occurrence should be modified. The precise time of occurrence is of little interest. Therefore, time in seconds should be divided without leaving a remainder by a small period of time. I use 60 seconds. This is enough to check if co occurrence happens.

Reduce Phase:
The result of the Map phase is sorted to efficiently find a list of all occurrences at a given place and time.
The reversed result from the map should be sorted to find all occurrences of a given id. Then for every occurrence of an id find all ids that occurred at the same time and place. Add all resulting lists for an id and the calculate frequency statistic. Save as a result several highest results. It is impossible to save a full list as this would require too much memory.

Parallel python implementation:
Map phase:
Files are divided into sublists containing one million records from the CSV file. Each is put onto a job queue. Then a few threads (the number depends on the machine) process these sublists. The result is a **numpy**??? array of 3 element **tuple**?? (place, time period, id number).
Sort Phase:
The resulting lists are joined.
From this we create two collections.
List sorted by id number -- for an efficient retrieval of all occurrences of id.
Dictionary where key is tuple(time, place) and key all id numbers that occurred there.
Reduce phase:
Loop through the first list and create a sublist with all occurrences (time, place) of an ID number. For every element on this list retrieve all co-occurring ids in the created dictionary. Join all the results and create ranging of frequency. Save several of the biggest results. Look for all id numbers that co occur with other ids and the reverse relation is similar.
Results:
This implementation performed exceptionally well. Python is considered by many to be a slow language. But the right algorithm with Numpy performed very fast. 15 million records and 1.1 Gb of data was processed on a laptop in 10 minutes! CPU utilization was around 70% and memory usage topped at 6 Gb.
Hadoop Implementation
Implementation was similar to the python one. The only difference is that parts of the sort phase that are not automatic are moved to the reduce phase (see Code Snippets 4, 5, 6 for detailed implementation).

```
------------------------------------------------------------
First Stage
This should be done for bluetooth data, and car travel data.
At this point we need to convert CVS data into more efficient structure.
Moreover, time is changed to window number -- events that occure in the same time window may be correlated
It is required to reduce number of occurences at reduce phase.
Some id occure very often. Probably bluetooth device is registered few times when car passes.
Or a cammera registers car few times when it is moving slowly in a traffic
------------------------------------------------------------
detection_window = 60seconds
map(log_rows)
    for(row in log_rows)
        time_diff = row['time'] - 1.1.2010
        time_in_seconds =  second(time_diff)
        window_number = time_in_seconds / 60
        key = (window_number, row['place'])
        result.add(key, row['id'])

reduce(key, values)
    uniq = unique_values(values)
    result.add(key['place'], key['window_number'], uniq)
```

Code snippet 4: The first stage of the Hadoop implementation.

```
------------------------------------------------------------
Second Stage
a_data and b_data are two data sets created by first map reduce stage
This stage should be performed two times -- for bluetooth and car travel data
At this processing point for every id ranking of coocuring id is created.
Map phase returns list of all coocuring ids for a given id.
At reduce phase all results are added to create ranking.
This ranking is then normalized, to create score that is related to probability of finding id after some id occurs.
This result is not final -- some id occure very often.
When bluetooth device is parked near detector it will be detected constantly.
It should be not claimed that this is correlated to anything.
------------------------------------------------------------
map(a_data, b_data)
    b_fast_lookup = create_map(b_data) # key is time and place, value is id
    for(occurence in a_data)
        related_occurences = b_fast_lookup[(time, place)]
        result.add(occurence['id'], related_occurences)


reduce(key, values)
    freq = calculate_element_frequency_on_list(values)
    normalized = change_frequency_to_probablty(freq)
    result.add(normalized)

```

Code snippet 5: The second stage of the Hadoop implementation.

```
------------------------------------------------------------
Third Stage
At this stage last results are combined to create accurate metric of occurence probability
Input is joined result forn last stage.
At this point previous results are combined to calculate score that is resistant to data noise.
Occurence chance of id with different id is cross validated by checking if the relation is strong the other way.
With this it is possible to renove id that are polluting data -- bluetooths that are positioned near detector and occure every second or id that are modified by
ITS system for legal resons.
------------------------------------------------------------
combine(prob_list_for_id)
    for(id_and_list in prob_list_for_id)
        list  = id_and_list['list']
        id = id_and_list['id']
        for(related_id in list)
            r_id = related_id['id']
            prob_list_for_id.find_list_for_id(r_id)
            final_result_score =   prob_list_for_id.find(id) * related_id['probability']
            result.add((id, r_id), final_result_score)
```

Code snippet 6: The third stage of the Hadoop implementation.

Results:
Speed was similar to python implementation. The map phase performed much faster as CVS implementation is faster and a job scheduling algorithm is better (no locks which are required in python, and better management of jobs). However, the reduce phase performed better on Numpy. Nonetheless, Hadoop is much more scalable for bigger data sets.

