Distributed Systems seminar: Shortest-path problem continuation

Heiki Pärn
Supervisor: Amnir Hadachi

Abstract

In graph theory, the shortest path problem is the problem of finding a path between two vertices in a graph such that the sum of the weights of its edges is minimized.

In last semester's report "Distributed Systems seminar: Shortest-path problem[1]" the previously created program was introduced that was able to perform shortest path search using A* algorithm on OpenStreetmap data.

There were many areas for improvement in the previous program. This report contains the numerous improvements and changes made to the program.

I. Introduction

The program developed had several issues both on the usability and performance side. The new version is a web service instead of a standalone program.

The repository is available at [2]. It is also running on Amazon microinstance at [3] .

II. A* algorithm

The program uses A* algorithm to find the shortest path. It is a search algorithm that is widely used in path finding and graph traversal.

As A* traverses the graph, it follows a path of the lowest known cost, keeping a sorted priority queue of alternate path segments along the way.

If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment instead. This process continues until the goal is reached. [4]

The algorithm uses a heuristic which has to be *admissable*. This means that it must not overestimate the distance to the goal. In this program the heuristic used is distance from goal-node.

The distance between two coordinates is calculated using the *Haversine* formula. Haversine formula gives distances between two points on a sphere using their longitudes and latitudes. This[5] implementation was used in the program.

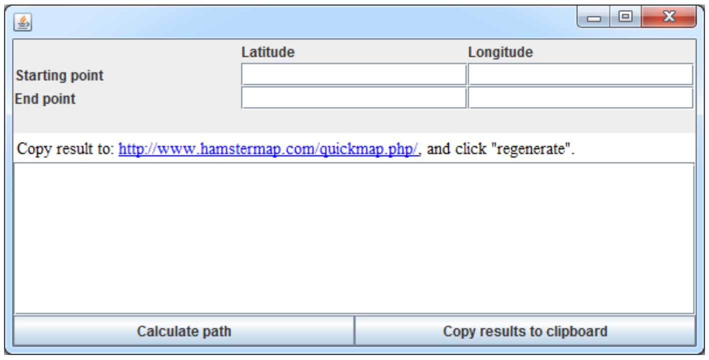
The A* algorithm implementation itself is based on the pseudocode from wikipedia[6].

III. Issues in previous program

In the old program there were several problem areas. The main one was long startup time. It took around half an hour to start up using the map of Tartu, which was clearly too long.

The lengthy startup time was the result of unprocessed map data and some optimization problems in the code. Instead of discarding irrelevant map elements during startup it would be more efficient to use a specialized OSM-data processing tool to preprocess the file.

Another issue was the fact that input and output was done in coordinates. This made it very inconvenient to use. It also made the program reliant on other applications to display the results.


Latitude
Longitude
Starting point
End point
Copy result to: http://www.hamstermap.com/quickmap.php/, and click "regenerate".
Calculate path
Copy results to clipboard

Picture 1: Old UI

In picture 1 above you can see the old interface of the program. It is not very user friendly.

IV. New features

The main features of the new version include:

- Client-server architecture
- Data preprocessing
- New interface for getting input and displaying results
- Optimizations
- Road priority

A. Client-server architecture

The program was modified into a client-server application instead of a standalone program. The server side uses Java servlet technology to respond to client requests. A Java servlet is a Java programming language program that extends the capabilities of a server.

It is easy to export the program as a .war file and deploy it on Tomcat server. Apache Tomcat is an open-source web server and servlet container developed by the Apache Software Foundation.

B. Data preprocessing

To recap the previous report, OSM map file is in XML format. It consists of *nodes* and *ways*. A node is a single point in space defined by its latitude, longitude and node id. Ways are relations between nodes.

