MARiO: Multi Attribute Routing in Open Street Map

Franz Graf Matthias Renz Hans-Peter Kriegel Matthias Schubert

Institute for Informatics, Ludwig-Maximilians-Universität München, Oettingenstr. 67, D-80538 Munich, Germany
{kriegel, schubert, zuefle}@dbs.ifi.lmu.de

Abstract. In recent years, the Open Street Map (OSM) project collected a large repository of spatial network data containing a rich variety of information about traffic lights, road types, points of interest etc.. Formally, this network can be described as a multi-attribute graph, i.e. a graph considering multiple attributes when describing the traversal of an edge. In this demo, we present our framework for Multi Attribute Routing in Open Street Map (MARiO). MARiO includes methods for preprocessing OSM data by deriving attribute information and integrating additional data from external sources. There are several routing algorithms already available and additional methods can be easily added by using a plugin mechanism. Since routing in a multi-attribute environment often results in large sets of potentially interesting routes, our graphical fronted allows various views to interactively explore query results.

1 Introduction

The Open Street Map (OSM)[1] project collects rich and up-to-date information about road networks and the landscape surrounding them. Combining this information with other publicly available information about the spatial landscape allows to derive a large variety of information that previously has not been considered in routing systems. For example, a network might contain information about the distance, the speed limit, the altitude difference or the number of traffic lights for each road segment. Thus, a driver looking for the route which fits best to his personal preferences might want to consider various cost criteria at the same time. When employing ordinary shortest path routing, multiple attributes can be integrated by selecting a preference function combining cost criteria. For example, a user might enter that his major preference is driving the fastest path with a weight of 80%, but still wants to consider driving distance with a minor weight of 20%. By still considering travel distance with a minor weight, the selected route might be considerable shorter and only slightly slower than the fastest path. Thus, the gas consumption and the risk of getting into a congestion should be considerably smaller. However, finding an appropriate weighting is not intuitive and thus, better solutions should be found.

To conclude, considering multiple attributes has the potential to improve the usability of routing but raises a lot of further research questions requiring new problem specifications and solutions. First works in the area where proposed in [1] and [2]. While [1]

[1] http://www.openstreetmap.org

ranks possible destinations w.r.t. to multiple cost attributes, [2] introduced route skyline queries. The result of a route skyline query consists of all routes connecting one starting point and one destination having an optimal cost value w.r.t. any linear combination of cost values.

In our demonstration, we want to present our framework for Multi-Attribute-Routing in OSM data (MARiO). MARiO is an open source project combining functionalities for data integration and preprocessing, implementation of new algorithms and performance evaluation. Our graphical frontend provides methods for posing queries and interactively exploring result routes. Since there is a number of queries computing multiple result routes, handling a result set of potentially hundreds of sroutes requires sophisticated tools. Thus, we integrated various interconnected views on the potentially multidimensional cost space and perform post processing in the form of clustering result routes.

The rest of this paper is organized as follows. In section 2, we provide an overview of the framework and its functionalities. Section 3 describes the already implemented algorithms. Afterwards, we sketch the content of the demonstration 4. Section 5 briefly summarizes the demonstrated system features.

2 System Overview and Functionalities

In this section, we want to give an overview of the functionalities of MARiO. We implemented our framework in Java 1.6 to be independent from a particular hardware platform.

A first functionality is importing map data from OSM. In order to apply multi-attribute routing, we cannot rely on the rich map representation provided by OSM. First of all, the OSM format contains a lot of unnecessary information for route computation. A second more important reason is that several of the employed optimization criteria are not directly maintained in the maps. For example, we have information about traffic lights and altitudes connected to the nodes which have to be reassigned and post processed into edge attributes of a multi-attribute graph. Furthermore, there is publicly available data from other sources than OSM that provide further useful information. Therefore, we allow to add topographic data from the SRTM[2] program. Another reason making preprocessing of the map information advisable is that available maps often contain a lot of nodes which are not required for routing purposes, e.g. nodes that are integrated to display turns in an edge. In order to allow efficient path computation, deleting these nodes and combining the neighboring edges can significantly reduce the number of considered routes.

