6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

Intelligent urban traffic development support system—the simulation software and the database*

Attila Fazekas, Lajos Kollár, Zoltán Zörgő, András Hajdu, János Kormos, Krisztián Veréb

Institute of Informatics, University of Debrecen
e-mail: {fattila,kollarl,zorgoz,hajdua,kormos,sparrow}@inf.unideb.hu

Abstract

In this paper we present the architecture an urban traffic surveillance system based on smart sensors capable of license plate recognition. A data model that is suitable to store data coming from this system is also discussed. This data model needs to be connected to the spatial description of the underlying road network.

Due to the relatively high cost of the sensors, a software for simulating vehicle movements and detection by the sensors is needed, as well. We have developed two applications for this purpose.

Categories and Subject Descriptors: H.4 [Information Systems Applications]; J.7 [Computers in Other Systems]; K.4 [Computers and Society]; E.1 [Data Structures]

Key Words and Phrases: Traffic simulation, Data model

1. Introduction

An increasing problem in the modern world is the rapidly growing urban traffic. The highly populated cities are not prepared to ward off the effects of this growth, such as traffic jams, flow slowdown, not speaking about the environmental pollution and the psychological effects. In most cases the lack of space is the major issue. The public road development as part of the urban planning has to be performed in the most appropriate way.

In an urban environment vehicle counting alone may not be enough to acquire satisfying information about the behaviour of traffic participants, their route choos-

*This work was supported in part by the NKFP2 grant 2/2002, and by the OTKA grants T032361 and F043090.

ing habits. Knowing not only quantitative information, but concrete vehicle paths, we can determine more precise origin-destination profiles, patterns and trends in the traffic. Therefore we intend installing sensor devices capable of recognizing license plates. These sensors will be placed on previously selected lanes in a well delimited area. Knowing the precision of the devices and simulating the urban traffic in that specific area, we can propose the minimum number of sensors needed in the actual data acquisition, and the position of the sensors.

Our aim is to create an intelligent urban traffic development support system to monitor urban traffic, and to predict traffic behavior. To reach this goal a project has been started with a consortium (the members are the Adaptive Recognition Hungary Ltd. [1] and the Institute of Informatics, University of Debrecen) to manage the development and deployment. In the last years the work has been separated into two large parts, namely the development for software and database techniques and simulation software tools and a development for a statistical model. The second part is explained in [2].

Our goal is to collect information about the urban traffic which can be the starting point of a statistical analysis which provides useful data for the designers and maintainers of the public road network in order to use them for both tuning the legacy network and further development.

This paper is organized as follows. Our 4-layered system architecture is discussed in Section 2. Section 3 describes the data model used for storing data acquired by the sensors. In Section 4 we discuss the need for a simulation software and two implementations are proposed. Section 5 contains concluding remarks.

2. Architecture

The architecture of our system is a 4-layered one. Each layer is based on the layer below it and provides services for the above layer.

The bottom layer is the *physical road network* itself.

Spatial model stores the physical location of both the roads and sensors. It provides an abstraction of the road network. It is considered as a "digital picture" of the underlying road network. Sensors form a directed graph called sensor graph. These sensors are considered to be *smart*: they are not only counting the number of cars but gather more information about vehicular traffic, including license plate identification of the crossing vehicles and (optionally) the photo of the license plate.

Since statistical model needs aggregated data which is not supported by the spatial model, we need a new layer, the inputs of which are the data gathered by the sensors and provides the required aggregates.

Hence, collected information from each sensor are inserted into a *database*. However, sensor graph can be derived from the spatial model, the need for faster processing suggests to store the graph in a database, as well. Therefore the database, besides storing data produced by the sensors, contains data about the sensors, as well, which need to be connected to the sensor descriptions of the spatial model.