3. NUMERICAL TESTS

The setup of the numerical experiment is the following: for comparison of the execution time and results calculated with the implemented algorithms the calculations were performed for data from two working days from May 2014 registered in the Wrocław Intelligent Transportation System. The interval for data was limited in this manner since for sequential algorithms, calculations within longer intervals were very long. For the whole processing 1 580 628 Bluetooth records were entered and 665 547 containing license plate numbers from two days. After the execution of the data preprocessing procedure there remained 796 580 Bluetooth records.

Python sequential implementation
The output of this algorithm is a set of pairs (license plate number, Bluetooth MAC number), which were matched on condition that the Bluetooth MAC was registered within a +-5 minutes window around the registration of the license plate number. The next registration of the passing had to occur within the time distance of at least 2 hours.

Such an approach eliminates problems of ambiguity of the match of the Bluetooth MACs to vehicles driving in columns through several Bluetooth gates. In this approach the vehicle needs to occur at least 2 times.
A given pair (license plate number, Bluetooth MAC number) enters the results on condition that the Bluetooth MAC occurred exactly in all runs. It means that if the vehicle was going at least those without the Bluetooth device it was not added to the final results.

Transact-SQL implementation
The output of this algorithm is a set of pairs (license plate number, Bluetooth MAC number), which were matched on condition that the Bluetooth MAC was registered within a +-5 minutes window around the registration of the license plate number. The next registration of the passing had to occur within the time distance of at least 2 hours. Such an approach eliminates problems of ambiguity of the match of the Bluetooth MACs to vehicles driving in columns through several Bluetooth gates. These are the same conditions as for the Python sequential implementation.
The difference is that a vehicle needs to occur a prescribed in the algorithm number of times (here we assumed 2).
A given pair (license plate number, Bluetooth MAC number) needs to occur at least twice, but does not need to occur in all runs by a Bluetooth gate. Therefore there are differences in the results.

Python parallel implementation and Spark implementation
The output of these algorithms is a set of triples (license plate number, Bluetooth MAC number, a number of merit). The number of merit to detect a match is a product of normalized incidence of occurrence of a Bluetooth MAC number in a prescribed 10 minute time window on a Bluetooth gate with a given license plate number multiplied by a normalized incidence of occurrence of a license plate number in that time window with a given MAC number. With a use of threshold value one can detect
matches. Both these algorithms performed very fast. For instance for data from 14 days (other than those used for the algorithm comparison) performed within 10 minutes.
Results of the performance of all three implementations are presented in Table 1. This table shows performance times, number of matches found as well the percentage of the compatibility between three implemented approaches.

Table 1. Results of the performance of implementation of three approaches to license plate numbers and Bluetooth MACs matching.

Algorithm	Number of matches	Performance time	Compatibility of results with Python Seq.	Compatibility of results with T-SQL
Seq. Python	18425	10:39:12	--	60%
Transact SQL	187588	20:09:33	60%	--
Par. Python with number of merit on the level 0.01	16151	00:10:00	2%	12%
Spark with number of merit on the level 0.03	191	00:10:00	15%	68%

CONCLUSION

In this paper we presented four algorithms for matching Bluetooth devices with license plate numbers within ITS data that operates ANPR cameras and Bluetooth gates. The first algorithm was written in Python and the second one was in Transact-SQL. Both are sequential algorithms. The latter two algorithms are parallel – one implemented in python using multiprocessing and the second one using Apache Spark. Parallel algorithms gave the same results with the same parameters setup of number of merits used. The highest number of matches was found by Transact SQL, the next was sequential Python implementation, afterwards was the parallel Python implementation, the smallest number of matches was obtained by the implementation on Apache HaDoop. As seen from Table 1 concerning the compatibility level reaching up to 68%, the presented algorithms are three different approaches to matching of Bluetooth devices with license plate numbers.
Sequential algorithms uncovered many more matches, however – by using parallel algorithms we find that they are less reliable. Parallel algorithms are also far more efficient.
Presented parallel algorithms may be used to online matching of Bluetooth devices with vehicles. Such matches may allow the identification of car/gasoline thefts. Vehicles that took part in the crime may be identified by their Bluetooth device if they were matched beforehand by the proposed method.

ACKNOWLEDGEMENTS

The work presented in this paper was partially financed from grant 0401/0230/16.

