Graph Algorithm Design for a Local Transportation Application

K. Zsobrák * and L. Gugolya**
* Óbuda University – Alba Regia Technical Faculty, Székesfehérvár, Hungary
*zsobrak.krisztian@gmail.com
**gugolya.laszlo@amk.uni-obuda.hu

***Abstract*—In our modern world, the importance of mobile applications is increasing. These apps can improve our comfort, we can obtain the needed information faster, independently from our location. Such applications use more complex mathematical algorithms, like graph theory, and therefore graph algorithms.**

A special graph build algorithm will be presented in this paper, which is based on local bus transportation of Székesfehérvár. The purpose of this algorithm is to find the best route from the start bus station to the destination, even by using multiple bus lines. The bus schedule is stored in a database, in a specified format. The graph is created from this data, by an algorithm described in this paper. The search can be performed in this graph, which results in a path that contains the bus travel information. The difficulty of such route-finding algorithms, that it has to operate in both space and time dimensions, because moving from a location to another can only be performed in specified times.

I. Introduction

In our fast world we need applications, that makes our life easier. With these software we feel, we are not required to remember unnecessary information, because when we need that, it is available on our mobile device. Therefore, it came to our mind, we implement a route-search, route-plan application based on existing similar solutions, but for Székesfehérvár's local transportation.

At now, one can get the information of lines by a static website of KNYKK Company (Fig. 1).

Figure 1. – Static line's information of Székesfehérvár

Another interface is a map based route planner, where the user can select the start and the finish bus stop on the map. (Fig. 2)

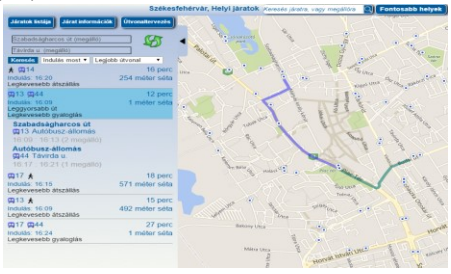

Figure 2. – Map based route planner of Székesfehérvár

But this website is not optimized to mobile devices and tablets, therefore it falters, and drastically slow on them. Also it is not responsive, and not suitable for use on the go.

Our example to follow is the BKK Futár solution, thanks to its clean design and useful functions. The Futár has a website and a mobile application, both with similar functionality for the users. It not forces the user to select the start and finish station on the map, but he or she can enter it into a text field. It can also search by street addresses.

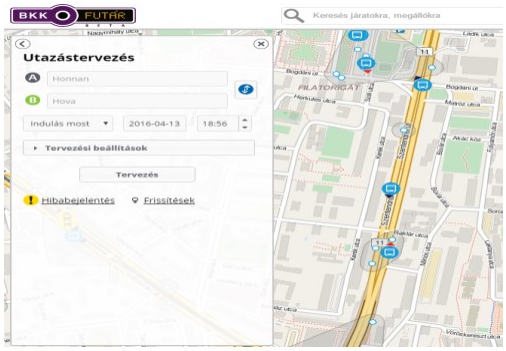

Figure 3. – Screenshot of BKK Futár website.

We choose to implement the application to Android operation system, therefore Java language and SQLite database is required.

II. Transport Information Services

Passenger information services, for example dynamic route-planning applications are designed to inform people about when and which line should they take. The base of this paper is the local transportation of Székesfehérvár, where only bus is available.

Generally, three parameters are required for the route-planner algorithm in order to generate useful data for the user: The departure location, the destination, and the departure time. The output of the algorithm is a list, from which the user can select the best. This has to contain the departure and destination time, the used lane's name, and when and where he or she should change.

Google Maps is capable of planning local transportation trips, and this feature is also available on mobile devices. The necessary GTFS based database have to be uploaded to the Google servers, and this function is accessible to everyone. But this software can be slow on older devices, which are not designed to have the requirements of the new versions of Google's software. The application mentioned in this paper (extended by many search optimization) can be faster on these devices, and independent from having an internet connection.

III. Database Plan

To create this search algorithm, the schedule data must be stored in a database. Nowadays a very popular transportation data format is the GTFS (Google Transit Feed Specification), which is very comprehensive, and can store complex schedules. Compared to Székesfehérvár's simple bus based transportation, it is easier to store data in the following database structure, but it is easily convertible to GTFS, which I plan to do after the schedule is fully uploaded.

A. Stations, Bus Stops

A *station* has a name, which is known by the passengers, and this appears in the schedule. A *stop* is a place, where the bus actually stops, and people can get on or off. The schedule does not contain the stops, but it has to be stored in order to display it on map. Because often multiple stops are belonging to a station, they create a 1:N relation. These are stored in the following relational database tables:

- Stations:
  - id
  - name
- Stops:
  - id
  - stationid
  - coordinate

B. Lanes and Lines

In Székesfehérvár people know bus *lines* by their number, which starts from 10, and ends with 44. They usually on the same route back and forth, but there are exceptions like fast lines, and circle lines. A line may include multiple *lanes*.

