Flexible Route Guidance Through Turn Instruction Graphs

[Work-In-Progress]

Dennis Luxen[*]
MapBox Inc.
1714 14th Street NW
Washington, DC, USA
dennis@mapbox.com

ABSTRACT

This paper provides a detailed description of work in progress on the algorithmic challenges of generating *textual* route guidance. More specifically, we show how to transform the generation of meaningful turn instructions into a path search problem on a graph. We develop a thorough algorithmic framework with well-defined data structures. Subsequently, we show how to use this algorithmic framework in a way that gives flexible route guidance reflecting either user preferences of personal local knowledge. The structural properties of the graph allow us to generate guidance along the route in time linear in the number of junctions on the computed routes in theory and within milliseconds in practice. As the results are preliminary some of the explanations are given by example.

Categories and Subject Descriptors

H.3.5 [**Information Systems Applications**]: Web-based services; G.2.2 [**Mathematics of Computing**]: Graph Theory—*Graph algorithms*

General Terms

Algorithms, Performance, Experimentation

Keywords

Route Descriptions, Textual Route Guidance Construction, Search Graph Modelling

1. INTRODUCTION

Map services offering route planning have become ubiquitous on the Internet. Virtually every major service provides its users with some sort of online maps and a routing functionality, e.g. the services of Google [14] or Bing [6] Maps. Additionally, route planning services have become more flexible over the course of time by providing users with additional routing algorithms, e.g alternative routes, or time-dependency to account for regular rush hour congestion.

The problem of routing in graphs has been solved (in theory) with Dijkstra's seminal algorithm since 1959 [9]. Unfortunately, it does not scale well in practice and it has been a major challenge to design optimal algorithms with real-time guarantees. Even a well-tuned implementation in C++ takes several seconds for a road network of continental size, e.g. the one of Northern America or EurAsia. Note that this is not practical in a setting where (literally) thousands of concurrent users expect instant answers.

To draw an outline, we do not actually focus on routing algorithms in this work, but we will give a brief recap for the sake of completeness. First approaches added a sense of goal-direction [15] to guide the search into the *right* direction. The route planning problem has seen a lot of results in recent years from the algorithm engineering community. These results give practical routing algorithms in real-world road networks of continental size with millions of nodes and edges. Some of these algorithms directly exploit the hierarchical properties, induced by metric and topology, inherent to real-world road networks, e.g. [12]. The algorithms achieve query times in the order of a single millisecond in practice. Even more recently, Abraham et al. [1] give further theoretical insights, which lead to the fastest known distance oracle [2] with sub-microsecond query times for queries on large road networks. Likewise, Arz et al. [4] gave a very intuitive distance oracle with single-digit microsecond query times that can be constructed in a few minutes on a off-the-shelves desktop computer.

For a recent overview on route planning techniques in static graphs, see the survey of Sommer [23].

1.1 More Related Work

It is fair to say that static route planning has been solved on road networks from an algorithm engineering point of view. As a result of this ground-breaking work more broader goals have been formulated for routing, e.g. the computation of alternative routes [18] or time-dependency [5].

Route guidance generation has seen some research in the past, too, but actual algorithmic results are scarce. None of the well-known online services publishes their methods. For example, Duckham et al. [10] suggest to include landmarks that are passed by into the route guidance. Westphal and

[*]This work was partially funded by DFG Grant 933/2, when the author worked at Karlsruhe Institute of Technology, Karlsruhe, Germany.

MapInteract'13, November 05 2013 Orlando, Florida USA
Copyright 2013 ACM 978-1-4503-2536-3/13/11 ...$15.00.
http://dx.doi.org/10.1145/2534931.2534938.

Renz [25] provide a graph transformation that puts a penalty on routing over complex junctions similar to our modeling in Section 2.

A related topic to route guidance is the generation of route sketches. The full geometry of a route is schematized such that it is generalized but preserves its major features. Public transportation maps [20] are a prominent example for these schematizations. Brandes and Pampel [7] show that is actually $\mathcal{NP}$-hard to construct an optimal sketch. Fortunately, it is possible to efficiently produce good and visually appealing results in practice, e.g. as the approximation algorithms of Delling et al. [8] shows. It draws a sketch with size proportional to route complexity while Agrawala and Stolte [3] give an algorithm based on simulated annealing . Schmid et al. [21] turn to the generation of *sparse maps* that do not clutter the presentation but contain sufficient information to let a traveller successfully reach a destination. Similarly, Kopf et al. [17] generate *destination maps* that show a sparse subset of all map data, i.e. the most important roads only. Somewhat related, Imai and Iri [16] arguably use shortcuts in a directed acyclic graph to simplify line geometries. We will draw from some of these previous results, most notably by formulating our approach as an abstract search problem.

