Review paper on Traffic Route Scheduling in Road Networks

Sanvir Singh¹, Ramanpreet Kaur²

¹Research Scholar, M.Tech(CSE), ²Assistant Professor (CSE),
¹²CGC Technical Campus, Jhanjeri.

Abstract: Road traffic jam difficulty is no single day or once in a week problem; it will be possible in daily life. So, it’s suggested or computing to choose a best optical path from source to destination in best possible way. Many people frequently deal with this question when planning trips with their cars. There are also many applications like logistic planning or traffic simulation that need to solve a huge number of such route queries.

I. INTRODUCTION:

Current commercial solutions usually are slow or inaccurate. [1] The gathering of map data is already well advanced and the available road networks get very big, covering many millions of road junctions.[6] Thus, on the one hand, using simple-minded approaches yields very slow or expensive for the service provider if he has to make a lot of computing power available. On the other hand, using antagonistic heuristics yields inaccurate results. For the client, this can mean a waste of time and money. For the service provider, the developing process becomes a difficult balancing act between speed and sub optimality of the computed routes. Due to these reasons, there is a considerable interest in the development of more efficient and accurate route planning techniques.

A. The Shortest-Path Problem

A road network can easily be represented as a graph, i.e., as a collection of nodes V (junctions) and edges E (road segments) where each edge connects two nodes. Each edge is assigned a weight, e.g. the length of the road or an estimation of the time needed to travel along the road. In graph theory, the computation of shortest paths between two nodes is a classical problem. Actually, we can distinguish between several variants of this problem:

- **Point-to-point**: compute the shortest-path length from a given source node s ∈ V to a given target node t ∈ V;
- **Single-source**: for a given source node s ∈ V, compute the shortest-path lengths to all nodes v ∈ V;
- **Many-to-many**: for given node sets S, T ⊆ V, compute the shortest-path length for each node pair (s, t) ∈ S × T;
- **All-pairs**: a special case of the many-to-many variant with S := T := V.

B. Speedup Techniques

There are several requirements that such a speedup technique should ideally fulfill:

- The query times should be as fast as possible.
- The result should be accurate, i.e., a provably optimal path should be computed.
- The method should be scale-invariant, i.e., it should be optimized not only for long paths. In other words, the running time of the computation w.r.t. the available data of a shortest path in a large graph should be not much higher than the running time of the same computation in a smaller graph.

- If the approach uses some preprocessing, it should be sufficiently fast so that we can deal with very large road networks.
- Precomputed auxiliary data should occupy only a moderate amount of space.
- Updating some edge weights (e.g., due to a traffic jam) or replacing the entire cost function should be supported.

II. LITERATURE SURVEY

We introduce basic data structures, algorithms with some notations and this notation will be used in this thesis. All the fundamental part will be covered from Graphs, but that will be used in some different directions. [2] We have already studied that G=V,E, where G represents the Graphs, V is vertex and E is an edges. We expect a direct graph tt = (V, E) with a node set V of size n and an edge set E ⊆ V × V of size m as input. A weight function w: E → R+ assigns a nonnegative weight w((u, v)) to each edge (u, v). We usually just write w(u, v) instead of w((u, v)). In road networks node generally used junctions and edges represented as road segments. As already discussed in the starting that Dijkstra’s algorithm is used to **Single-Source Shortest-Path Problem**. Dijkstra’s algorithm [2] can be used to solve the single-source shortest-path (SSSP) problem, i.e., to compute the shortest paths from a single source node s to all other nodes in a given graph. Starting with the source node s as root, Dijkstra’s algorithm grows a shortest-path tree that contains shortest paths from s to all other nodes. Let us fix any rule that decides which element Dijkstra’s algorithm removes from the priority queue in the case that there is more than one queued element with the smallest key. Then, during a Dijkstra search from a given node u, all nodes are settled in a fixed order. The Dijkstra rank rk(u) of a node v is the rank of v w.r.t. this order. u has Dijkstra rank rk(u) = 0, the closest neighbour v₁ of u has Dijkstra rank rk₂(v₁) = 1, and so on.

III. PROBLEM FORMULATION

Let us consider the following naive route planning method:
1. Look for the next sensible motorway.
2. Drive on motorways to a location close to the target.
3. Leave the motorway and search the target starting from the motorway exit.

Of course, it is true that this fast method does not always yield the optimal solution, but, in many cases, we obtain a reasonable approximation (pro-vided that source and target are not too close together and that we travel in a country whose motorway network is well developed).[4] This naive route planning method is based on a simple rule of thumb: when we are on our way to a remote target and pass by a city on a motorway, it usually does not pay to leave the motorway and look for a faster way through the city; in other words, usually, we can safely ignore all ‘less important’ city streets and stick to the ‘more important’ motorway since we know that the motorway provides the fastest way. [3] The approach that is used by some commercial route planning systems is based on the above idea:

1. Search from the source and target node (‘bidirectional’) within a certain radius (e.g. 20 km),
   consider all roads.
2. Continue the search within a larger radius (e.g. 100 km),
   consider only national roads and motorways.
3. Continue the search only in motorways.

Freeway Pecking order or Highway hierarchies are the first route planning technique that was able handle the road network of a whole continent, achieving speedups of more than a factor 1 000 compared to Dijkstra’s algorithm. They offer a good compromise between preprocessing time, memory consumption, and query time. In particular w.r.t. preprocessing time, they are superior to practically any other method that achieves significant speedups.

A priority queue Q manages a set of elements with associated totally or- dered priorities and supports the following operations:

- insert – insert an element into the priority queue,
- deleteMin – retrieve the element with the smallest priority and remove it from the priority queue,
- decreaseKey – set the priority of an element that already belongs to the priority queue to a new value that is less than the old value.[6]

IV. OBJECTIVES

There are two main objectives:

- The computation should be as speedy as possible and the resulting set should be as small as possible.
- Our objective to obtain an exact algorithm requires the definitions of highway network: An edge \((u, v) \in E\) belongs to the highway network if there are nodes s and t such that \((u, v)\) is on some shortest path from s to t and not entirely within the neighborhood of s or t.

V. METHODOLOGY OF PROPOSED WORK

Highway-node routing is our method of choice when considering a mobile scenario (e.g., a car navigation system). In this case, a concrete realization can take advantage of the conceptual simplicity and the low memory requirements. When dealing with point-to-point queries in a server environment (e.g., route planning systems that provide their services in the internet), transit-node routing can provide excellent response times as long as we consider a static scenario. The query times of highway-node. In particular, we have started to determine better highway-node sets, to parallelize the preprocessing of highway-node routing, and to write an implementation of highway-node routing for a mobile device.

Present Traffic Situations. One challenge is to deal with a massive amount of updates to the cost function. These updates reflect the present traffic situation, in particular unexpected events like traffic jams and their effects on the surrounding area. Here we will introduce existing methods like highway-node routing can cope only with a moderate amount of changes.

Upcoming Traffic Situations. Another challenge is to incorporate predictions for upcoming traffic conditions. Such prediction is based on statistical data and is expressed by time-dependent cost functions, which can project, for example, a slower average speed during the morning rush hour. A direct application of existing approaches would fail since a time-expanded representation of a large road network would exceed the available memory. Furthermore, in a time-dependent scenario, all bidirectional search techniques face the problem that simultaneously performing a forward and a backward search normally requires the knowledge of both the exact departure and the exact arrival time.

VI. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

- Processor: Pentium IV or Higher
- RAM: 4 GB or Higher
- Hard Disk: 100 GB or Higher:

Software Requirements:

- Platform: Linux/Ubuntu

REFERENCES