After loading network data into an internal adjacency list representation, it is possible that additionally preprocessing steps are required. An important functionality for many routing algorithms, e.g. A*-Search, is to compute an approximation for the minimal cost of a path between two nodes. A common approximation for the shortest path w.r.t. network distance is the Euclidian distance between the spatial coordinates of both nodes. However, the same idea is not applicable for general attributes. For example, the

[2] http://www2.jpl.nasa.gov/srtm/

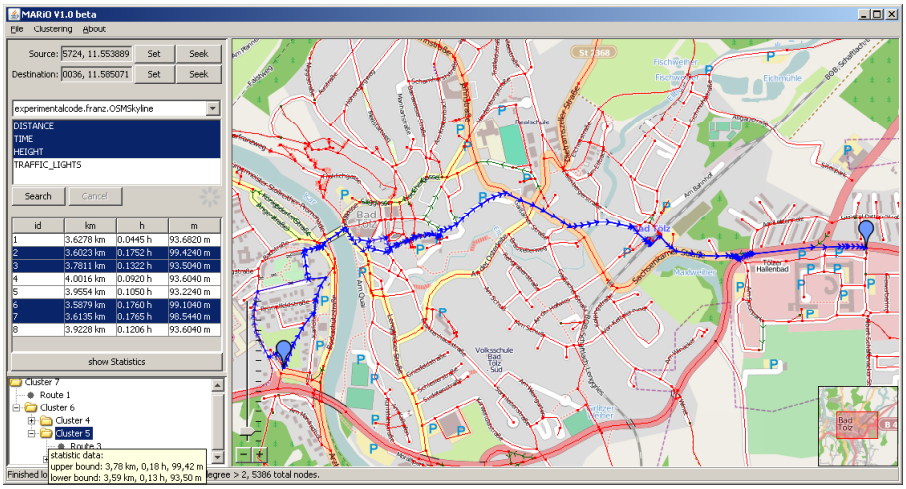

Fig. 1. Screen shot of the MARiO Frontend.

number of traffic lights on a route cannot be estimated based on distance. Therefore, we implemented a reference node embedding storing at each node the distance to each of a well selected set of reference nodes. The advantage of this approach is that it is viable to arbitrary positive edge attributes. The drawback of the approach is the large memory consumption because it is necessary to store a distance value for each node, each reference node and each attribute type. To significantly lower the memory consumption of this method, we implemented a sparse variant of the embedding being proposed in [3].

To integrate various query types and compare algorithms solving the same problem, we designed our framework in a way allowing the fast and flexible integration of new algorithms. Therefore, new algorithms are integrated by employing a plugin mechanism. As a result, it is possible to add additional query types or algorithms without altering the original code of the framework. After adding the algorithm the framework lists the algorithms in the frontend and automatically generates a dialog to select parameter values. The result is expected to be a list of result routes which can be displayed in user interface. A further generalized feature of the framework is the possibility to analyze the performance of the algorithms. Therefore, is is possible to monitor and report general performance measures for route planning algorithms like query time, result size, the number of accessed network nodes or the number of extended routes.

A final component of the MARiO framework is its frontend which is displayed in figure 1. The frontend allows to display the OSM map data by using the map view component of SwingX-WS[3] which contains versatile viewing controls. Furthermore, the frontend allows the user to pose queries using various algorithms and provides multiple methods for displaying the result set. The first view on the result set consists of a grid control containing the cost w.r.t. each of the selected cost attributes. There exists further views visualizing the cost values for the case of two and three attributes. To handle the

[3] http://swinglabs.org

particularly large number result routes that sometimes occur in multi-attribute routing, we can display the result in the form of a clustering tree. The clustering is derived by single link clustering which is based on a weighted variant of Hemming distance. Thus, the result is clustered w.r.t. the visited nodes instead of the cost attributes. The resulting clustering can be seen in the lower left corner of figure 1.

3 Implemented Algorithms

In the previous section, we described the general functionalities that can be used when implementing and testing a routing algorithm. In this section, we shortly review the already available algorithms. For basic shortest path computation based on a single cost attribute, the framework implements Dijkstra's algorithm and A*-search. The A*-search is based on the reference point embedding named above.