The remainder of this paper is structured as follows. Section 2 gives a brief overview on the modeling and methodology of route planning. Subsequently, Section 3 explains our approach and the modeling of the problem as a graph search. Section 4 then gives an approach to generate flexible route guidance by exploiting the properties of the graph search. Section 5 explains how to deal with name ambiguity. Last but not least, Section 6 discusses preliminary results achieved so far, while Section 7 draws a conclusion and identifies future work.

2. BASIC FRAMEWORK

We give a brief overview on the general setup of our use of routing in terms of a basic frame work and identify corresponding data structures [19]. Note that the actual method of finding shortest paths in a graph is viewed as the application of a black box. Nevertheless, this basic modeling is also the basis of our work.

The road network is modeled as a directed graph $G = (V, E)$ where V is a set of nodes (or vertices), i.e the set of junctions, and $E \subseteq V \times V$ is a set of edges, i.e. the set of road segments between junctions. Each edge $e \in E$ has a weight $w(e) \in \{\mathbb{R}_+ \cup \infty\}$ that reflects the cost to traverse that edge, e.g. the euclidean distance between start and end vertex or the travel time needed to traverse the edge. A turn can be specified by giving adjacent edge segments $\langle e_1, e_2 \rangle$, with $e_1, e_2 \in E$ or by specifying the corresponding node triple $\langle u, v, w \rangle$ with $e_1 = (u, v)$ and $e_2 = (v, w)$. Turn costs or penalties associated with a given turn $\langle u, v, w \rangle$ are denoted by $tc(u, v, w)$. Assuming that traffic originates and ends at vertices only, a shortest (or quickest) path can be computed easily by Dijkstra's seminal algorithm [9] on this so-called *node-based graph*. The geometry of the road network is modeled by vertices of degree 2 that approximate curves by strings of short linear segments.

Like [17], we view intersections as critical *decision points*. It is at these junctions along the route that guidance must give the corresponding action to take, e.g. to advise a turn and to indicate when to expect the next instructions

2.1 Our Graph Model

A node-based graph resembles the basic topology of a road network. Unfortunately, it does not account for turn restrictions or general turn costs as-is. Consider the example of Figure 1. It shows two exemplary turns a and b in right-

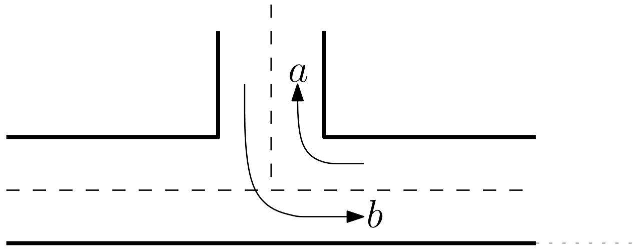

Figure 1: Turn maneuvers are associated with non-uniform costs.

hand traffic. Obviously, one would expect turn `a` to be *easier* to perform than turn `b` in reality as the latter one has to potentially cross opposing traffic. Or perhaps the turn may even be forbidden at all. Thus, it may make sense to propose an entirely different route or at least we would like to account the *turn costs* of such expensive maneuvers. The graph does not contain this turn information as is and it must be given separately. Note that the argument is symmetric for left-hand traffic.

Adding these costs to a routing algorithm in node-based graphs is feasible but it leads to a label-correcting variant of Dijkstra's algorithm where nodes may be reopened after being settled. Unfortunately, this variant is particularly difficult to analyze. This happens, for example, when a left turn (u, v, w) is forbidden and traffic has to do a loop of right turns to enter segment (v, w), i.e. to find a path that contains a turn (u', v, w), with $u \neq u'$. One way to account for this inconvenience, and to return to a label-setting variant of Dijkstra's algorithm, is to *expand* the graph in a way such that it explicitly encodes all possible turns in its topology.

This *pseudo-dual* of the node-based graph is called edge-based or *edge-expanded graph* G^e [26, 24]. We will stick to the latter name. Edge-expanded graphs are constructed as follows.

Given are a node-based graph G and a list of turn restrictions and a list (or function) of turn costs. Each possible turn (u, v, w) in the node-based graph G is represented by an edge (a, b) in the edge-expanded graph G^e. Likewise nodes $a := (u, v)$ and $b := (v, w)$ of G^e resemble source and target segment of a turn (u, v, w). The costs associated with an edge-expanded edge $w(a, b) := w(u, v) + tc(u, v, w)$ is the sum of the incoming (node-based) edge weight plus the costs associated with the turn. Note that turn restrictions can be handled setting $tc(\cdot) := \infty$.