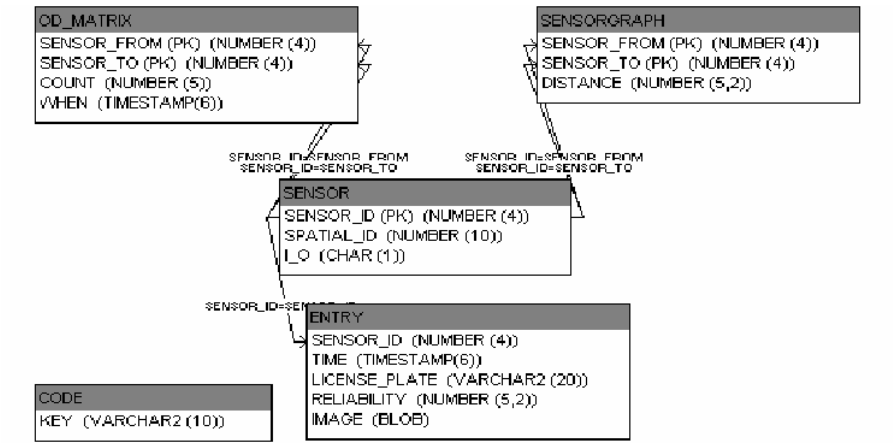


Figure 1: Data model used for storing sensor data

Data are collected in order to create a forecast, e.g., about the preferred paths of drivers at Easter, or the average speed of vehicles in the morning, etc. To do so, complex *statistical computings* are needed. These issues are discussed in detail in [2].

3. Data model

As we mentioned, data collected by the sensors are inserted into a database. Each sensor has a unique ID, knows the timestamp of the photo, the recognized license plate of the vehicle, and optionally, the reliability of the recognition and the photo itself (in JPG or BMP format). These data serve as a base for the statistical analysis. To store the gathered information we have developed the data model shown in Figure 1.

This model is based on the relational model for databases [3]. Therefore, all the entities of Figure 1 are mapped to a database table and SQL should be used for data manipulation.

Derived data from the spatial model should be stored, as well. The physical location of sensors is given by the spatial model. To achieve independence of the spatial model, sensors are assigned unique identifiers which can connect to the spatial model using a database table serving as an interface between the spatial and the data models. This table needs to map each sensor as a spatial model element to a generated ID. This solution makes the data model independent of the underlying spatial description.

`SENSOR` stores the information about sensors. We have three sensor types used when computing origin-destination (OD) matrices: input, output and pass-through.

As we mentioned before, sensors form a sensor graph. This can be derived from

the spatial model but for faster processing we need to store it in the database, as well. `SENSORGRAPH` stores which sensors have direct connections and the distance of them. The latter can be used for computing average speed of vehicle flow, for example.

`ENTRY` stores data acquired by sensors. For each vehicle, the time of recognition is recorded as a timestamp which can be used for computing trends and seasonalities. The result of the recognition is a license plate. The reliability of the recognition and the image taken may also be provided by the sensors. The former allows the statistical correction of the erroneous data caused by the incidental incorrect recognition.

The table `OD_MATRIX` is used from time to time for aggregating information needed to compute OD matrices.

`CODE` is a singular table (its singularity is maintained by a database trigger) used for security reasons but its detailed discussion is outside the scope of this paper.

3.1. Functions, services

Logical data independence as a general principle in database design suggests to have the data model independent of the other parts of the system which results in avoiding data manipulation directly in the tables (i.e., the subsystem controlling sensors should not insert new entries using direct INSERT SQL statements, and the statistical model consuming the produced aggregates should be prohibited from direct SELECTs). An interface should be defined through which other parts of the system can access database services such as determining average speed between two sensors or computing the OD matrix.

The interface can be implemented using stored subprograms because the change of the database structure results in refactoring this code only and the other parts of the system are not affected.

Besides trivial services, such as adding, re-locating or deleting a sensor, querying sensor type or deleting obsolete data, this interface should provide all the information which are needed for the statistical analysis. Production of OD matrices, determining both the average speed of a vehicle and the number of appearing/disappearing vehicles, and appointing the flow rating of a given sensor are included in these services and they need to be extensible.

3.2. Security issues

Since personal data are stored (the technology allows the tracing of whole paths of given vehicles), it is very important, how data are protected against incompetent access. Although data are stored itemised (i.e., there are entries for every recognized license plate), no one should be allowed to access these detailed data. Only the number of vehicles are important so aggregated data need to be provided, i.e., one can get information about the number of vehicles on a given path but cannot access data of concrete ones.