An example of a node:

```
<node id="8220937" lat="58.3790611" lon="26.7303632" version="6" timestamp="2011-03-19T12:36:44Z" changeset="7604312" uid="156900" user="k__"/>
```

An example of a way:

```
<way id="33239457" version="5" timestamp="2014-08-23T19:58:51Z" changeset="24964005" uid="357111" user="enedaniel">
    <nd ref="3029578499"/>
    <nd ref="330041270"/>
    <nd ref="3029578545"/>
    <nd ref="377108084"/>
    <nd ref="3029578257"/>
    <nd ref="377110823"/>
    <nd ref="330042739"/>
    <tag k="highway" v="residential"/>
    <tag k="name" v="Lääne"/>
</way>
```

Osmosis[7] was chosen as a separate tool to preprocess map data. The filtering parameters used in this project were:

```
osmosis --read-xml input.osm --tf accept-ways highway=motorway,trunk,primary,secondary,tertiary,residential,motorway_link,trunk_link,primary_link,secondary_link,tertiary_link,motorway_junction,mini_roundabout --used-node --write-xml output.osm
```

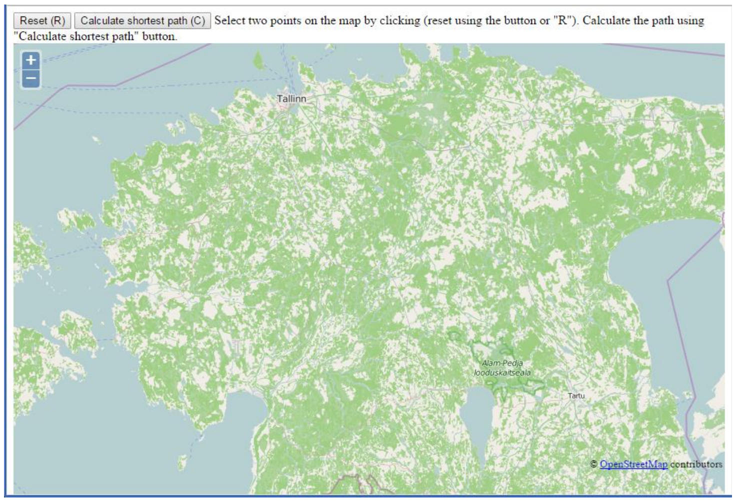
This filters the data so that the only *ways* left in the map file are the ones specified – highways which have a tag *motorway*, *trunk*, *primary*, etc. All irrelevant *ways* like footways, paths and so on, are removed. More information about different types of highways can be found at [8]. As a result of this the map file size generally decreased by over 90%.

C. Client side interface

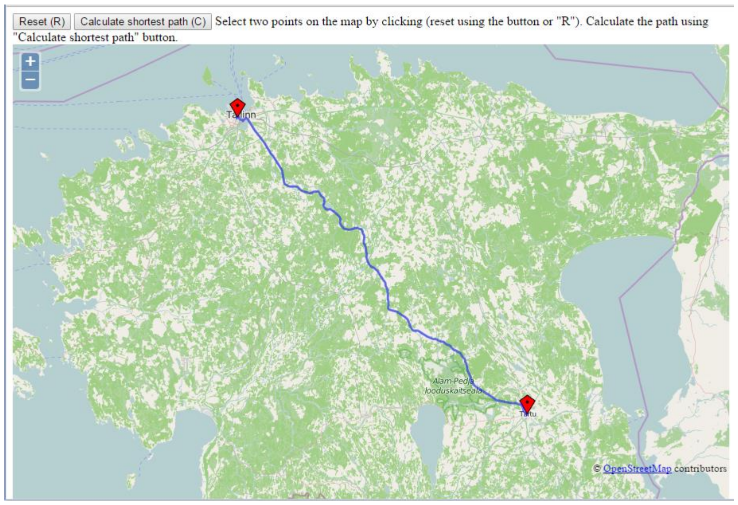
Client side was developed in Javascript using OpenLayers[9] library. OpenLayers is an open source JavaScript library for displaying map data in web browsers. It provides an API for building rich web-based geographic applications similar to Google Maps.

In the new program user can zoom and scroll around the map and choose two input points for shortest path calculation by clicking. By pressing "C" or the respective button above the map, a request is sent to the server to find a path between the chosen coordinates.

The server sends back a string containing a list of coordinates that make up the path, which is then displayed on the map.