The actual construction is straight-forward and essentially an enumeration of the input graph. First, each directed input edge is transformed into an edge-expanded node. Second, all turns in the input graph are enumerated and a corresponding new edge with appropriate costs is inserted into the edge-expanded graph if (and only if) the turn is not forbidden. Turn restriction are explicitly modeled by omitting the corresponding (edge-expanded) edges from G^e. Whereas the node-based graph consists of roads between junctions, the edge-expanded one models turns between roads.

In practice the size of the graph data structure increases by about a factor of 2–3. But now a simple label-setting variant of Dijkstra's algorithm suffices to compute paths respecting turn costs and restrictions. Note that it is easy to reconstruct the node-based path from the edge-expanded representation by storing the middle node v with each edge-expanded edge. Further details can be found in the experimental study of Volker [24]. Note that the extra book-keeping of Geisberger and Vetter [13] on routing in a node-based graph with turn costs is a rather complicated on-the-fly simulation of routing in the edge-expanded pseudo-dual.

The explicit turn modeling found in the edge expanded graph has further advantages. It is easy to store additional data along side each edge that encodes further information on the turn. In our implementation [19], we also encode basic turn instructions with each turn as if every turn in our graph leads over a decision point. We call each of these pieces of guidance a *micro instruction*. Based on analyzing the geometry of turn $\langle u, v, w\rangle$ we note the direction of the local turn during graph construction. Also, we note certain turn maneuvers like entering or leaving roundabouts, the highway or access restricted areas. Each of these basic instruction is represented by a distinct integer, i.e. we need $O(\log k)$ bits to represent a total of k distinct instructions. This does not pose a real space penalty under the assumption that we only need to consider a reasonable number of distinct micro instructions.

3. GUIDANCE CONSTRUCTION

Now that we have explained the basic outline of our setting we turn to the construction of *reasonable* route guidance. As we argued above, the routing component is a black box that returns a route as a string of adjacent nodes (or edges). It is traversed in the order induced by the shortest path. From this result we have to generate the route guidance. We retrieve the full string of micro instructions for the entire route by essentially following the parent pointers of the nodes of the computed path.

While we have a over-detailed string of micro instructions this list is arguably not of great value as guidance to any user as it contains an over-abundance of instructions that are valid only for small pieces of the route only, e.g. a couple of dozen of instructions repeatedly telling to keep straight for the next few meters. Obviously, there is a need for a more reasonable and more compact representation of the information. Metaphorically speaking, we need to boil this string down into a meaningful gravy. Figure 2 gives an illustration of how the string of micro instructions can to be condensed to yield a meaningful (textutal) route guidance.

A *turn instruction graph (TIG)* is derived from a computed route on an edge-expanded graph. Its node set corresponds to the road segments of the route while its edges correspond to the turn instructions. Initially, each edge has unit weight. Note that at first a TIG is essentially the computed route. We allow two basic operations to this graph. The first operation is *node contraction*, which removes a node from the graph and replaces its adjacent edges by a so-called shortcut that aggregates the information of the adjacent edges. Consider the following examples. We may shortcut any turn followed by a straight-ahead into a single instruction.

Example 1. *The string of instructions*
1. Go straight, for 100 meters

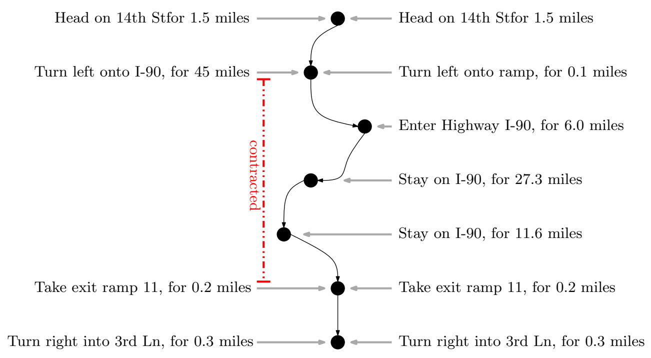
Head on 14th Stfor 1.5 miles
Head on 14th Stfor 1.5 miles
Turn left onto I-90, for 45 miles
Turn left onto ramp, for 0.1 miles
contracted
Enter Highway I-90, for 6.0 miles
Stay on I-90, for 27.3 miles
Stay on I-90, for 11.6 miles
Take exit ramp 11, for 0.2 miles
Take exit ramp 11, for 0.2 miles
Turn right into 3rd Ln, for 0.3 miles
Turn right into 3rd Ln, for 0.3 miles

Figure 2: A raw String of Micro Instructions is Condensed into a Meaningful Route Guidance.

2. Go straight, for 70 meters
is replaced by the single instruction
1. Go straight, for 170 meters

The second operation is edge removal, which allows us to mark (sub-)strings of micro-instructions as unneeded.