We have created an interface on top of the data model which consists of several stored procedures and functions. For security reasons, only the interface is allowed to manipulate itemised data, i.e., no one but the interface is authorized to access database tables directly. Thus, data protection is played back to the security mechanism of the underlying DBMS.

Since sensors are intended to send data to the database over network, data need to be secured not only in the database but in the course of network communication, as well. Accordingly, license plates should be encrypted using cryptographical methods, e.g., DES3. Decryption should be done before inserting into the database and the proposed interface approach provides an easy way to do so.

3.3. Implementation

The proposed data model can be implemented in any RDBMS or ORDBMS, e.g., Oracle, MySQL, PostgreSQL, etc. For implementing interfaces, it is subservient to choose such a DBMS which supports stored subprograms. If they are not supported by the DBMS, interface can be implemented in a high-level programming language, e.g., Java or C++, which accesses the database via JDBC or ODBC.

For our implementation we have chosen Oracle9*i* as a back-end because of its high performance, wide support for aggregations (data warehouse operations) and its powerful PL/SQL [4] language for creating stored subprograms.

4. Simulation software

For testing our system we needed many sensors to be deployed. As they are "smart", e.g., equipped with a camera, their costs are relatively high. For this reason a simulation software has been developed which is able to simulate not only the movement of vehicles but license plate recognition, as well. Beside these, the program has to be able to generate common traffic events, and to follow some of the behavioural differences between drivers. This kind of software can be used not only for testing purposes but tuning the minimum number of sensors to be deployed to a given area.

There are some traffic simulation projects, but this last feature is not included in any of them. So we need to develop our own software for this task.

The results of the simulation will be used not only in planning the data acquisition, but in refining the requirements for the sensors, which are currently developed by our partners.

4.1. Specifying requirements for the simulation software

First of all we need to specify the desirable features for the software. Some of these requirements are absolutely necessary; some of them could help in a more accurate and more realistic urban traffic simulation.

4.1.1. Simulating road system and traffic

It is a basic feature to store a *model of the road system* in an easily expandable way. A too complicated storage may cause performance problems. As we do not intend to simulate whole cities, only smaller regions of a few square kilometres, we do not calculate with large storage needs. The internal format must contain some geographical positioning information about the road sections. The ability to identify the lanes separately is also a must. For a more precise driver behaviour simulation, the visibility and terrain conditions could also be represented. For an easy data exchange with the local Transportation Departments, the application should have some built-in import features for the most commonly used *map formats*. As we noticed no trends in this matter in Hungary, we haven't focused on none of the alternatives.

As streets are either *one-way* or *two-way* ones, we cannot ignore this in a simulation. But as we all know, there are some exceptions too—like vehicles with distinctive flare. It would be interesting to simulate a traffic situation in which police or fire engines are involved.

As urban traffic is happening in time, and it is far from being constant in time, many statistical descriptors need to be calculated for specific time intervals. Thus we need to simulate the *timeline* too, even if the virtual time flows quicker or slower as our one. Thus, a timestamp can be attached to the data acquired.

As in real life, traffic flow may be controlled with *traffic lights*. When implementing this feature we need to able to simulate concrete traffic lights or light system programs. We need to identify the lights controllers separately to be able to simulate the outages, or the overnight caution light program.

It is obvious that not all vehicles have the same *speed*, thus we shouldn't ignore this. This involves some very complicated problems, like overtaking, and this, on it's turn involves the need of representing the size and some other physical parameters of the vehicles. We have to mention, that in the current state we have ignored this attributes, and we have calculated with some average values.

As traffic in general is guided by *traffic signs*, urban traffic is highly influenced by traffic rules. These ones cannot be ignored. It is not absolutely necessary to actually store the traffic signs, but to incorporate their effect in the internal representation. Nevertheless, the first alternative is more obvious. As not all drivers respect these regulations, why should a simulated driver be different? The simulation of a realistic urban driver is a very challenging problem, involving artificial intelligence and psychological research. In the current stage we reduced this task onto specifying some stochastic parameters for the virtual car objects.

4.1.2. Simulating data acquisition

As mentioned above we need to simulate the activity of the proposed sensors and the data acquisition process. This way—with regard to, and using the chosen statistical methods—we can give some guidelines for placing the physical sensors, the number of sensors needed, and the expectable result reliability.

