OPTIMAL ROUTE SEARCH IN GEOGRAPHICAL INFORMATION SYSTEMS

Ramūnas Kraujutis
Tel. +37060064807, ramunas@kraujutis.lt,

Road networks are represented by a graph with crossroads as nodes and roads as edges. Therefore traditional algorithms for the shortest path search in graphs may be used, but their disadvantage is that they work with static and pre-processed data. However nowadays road information changes frequently, but data pre-processing can take long and full rebuild of graph may be required. Because of these new algorithms, data structures and fast update of pre-processed data are required for real time route search in frequently changing road networks.

**Keywords**: optimal route, graphs, shortest path, roads, road network, real time search

**Introduction**

Route is the list of vertices between two or more points in road network. In the modern world, route planning is used by many people planning their trips using GPS navigation systems. Fast route search algorithms are important for emergency medical service, police and other municipal services. It is very important to find optimal route using real-time traffic information in roads. Carriers could improve their logistics services, reducing the environmental pollution.

Route search in Geographical information Systems (GIS) differs from standard graphs search algorithms, because it must take into account the additional geographic information. Turning direction at the intersection may influence the optimum choice of route, since turning to the left is much longer-lasting event than turn to the right.

In this literature review algorithms for route search in road networks analysis and classification: traditional and non-traditional optimal route search methods.

The route with a minimum price is optimal route. In GIS, the price is typically distance or time. In some cases route can be minimized by distance / time ratio or the other metrics.

Road network, needed for search, consists of local roads, city streets and highways. Road network is visualized as graph, where edges are the roads and vertices are the crossroads and ending points.

Traditional shortest path algorithms, like Dijkstra, finds shortest path from initial s to end t vertices in graph $G = (V, E)$. However, the speed of these simple algorithms, in cases with many vertices and edges is not acceptable.

The following review provides several methods to shorten the search time in European-size road network, up to a few microseconds.

**Route search in static road network**

One of the most common tasks is to search the optimal route only with departure and arrival points. Route search used to plan trips by car, bicycle or public transport. Today there exists a number of route-planning web pages, handling huge quantities of search operations per day and doing it really fast.

Road network – is a graph, where crossroads and road end points are vertices and roads themselves are edges. Roads may have directions, so it is more convenient represent it as oriented graph $G = (V, E)$ having $n = |V|$ vertices and $m = |E|$ edges. Each edge $(u, v)$ must be nonnegative weight $w(u, v)$. The shortest path between starting $s$ and ending $t$ vertices is a minimal weight path $d(s, t) \geq 0$ from all paths which connects vertices $s$ and $t$.

In this section I will attempt to review the optimal route search algorithms applied for static road networks. In this case, for route price function distance, time or fuel consumption metrics can be used. Overviewed search methods work better if travel time and prices is correlated positively. The main task for search algorithms is to calculate optimal route price between departure and arrival points. To speedup search time, many search algorithms use preprocessed initial data.

**Bidirectional search**

Bidirectional search runs two simultaneous searches in a directed graph: one forward from the initial state and one backward from the goal, stopping when the two meet in the middle when the same vertex is visited from both sides. Visited vertices will be located as circles around initial and goal vertices. In traditional one direction search visited vertices are in one circle. In Figure 1 visited vertices by Dijkstra one direction and bidirectional algorithms are shown.
Bidirectional Dijkstra’s algorithms is terminated when two circles from initial and goal vertices met. One direction search stops when circle from initial point $s$ hits goal point $g$.

Bidirectional search allows to speedup search of the approximately two times.

**Use of heuristics**

Most modern navigational systems with limited processor and memory resources use heuristic in route calculation. Originally it was designed for A*. Heuristic algorithms may not include in search “smaller” streets (except when there are close to the target). Route networks data should be carefully prepared, all streets should be ranked. These and other techniques help to increase speedup in larger road networks.