A second type of query being already implemented is a route skyline query. To calculate the route skyline for a given set of quality criteria, we employ the ARSC algorithm described in [2]. The basic idea of this algorithm is a best first traversal of the graph beginning with the starting position. During query procession the algorithm maintains two data structures. The first is a priority queue containing all nodes that still must be visited to find all skyline paths. The second structure consists of a table storing the already encountered pareto-optimal sub-routes for each visited node. Due to the monotonicity of local sub-routes, it can be shown that each sub route of a skyline route ending at the destination must be a skyline route between the starting location and its ending location. Thus, extending any path which is not part of the local skyline of its ending location cannot lead to a skyline route to the destination. To further speed up skyline computation, we additionally compare the lower bound approximation for any path to the current skyline of paths of the destination. If the lower bound approximation is already dominated by a member of the current skyline of the destination, the path can be pruned as well. The algorithms terminates when there is no path left that could be extended into a member of the route skyline to the destination node. For a more detailed description of the algorithm please refer to [2].

4 Demonstration

To demonstrate the functionalities of the MARiO framework, we will focus on query processing and result browsing in the frontend.

To pose a query, the user has to select an available query algorithm. Depending on this selection, the system can now generate a query dialog requesting the required input parameters from the users. For example, a route skyline query being processed by the ARSC algorithm requires a set of cost attributes, a starting point and a destination. The cost attributes are selected as a subset of the attributes being supported by the currently loaded graph. To select spatial locations the system allows to mark the coordinates directly on the map view. As an alternative, MARiO supports an address search to pinpoint locations. After parameter selection, the search is being started and the system collects the statistical information about query times, visited nodes and extended routes.

The result is a set of routes in the network which are characterized by a trajectory and a cost vector describing the cost of each of the selected attribute types. A basic view of this result set is a grid control containing a row for each result route and a column for each type of selected cost attribute. When clicking one or several routes in the control the corresponding route is marked in the map view. Furthermore, it is possible to sort the result set by any type of selected cost in the result set. A further view on the result data that is being made available for two attributes is a 2D vector view. For the route skyline query, this view always displays the well-known step function of a skyline. For 3D data, there exists a further view displaying the result set in simplex.

A final feature being extremely useful for rather large result sets is to view the result routes by browsing its cluster tree. The tree is displayed in an tree control and thus, a user can navigate deeper into the cluster by expanding the notes. To get an impression of the contents of a cluster, it is possible to select a node in the tree and simultaneously display all contained routes in the map view. Furthermore, the tool tip of the node displays upper and lower bounds for each cost value of the clustered routes. For example, a cluster might be described by 4 routes having a travel time between 0.25 and 0.5 hours and a distance between 10 and 12 km. By clustering result routes w.r.t. the visited nodes in the graph, the routes within a cluster do not have to minimize the displayed intervals. However, the clusters display similar trajectories on the map view. Thus, top-level clusters distinguish rather general areas a trajectory is visiting while low-level clusters rather represent local variations. Thus, examining the top level can be employed to investigate general directions and by traversing the tree the user can stepwise decide which route fits best to her particular preferences.

5 Conclusion

In this proposal, we introduced MARiO a framework for Multi-Attribute Routing in OSM data. Our framework, has three main functionalities. The first is data integration and preprocessing in order to construct multi-attribute graphs from OSM data. The second is the simple implementation and integration of new algorithm via a plugin mechanism. Finally, we provide a frontend for posing queries and exploring query results. Since the result set being generated by a multi-attribute routing algorithm can be rather large, there exists several interconnected views displaying result routes on the map, in the cost space or summarize the result with a clustering algorithms.

References

1. Mouratidis, K., Lin, Y., Yiu, M.: Preference queries in large multi-cost transportation networks. In: Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach,CA,USA. (2010) 533–544
2. Kriegel, H.P., Schubert, M., Renz, M.: Route skyline queries: A multi-preference path planning approach. In: Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach,CA,USA. (2010)
3. Graf, F., Kriegel, H.P., Renz, M., Schubert, M.: Memory-efficient a*-search using sparse embeddings. In: Proc. ACM 17th International Workshop on Advances in Geographic Information Systems (ACM GIS), San Jose, CA,US. (2010)