First of all it is necessary to specify the position of the virtual sensors. The manual placing of the simulated sensors is inevitable, but in some cases, beside the manual one, random placement could be helpful. Automated full-range coverage with sensors of a selected region is also useful, because the starting point for the optimisation may be such a fully covered state.

Besides their own optical range limitations, there can be situations in which part of the field of vision of a sensor is permanently covered by field objects. So, when specifying position—including the height over the road surface—and the viewing direction of the device, the actual range could be also specified.

To track the traffic flow in commonly used routes consisting on many consequent streets, virtual path definition could also be useful.

4.1.3. Exception event simulation

There are normal events occurring during a journey. Some of them are actual traffic events, like jam, or lane lockdown. These events will come trough as irregularities in the data flow. Another sort of exception is caused by hardware or software error. As such, not all exceptions are errors. Let's take for example the case when a car disappears. That means it is not seen by the sensors for a while. The most common event this represents is that the vehicle has ended its itinerary, and it stopped somewhere between two sensors. But of course this can be caused by misrecognition of the license plates too.

It can happen that a vehicle jumps over a sensor. This can be an error in the recognition, or the license plate may temporary covered by another car, so the car is not "seen" by sensors.

It is possible that a car stops, but after a while, it is back again. This situation should not affect the calculation of the average time for a street.

But some events are real errors occurring in the software of the sensor devices, the hardware or in the network.

So we can see that the simulation software should be able to generate such exception events, but handling such data by the statistical model is a very challenging problem.

4.1.4. Simulation output

The simulation software is expected to provide well formatted output data, which can be used as input for the statistical analysis on one hand, and can be interpreted by humans to follow the events standing as base for the data. Thus we expect more than one output files.

The main data file must have the same format and content logic as the files acquirable with the sensor system. The timestamp contains common hour–minute–second time information. When simulating traffic over many days, the date information must also be included. If the sensor devices are capable of generating the same identification data for a specific license plate, it is not absolutely necessary to store the actual license plate content. This could be another improvement in

security. The reliability factor is calculated by the sensor itself for every recognition separately. This "self-rating" should also be simulated. It can happen to have a correct sensor recognition, but low level of sureness. To implement this feature, we need to know the recognition algorithm.

Another data file can contain information about the events generated by the simulation.

4.2. Implementations

Because our simulation needs contain very special requirements, we found no appropriate software available for the public. That is why we began to implement different combinations of features. At this point we have two implementations. The first one is a GUI (Graphical User Interface) based Windows program, the second one is a console application.

4.2.1. TrafficSim

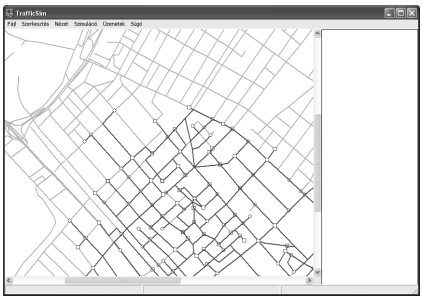
As mentioned above this software is a graphical application. It was developed in collaboration with our partners, the Adaptive Recognition Hungary [1] Ltd. This application supports MapInfo Interchange Data Form-style file import. Traffic flow can be predefined for a simulation but we can use random traffic too. This can be stored and reloaded to be investigated under other conditions. The program can transmit using TCP/IP communication recognition event data to a server application. The users can define virtual paths with adjustable flow weights. Sensors can be placed on different parts of a road; the height from the ground can be specified too. Users can Save and Load virtual paths and the position of the sensors.

In the initialization file we can specify parameters for the recognition error simulation: missed recognitions, bad character recognition, extra tailing characters, extra leading characters, extra characters in the middle and missing characters at the beginning, the end and in the middle of license plates.