**Road Hierarchy**

The first and practical route search algorithms were based on road hierarchy. Large road network graph $G = (V, E)$ is divided into small components $G_1, G_2, G_3, \ldots = G$ (also graphs). Each graph $G_1 = (V_1, E_1)$ consists from shortest paths in that graph $E_1$, and $V_1$ are vertices connected by $E_1$ edges. Route search can be performed using the newly formed graph $G_1$ and components with starting vertex $s$ and goal vertex $g$. Repeating the procedure, you can create graphs $G_2, G_3$ with fewer vertices in each level.

Preparation of hierarchies in large road networks takes lot of resources and time.

**Highway Hierarchies**

Usually travelling by car from a certain point in the city, we are trying to find nearest street. From simple city street we are trying to get into the city’s main street trying to reach town to the district-level roads. If a travelling distance is long enough, highways are desirable. Near the arrival point we are trying to choose, we are starting to choose the roads more at a lower hierarchical level.

Highway hierarchical search algorithm is based on a similar basis [11]. For the initial $s$ and goal $t$ vertices local area with $H$ closest vertices are defined. In each of these areas a vertex $(u, v) \in E$ exists that belongs to higher level of hierarchy. Search in initial and goal vertices are performed in local area until neighborhood limits are reached, later switching to highway network with much smaller than the complete graph and an optimal path includes.

Each higher level of hierarchy graph is optimized by removing the intermediate vertices and calculating the new edges. Iteratively repeating this process the highway hierarchy is constructed.

The optimal route search uses bidirectional search. Both Dijkstra, and A* [12] algorithms can be used.

Highway hierarchy – is a way to simplify road network and lets speedup search up to 8000 times comparing to traditional Dijkstra algorithm ([11] 43 p.).

**Distance tables**

To improve search performance distance tables can be used. For higher level graphs in graph hierarchy $G = (V, E)$ pre-calculated distances and paths between each vertex $v \in V$ can be stored in distance table:

<table>
<thead>
<tr>
<th></th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$\ldots$</th>
<th>$V_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>$d(V_1, V_1)$</td>
<td>$d(V_1, V_2)$</td>
<td>$\ldots$</td>
<td>$d(V_1, V_n)$</td>
</tr>
<tr>
<td>$V_2$</td>
<td>$d(V_2, V_1)$</td>
<td>$d(V_2, V_2)$</td>
<td>$\ldots$</td>
<td>$d(V_2, V_n)$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$V_n$</td>
<td>$d(V_n, V_1)$</td>
<td>$d(V_n, V_2)$</td>
<td>$d(V_n, V_n)$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Example of distance table:

<table>
<thead>
<tr>
<th></th>
<th>Amsterdam</th>
<th>Berlin</th>
<th>Copenhagen</th>
<th>London</th>
<th>Moscow</th>
<th>Rome</th>
<th>Warsaw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam</td>
<td>365</td>
<td>381</td>
<td>220</td>
<td>1325</td>
<td>808</td>
<td>673</td>
<td></td>
</tr>
<tr>
<td>Berlin</td>
<td>365</td>
<td>381</td>
<td>220</td>
<td>1325</td>
<td>808</td>
<td>673</td>
<td></td>
</tr>
<tr>
<td>Copenhagen</td>
<td>381</td>
<td>225</td>
<td>590</td>
<td>970</td>
<td>948</td>
<td>415</td>
<td></td>
</tr>
<tr>
<td>London</td>
<td>220</td>
<td>575</td>
<td>590</td>
<td>1540</td>
<td>890</td>
<td>890</td>
<td></td>
</tr>
<tr>
<td>Moscow</td>
<td>1325</td>
<td>995</td>
<td>270</td>
<td>1540</td>
<td>1462</td>
<td>710</td>
<td></td>
</tr>
<tr>
<td>Rome</td>
<td>808</td>
<td>730</td>
<td>948</td>
<td>890</td>
<td>1462</td>
<td>810</td>
<td></td>
</tr>
<tr>
<td>Warsaw</td>
<td>673</td>
<td>320</td>
<td>415</td>
<td>890</td>
<td>710</td>
<td>810</td>
<td></td>
</tr>
</tbody>
</table>

*Picture 2. Example of distance table [13]*

Hierarchical search algorithms can use distance tables, when it reaches level with pre-calculated distance tables. Usage of pre-calculated distance tables allows to speedup search up to 2 times [14]. Pre-calculation of distances requires additional resources of processor and memory.