Reset (R)
Calculate shortest path (C)
Select two points on the map by clicking (reset using the button or "R"). Calculate the path using "Calculate shortest path" button.
Tallinn
Alam-Pedja looduskaitseala
Tartu
© OpenStreetMap contributors

Picture 2: New UI


Reset (R)
Calculate shortest path (C)
Select two points on the map by clicking (reset using the button or "R"). Calculate the path using "Calculate shortest path" button.
Tallinn
Alam-Pedja looduskaitseala
Tartu
© OpenStreetMap contributors

Picture 3: Showing the result

D. Optimizations

Many optimizations were made to the program to increase its performance. One of them was using data structures better suited for the task.

For example, the priority queue in A* algorithm was previously implemented by simply using an array which was sorted in every iteration. Now it uses a binary heap instead.

In several places hashmaps/hashsets are now used which have the benefit of constant (O(1)) time complexity for data insertion/search. To give an example, there is the *closed set* in A* algorithm. Elements are added into and searched from the set. Previously it was implemented as a linked list (O(n) complexity for search) but it makes much more sense to have it as a hashset.

Previously DOM parser was used to read in XML data but now it uses SAX parser instead. The advantages of DOM parser are the ability to write into the file and random access but neither of those features are needed here.

SAX parser is a much better choice in this case because it doesn't load the entire file into memory. This makes it much more memory efficient, especially for larger XML files, because it instead uses stack while parsing the file.

DOM caused the program to use up a lot of memory during startup. Previously it needed around 1.5GB of memory to start up using map of Estonia. Most of the memory was used for XML parsing. With SAX the memory usage goes up to ~0.34GB and stays at ~0.33GB after startup.

After this change the program could be deployed on EC2 microinstance with map of Estonia, which was previously impossible since it has 1GB of memory available.

E. Road priority

As an experimental feature I implemented "road priority" system. The main goal for this was to make the application prefer larger roads when searching for a path. Nodes on a way are given a priority based on the type of way. For example, ways marked with *trunk* or *primary* are given a higher priority than *secondary* or *residential*.

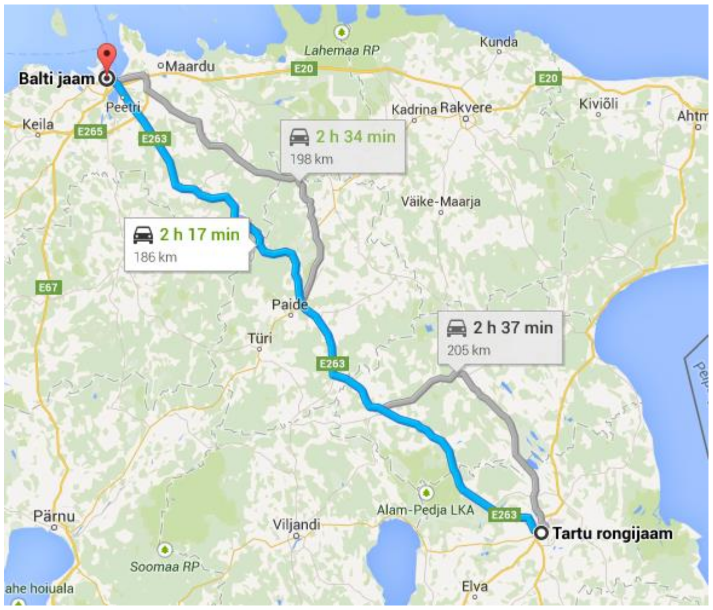
Their priority is taken into account when estimating the future path-cost in A*. In current implementation the estimated cost is divided by road priority. This means that higher priority roads get smaller estimates and since the algorithm chooses the node with the lowest future path-cost it will prefer larger road.

Also note that this can actually lower the performance of A* algorithm slightly but it seems to often produce a better output. The values for priorities are still being experimented with.