- Line
  Contains multiple lanes, has a publicly known number, such as 10, 11, 15. This data will be displayed to the user.
- Lane
  A lane has a well-defined path with station names, and times but there is no public identification for it. For example: Train station – Metal industry – Bus station. At least one lane is assigned to a line

They database structure is the following:

- Lines
  - id
  - name
- Lanes
  - id
  - laneid
  - mark
  - description

C. Paths

A path of a lane is simply the list of the stations where it stops, and the time it takes to get there from the departure time. These lists will be stored in a single table, because filtering by lane or station is easier.

- Path
  - id
  - laneid
  - stationid
  - minute

D. Departure times

The buses on lanes depart on specified times. These are always given in an hour-minute format. Smaller or larger time unit are unnecessary. For the simplicity, it is easier to store this time value in the number of minutes from midnight. If a p time is equal to h hours and m minutes, then p equals to h times 60 plus m.

- Departure
  - id
  - laneid
  - time
  - daytype

IV. A Sample Schedule

Because Székesfehérvár's schedule is complicated, here's a sample one (Fig. 4), in order to visualize the methods in this paper.

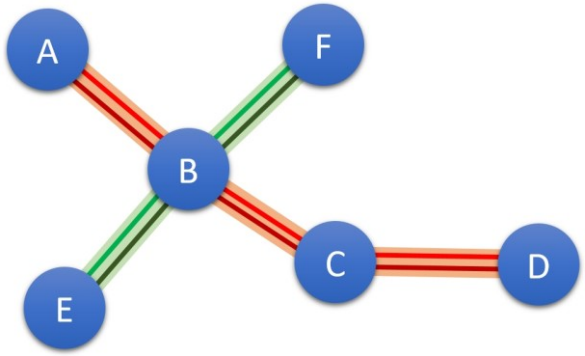

Figure 4. – A sample city

This sample city has 6 stations, labelled from A to F. There are two lines: 10 and 20. Each line has two lanes, one going forward, and the other backwards. The goal is to create an algorithm, that can easily find a route from station A to F.

A. Schedule

This is a list of the lines and lanes of the sample schedule. The list of stations and minutes are in pairs, showing the required time by the bus to get to that station from the departure time, in minutes.

- Line 10:
  - Lane 10-f:
    - Stations: A, B, C, D
    - Minutes: 0, 2, 6, 8
    - Departs: 8:00, 8:20
  - Lane 10-b:
    - Stations: D, C, B, A
    - Minutes: 0, 2, 6, 8
    - Departs: 8:10, 8:30
- Line 20:
  - Lane 20-f:
    - Stations: E, B, F
    - Minutes: 0, 2, 6

- Departs: 8:05, 8:25
- Lane 20-b:
  - Stations: F, B, E
  - Minutes: 0, 4, 6
  - Departs: 8:15, 8:35

From now on, I will use this sample data in the examples.

V. Building the Graph

Graphs in discrete mathematics, more specifically in graph theory, are mathematical abstractions, which contains two sets, nodes and edges, where a pair of nodes are interconnected by edges. A large number of graph types are extension of this definition. For example, a directed graph has edges, which node pairs are ordered, therefore a direction is assigned to it. The nodes and edges have a unique identification, usually natural numbers. This leads to assign multiple records of information to a graph object. This thesis defines this kind of graphs.

We know where a bus will be at any given time from the schedule. To display the location simpler, only one dimension is required, which will be the station identification number. Timeliness also requires only one dimension; which metric is the minute. These values are easier to present in a two-dimensional grid (Fig. 5).

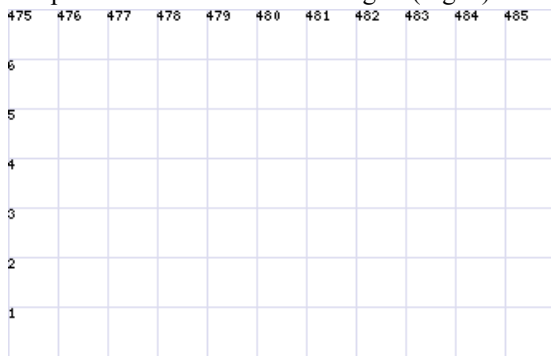

Figure 5. – Default grid

First add the nodes. We know the departure time, and where and when will the bus stop. For example, let's take lane 10-f, at 8:00 from the sample schedule. It departs from station A at 8:00. It takes 2 minutes to get to station B, therefor it is in B at 8:02, and so on. Add these nodes to the grid (Fig. 6.).