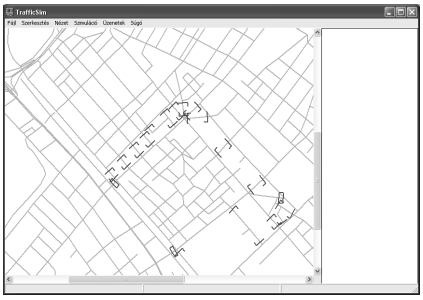
Screenshots of the application are shown in Figure 2. The spots in Figure 2(c) represent the virtual vehicles. During the simulation we can see the vehicles moving. A record to the "node" log file is added every time a vehicle reaches a road junction. The "sensor" log file contains the sensor identification, the actual license plate text and the simulated recognition. The "path" log's entries consists of the nodes hit by the vehicles.

4.2.2. CrossRoads

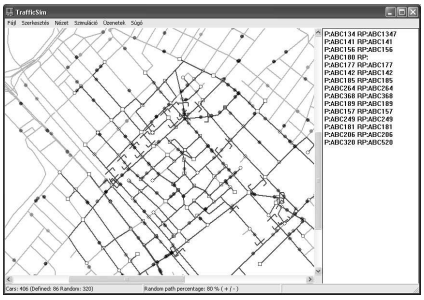
Features. This software is a console application, currently with no interactive interface. Road-system and simulation parameters can be specified in an XML file with special DTD. The application can simulate traffic light programs, supports multiple lane for a road section. It is timeline based. The user can define nodes to be reached by a vehicle. Every vehicle (driver) has its own affinity to quicker



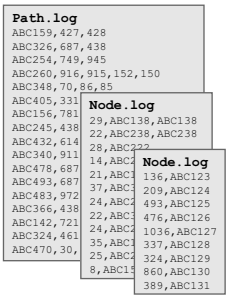
(a) Virtual paths loaded



(b) Sensors loaded



(c) Simulation in progress



(d) Fragments from output files

Figure 2: TrafficSim screenshots

or shorter tracks. This is a stochastic threshold values upon which the program chooses the direction in the intersections.

The simulation process. The database of the program consists of the lists of the vehicles, intersection, road sections and sensors. The program picks every vehicle one by one and checks the state of them. A vehicle can be either moving towards a node or waiting at an intersection. The position (the lane on which it is, and the relative position from the lane end) of the vehicle is stored in the vehicle object.

If the vehicle is on its way on a lane, the program checks whether it's reaching the end of the section. In this case chooses a new direction according to the predefined route or, if not present, the affinity of the driver. Otherwise checks if the vehicle has reached a sensor. In this case the virtual recognition is performed, and the result is logged.

If the intersection is controlled by traffic-lights, the vehicle is enrolled in the queue, according to its direction. During the wait-state the vehicle is idle. If there is no traffic light or finds clear signal, then the vehicle is passed to the desired road. If the vehicle reaches an output node, then it is removed from the active vehicle list.

The simulation ends if there are no more vehicles on the roads, or it is inter-

rupted.

The input file format. As mentioned before, this implementation uses as input an XML (Extensible Markup Language) file with special DTD (Document Type Definition). This way the introduction of new features is very simple. The definitions contain human-readable information besides the relational and other technical ones.

5. Conclusions

The development of easily expandable and configurable traffic simulation software is a must for our project in order to simulate real traffic events, vehicle movements and sensor detection. The database gets the simulation output transparently as it would come from real sensors. This allows the switching to real sensors without affecting both the data model and the statistical model using the services of the data model.

If the traffic simulation software is developed according to well-defined requirements, such a software could be used in other projects too. Our implementations do not meet all desirable requirements, but they were suitable for the basic statistical models. Our future aim is to create a more realistic, and more accurate software for traffic simulation.

References

[1] Adaptive Recognition Hungary Ltd., `http://www.arhungary.hu/`

[2] András Hajdu, János Kormos, Krisztián Veréb, Attila Fazekas, Lajos Kollár, Zoltán Zörgő, *Intelligent urban traffic development support system—statistical approach*, 6^{th} ICAI, to appear.

[3] Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems, Fourth Edition*, Addison-Wesley, 2003.

[4] Steven Feuerstein, *Oracle PL/SQL Programming, Third Edition*, O'Reilly, 2002.

Postal address

András Hajdu, Attila Fazekas, Lajos Kollár, János Kormos, Krisztián Veréb, Zoltán Zörgő
Institute of Informatics
University of Debrecen
P.O. Box 12, H–4010 Debrecen
Hungary