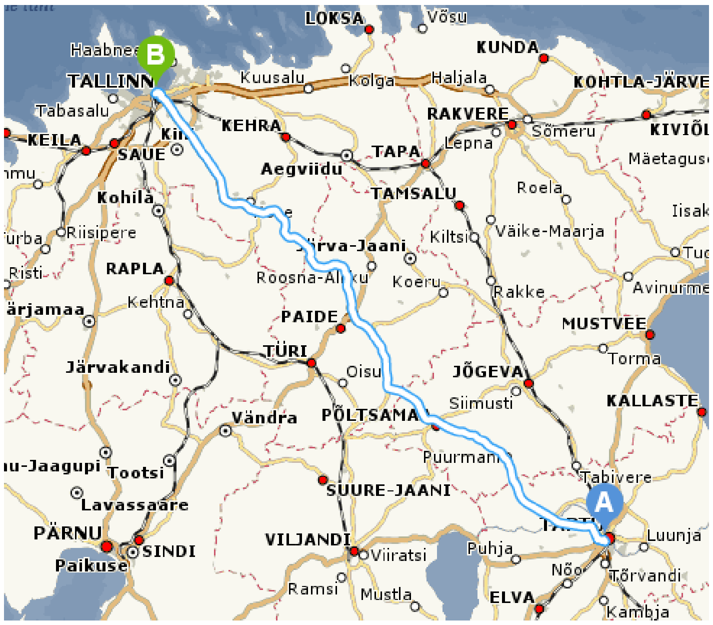
V. Results

As an example I tested finding a path from Tartu train station to Tallinn train station (Balti jaam). I compared the application to existing routing services.

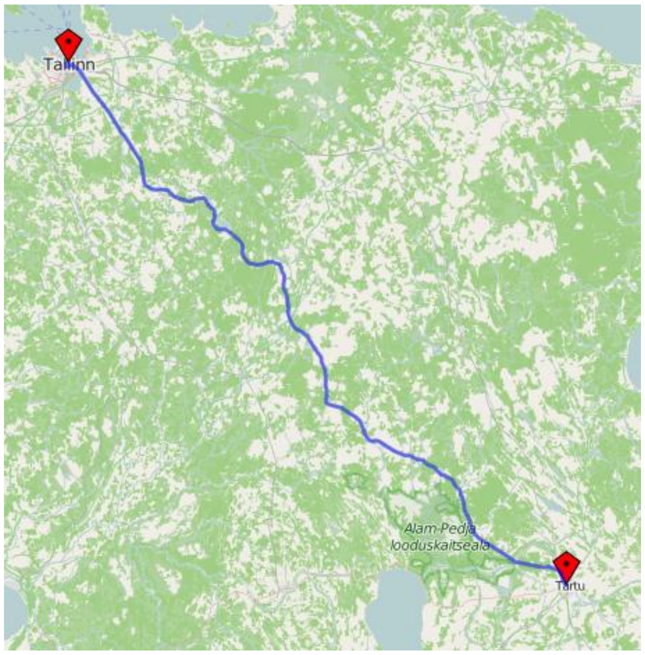
On picture 4 on the next page you can see the result of Google maps[10], on picture 5 the result of Delfi map[11]. Picture 6 is the output of the developed application. As you can see they are overall mostly the same but they have some differences inside cities.



Picture 4: Google maps result



Picture 5: Delfi map result



Picture 6: Application result

The shortest Google Maps route is 186 km. My route is 185.55 km. Path length is printed in browser console after the calculation.

VI. Possible future work

To further expand the program would probably require implementation of a more advanced routing technique such as landmark based path estimation or shortest path algorithm using mapreduce. Current solution is unlikely to scale much larger.

Summary

The aim of this project was to improve the previously created program. The new program is a web application with a more intuitive user interface. In addition to new look it has also undergone many important changes in core parts of the program.

Numerous optimizations were made that increase the performance of the program. These include using more efficient data structures and different XML parser.

A* algorithm is still used for finding the path. The heuristic used for A* algorithm is the distance between nodes, which is calculated using the Haversine formula. An experimental addition to the algorithm are the road priorities.

References

[1] Distributed Systems seminar: Shortest-path problem, Heiki Pärn
[2] https://bitbucket.org/heiki112/ds_shortestpath_v2
[3] http://52.28.71.17/ShortestPath/
[4] http://waprogramming.com/download.php?download=50af7709377c22.88356189.pdf
[5] http://rosettacode.org/wiki/Haversine_formula#Java
[6] http://en.wikipedia.org/wiki/A*_search_algorithm#Pseudocode
[7] http://wiki.openstreetmap.org/wiki/Osmosis
[8] http://wiki.openstreetmap.org/wiki/Key:highway#Roads
[9] http://openlayers.org/
[10] https://maps.google.com/
[11] http://kaart.delfi.ee/