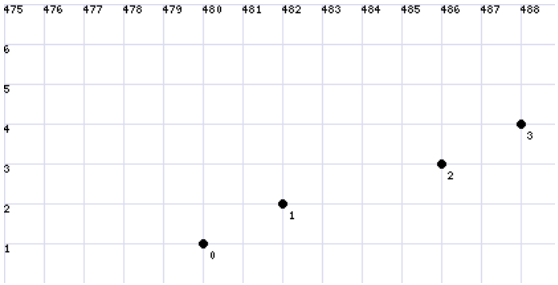

Figure 6. – Grid with a line's nodes

Connecting these nodes with directed edges will mean, that one can travel by bus between these points in space and time, but only in forward direction. The edges must contain the bus lane id data, and its weight, which is its time dimension length.

Then add all the nodes by iterating through all of the departure times, and connect them likewise. After that we got the full map of bus lines. (Fig. 7.)

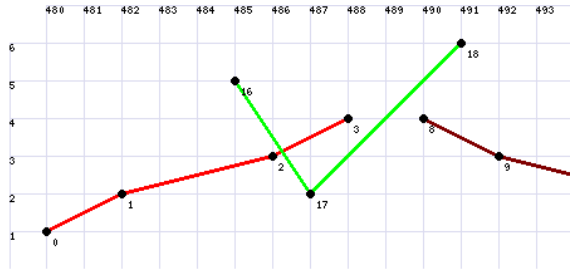

Figure 7. – The map of bus lines

Apart from that the figures are in a grid, this is a regular directed graph, with information assigned to its edges and nodes. But this is yet incomplete. For creating a simpler search algorithm, the graph must be extended by adding edges that connect station nodes in a chronologically ascending order. (Fig. 8.) These will represent the waiting at a station.

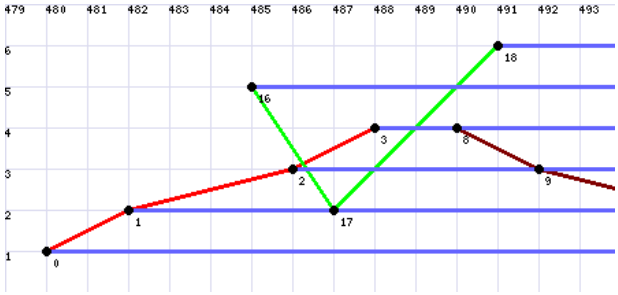

Figure 8. - The extended graph by the waiting edges

An example: Travel from station A to station F could not be done without change at station B. The shortest path would be node 0, 1, 17, 18.

VI. Graph Search

At the beginning of the search, the algorithm knows the start and finish station, and start time, but does not know the start nor the end node. For the better performance, a trip-end-time must be created, which the search algorithm cannot exceed. For example, 90 minutes from the start time. The start node is easier to find, because it will be the first that has the department station as station data, and its time data is greater or equals to the department time. The algorithm must try multiple start nodes between the start and end time to get better results.

When a start node is selected, a non-informed search has to find the nearest node, which has the arrival station as station data. The nearest node has the lowest weight path from the start node. This could be done by breadth-first search, or depth-first search.

However, this algorithm has to be fast with relatively low memory complexity, due to the mobile environment. Series of optimization has to be done on the graph build, the search algorithm, and the model of classes.

VII. Conclusion

In this solution, the mobile app is independent from other route planning services, efficient in offline mode, and gives more freedom for the developer. The application can be lightweight, and free from unnecessary services. But a possible drawback could be the lack of integration with other applications.

There are multiple designs for the search algorithm, that give the best route for the user, but they have to be

implemented and tested for memory and time complexity. Numerous optimization methods are planned, but these will be significant, when the full 100 lane schedule of Székesfehérvár is uploaded to the database, also which size is yet unknown. Based on the performance on mobile devices, the location of the calculation has to be taken into considered too. A device-based method gives the feature of offline route-planning, but the database has to be copied on modification. The server-based has the benefit of reliable calculation time, and address geocoding for departure or destination location, but slow internet connection can greatly affect the performance.

REFERENCES

[1] M. Braga, M. Y. Santos, A. Moreira, "Integrating Public Transportation Data: Creation and Editing of GTFS Data", *New Perspectives in Information Systems and Technologies, Volume 2, 2014,* pp 53-62

[2] T. Teorey, S. Lightstone, T. Nadeau, H. V. Jagadish, *Database Modeling and Design: Logical Design*, 2011

[3] R. J. Trudeau, *Introduction to Graph Theory*, 1993

[4] I. Fekete, T. Gregorics, S. Nagy, *Bevezetés a mesterséges intelligenciába*, [Introduction to Artificial Intelligence] 2006