Transit nodes

Transit nodes algorithm is based on the simple idea [15] [16] [17]. When planning long trips and analyzing the map around departure point several “important” leaving points (roads/crossroads) are chosen. These points are called access nodes and marked $\tilde{A}(s)$. Around the arrival point every access nodes are marked as $T \in V$ and called transit nodes.

This method pre-computes not only a distance table for important (transit) nodes but also all relevant connections between the remaining nodes and the transit nodes. According that transit nodes are chose in both departure and arrival points, route search can be executed only in transit nodes graph. To improve performance of transit nodes route search, distance tables can be used. About 10 000 transit nodes can be used in Europe-size road network [15].

Optimal route price from start vertex $s$ and goal vertex $t$ passing through $u$ and $v$ transit nodes is marked:

$$ d^*(s, t) = \min_{u, v \in \tilde{A}(s), v \in \tilde{A}(t)} (d(s, u) + d(u, v) + d(v, t)) $$

It departure and arrival point are close, transit node algorithm shouldn’t be used. So it is important to use locality filter $L : V \times V \rightarrow \{true, false\}$ that decides whether source and target are too close so that we need a special treatment to guarantee the correct result.

This algorithm allows to speedup search up to 1400000 times comparing with standard Dijkstra algorithm ([15] 13 p.). Interestingly, the most difficult queries are now the local ones where the shortest path does not touch any transit node.

Route search in with real time data

Today, situation in roads changes very frequently, so it is very important use most recent data in optimal path calculation. Continuously changing information about traffic jams impacts results of price function. Over the next few years, a lot of road sensors will be installed, so information flow about traffic situation will increase significantly. Shortest path algorithms should take into account data obtained from sensors integrated into the cars.

Many of above mentioned shortest path algorithms use long-lasting data preparation. These algorithms can’t be used for real time shortest path search, because after data preparation situation in roads may change significantly.
Current search algorithms should be modified in order to avoid outdated search results. Preprocessed data should be updated after every update of road network. However, data structures used for prepared data, does not allow quickly involve changed data. Often any change to road network requires a complete rebuild of prepared data.

Static data should be separated from frequently changing data to avoid outdated search results and long-lasting full rebuild of road network. Route search function should be modified to take into account information about dynamic data.

During rush hours average speed in cities are much lower than usual. In order to improve the accuracy of the shortest path, historical statistical data should be evaluated. In this case when departure time is known, price function should be depending on time. Almost all search algorithms used bidirectional search, in this case, arrival time is not known. Below the

Picture 5 shows the areas that can be accessed by car within 5, 10 and 15 minutes.
Areas that can be accessed by car within 5, 10 and 15 minutes

Approximate arrival time may be used with bidirectional search.
Route search scheme based on historical statistics:

**Dynamic route search**

- Road network
- Dynamic data
  - Dynamic, constantly changing data is taken into account
  - Statistical data of road load
- Prepared data
- Route search algorithm
- Historical data

**Picture 5.** The areas that can be accessed by car within 5, 10 and 15 minutes

**Conclusions and Future Work**

In this paper, an overview of existing traditional route search algorithms. Traditional Dijkstra and A* algorithms are most used and widely adopted, but these algorithms can’t be used practically in large road networks due long execution time. For large road networks improved algorithms are used (mentioned in section: Route search in static road network). These algorithms are very efficient with non-changing road network. They may find route between any two vertices in Europe in few microseconds. With such rapid algorithms route visualization takes more time than search itself.

Route search algorithms of **static** road network are very fast, but much more realistic scenario is with **dynamic** data changing over time.

Material provided in this work could be used in creation process of routing algorithm of frequently changing road network.

One of existing search algorithm can be chosen, used data structures fully analyzed and the methods of how to efficiently update prepared data by avoiding the full rebuild.
References

22. Krylovas. Diskrečioji matematika. VGTU 2009