REFERENCES

[1] Kolour, H. S., Shahbahrami, A.: An Evaluation of License Plate Recognition Algorithms. International Journal of Digital Information and Wireless Communications (IJDIWC) 1(1) in The Society of Digital Information and Wireless Communications, (ISSN 2225-658X), pp. 247-253, 2011.
[2] Kazagli, E.: Arterial Travel Time Estimation from Automatic Number Plate Recognition Data, Ph.D. thesis., Department of Transport Science, Division of Traffic and Logistics, Royal Institute of Technology, September 2012, Stockholm.
[3] Bazan, M., Bożek, M., Ciskowski, P. Halawa, K., Janiczek, T. Kozaczewski, P., Rusiecki, A.: Some aspects of Intelligent Transport System auditing. Archives of Transport System Telematics, Volume 8, Issue 3, 2015, pp. 3-8.
[4] Ciskowski, P,. Janik, A., Bazan, M., Halawa, K., Janiczek, T., Rusiecki, A.: Estimation of Travel Time in the City Based on Intelligent Transportation System Traffic Data with the Use of Neural Networks. Dependability Engineering and Complex Systems: Proceedings of the 11-th International Conference on Dependability and Complex Systems DepCos – ROLCOMEX, Ed. by W. Zamojski et. al., June 2016, Poland.
[5] Ma, Y., Chowdhury, M. Sadek, A., Jeihani, M.: Integrated Traffic and Communication Performance Evaluation of an Intelligent Vehicle Infrastructure Integration (VII) System for Online Travel-Time Prediction, IEEE Trans. on Intelligent Transportation Systems, 13(3), 2012, pp. 1369 – 1382.
[6] Jindal, V., Bedi, P.: Reducing Travel Time in VANETs with Parallel Implementation of MACO (Modified ACO). In: Snášel V., Abraham A., Krömer P., Pant M., Muda A. (eds) Innovations in Bio-Inspired Computing and Applications. Advances in Intelligent Systems and Computing, vol 424. Springer, 2016, Cham.

[7] Wang, Y., Malinovskiy, Y., Wu, Y.-J., Lee, U. K.: Error modeling and analysis for travel time data obtained from Bluetooth MAC address matching, Research Report, Agreement T4118 Task 46, 2011-01, Washington State Transportation Center (TRAC), December 2011.
[8] Park, J. H.: Optimal Number and Location of Bluetooth Sensors Considering Stochastic Travel Time Prediction, The 56th Annual Transportation Research Forum, March 2015.
[9] Wieck, M.: Use of Bluetooth Based Travel Time Information for Traffic Operations. In: ITS America, the 18th World Congress on Intelligent Transport Systems, Orlando, Florida, United States, October 16-18, 2011.
[10] Hainen, A., Wasson, J., Hubbard, S., Bullock, D. M.: Estimating Route Choice and Travel Time Reliability with Field Observations of Bluetooth Probe Vehicles.Transportation Research Record Journal of the Transportation Research Board 2256(-1), December 2011, pp. 43-50.
[11] Senkar, B., Karau, H.: Fast Data Processing with Spark - Second Edition, ISBN-13: 978-1784392574, Packt Publishing, 2013.
[12] Wadkar, S., Siddalingaiah, M.: Building Real-Time Systems Using Hbase, in Pro Apache Hadoop, ISBN: 978-1-4302-4863-7, 2014, pp. 293-323.
[13] Perera, S.: Instant MapReduce Patterns – Hadoop Essentials How-to Progressing, Packt Publishing, ISBN 139781782167709, 2013.
[14] Karanth, S.: Mastering Hadoop, Packt Publishing, ISBN-13: 9781783983643, 2014.
[15] International Transport Forum, CPB, OECD, Big Data and Transport: Understanding and Assessing Options, 2015.

ALGORYTM IDENTYFIKACJI POJAZDÓW POPRZEZ URZĄDZENIA BLUETOOTH WYKORZYSTUJĄCY NARZĘDZIA BIG DATA

STRESZCZENIE

Współczesne pojazdy wyposażane są w wiele różnych urządzeń Bluetooth, które logują się do infrastruktury ITS za każdym razem gdy przejeżdżają one w zasięgu czytników Bluetooth. Położenie czytników Bluetooth jest zagadnieniem istotnym dla metod predykcji czasu przejazdu w regionach zurbanizowanych. Technologia Bluetooth jest użyta do poprawy dokładności czasu przejazdu i jest uzupełnieniem dla identyfikacji pojazdów po numerach rejestracyjnych. Algorytmy do predykcji czasu przejazdu są używane do proponowania użytkownikom trasy alternatywnej w celu przejazdu przez najbardziej zatłoczone regiony miasta w sposób najbardziej efektywny. W artykule jest prezentowana implementacja algorytmu, który pozwala połączyć urządzenia Bluetooth i telefony znajdujące się w pojazdach z samymi pojazdami. Do tego celu angażuje się narzędzia Big Data takie jak Apache HaDoop i Apache Spark. Do zbudowania zadań Map-Reduce używa się Hive-SQLa. Algorytm był testowany na danych z wrocławskiego ITS. Wyniki działania algorytmu mogą być użyte do lokalizowania skradzionych pojazdów.

SŁOWA KLUCZOWE

HaDoop, Spark, identyfikacja użytkownika