Example 2. *The following string of instructions*
1. enter roundabout
2. stay on roundabout
3. ...
4. leave roundabout
is replaced by the single instruction
1. enter roundabout and leave on k-th exit

Here, we can delete all incoming edges to all `stay on roundabout` nodes without breaking the final result. This rule is also called *roundabout counting*.

3.1 Evaporation of the Micro Guidance String

To compress the micro guidance string into a short but accurate description of the route, we need to identify important decision points and report only those. The intermediate portions of the string of micro instructions is either aggregated or simply omitted by the operations detailed above.

We interpret the micro guidance string as a set of directed edges pointing from one action point to the next. Each edge carries unit weight and, obviously, each node in this string corresponds to a graph node along the route. At first there is only one possible guidance in which every instruction has equal weight. Unit edge weight is assigned to resemble the particular situation. In other words, the edge weights of the TIG resemble a measure of *importance* of instructions. It is not related to the actual travel time or distance of the route in reality. We are going to modify the edge set of the initial TIG by applying a number of transformation operations. The resulting graph will contain several (partially) parallel paths with each resembling a different possible route guidance.

Modifications will be applied iteratively until we can extract a sufficiently small but accurate route guidance. The graph, which is just a string of edges at first, is transformed by inserting new *shortcut* edges and by deleting those edges which can be safely identified as unneeded. Nevertheless, these operations must always ensure that at least one path between source and origin node exists, i.e. that we always have *some* guidance.

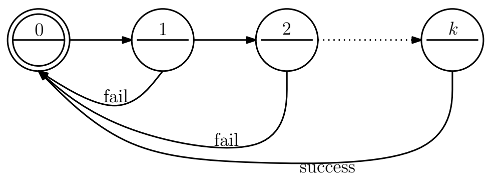

Figure 3: Finite State Machine that Accepts an Instruction Pattern.

3.2 Applying Pattern Matching

Now, that we have established a basic understanding of the method, we identify contractable parts of the guidance string by applying methods from pattern matching. Most of the rules to identify nodes for contraction (edges for removal) are simple rules as argued in Examples 1 and 2 of the previous section. We define these rules in terms of formal languages. A (sub-)path in the instruction graph is said to *contractable* (or *removable*) if there exists a recognizer based on a formal language. E.g. Examples 1 and 2 can be expressed in terms of a regular expression or equivalently finite state machines. In other words, it is recognizable by an acceptor that recognizes the formal language of a pattern. For the sake of an easy explanation we focus on a variant of finite state machines (FSMs), here, but the method is straight-forward to transfer to more powerful language representations such as context-free grammars, or even more powerful languages. A state change corresponds to turning from one road segment to the next. The only crucial property is that the recognizer automatically restarts both in case of failure as well as in case it accepts.

Figure 3 gives a sketch of the instruction pattern recognizer. Contrary to plain text-book FSMs, e.g. as explained in the formidable textbook of Sipser [22], the start state has a bit more decisiveness at hand. It decides whether it was reached through an accepting or rejecting state transition. Thus, the acceptor automatically restarts in case of failure.

The string of micro instructions, i.e. labels of the edges of the turn instruction graph, is taken as an input to the recognizer. In case of failure, i.e. when the pattern does not match, the state is reset and it is tested if a pattern beginning at the next micro instruction can be matched. In case of success, i.e. when a pattern does match, an edge from the beginning instruction to the last match is inserted into TIG. The new edge spans the entire matched pattern and represents an abstraction of the previous string of instructions and as such the edge information like length or duration of instruction is aggregated. Furthermore, the edge weight of the newly added (instruction) edge must reflect the relative importance of the shortcut over the string of micro instructions that it represents. Without loss of generality, we assume it to be of lower value. Note again that the principle of processing is transferable to any pattern matching technique.

3.3 Extracting Guidance

The turn instructions graph, after all pattern matching is done, is still a directed-acyclic graph. Also note that there may exist a number of different paths between origin and destination. Each of these paths represents a possible (but not necessarily reasonable) guidance output. Again without loss of generality, we assume that we are searching for a path of minimum weight as lower edge weights represent higher *importance*. While we could simply use Dijkstra's algorithm (again) to extract such a minimal guidance, we can actually retrieve a final guidance by exploiting the acyclicity of the TIG, which is much more cache-efficient as we explain below. A path search in such a graph can be done by a linear scan of the TIG from origin to destination node in the order of the initial turn instruction graph. Note that there is no need to actually maintain a priority queue, although the search is not performed in a tree: Initially, each node is associated with cost *infinity.* The search settles each node by visiting the nodes in ascending order. All outgoing edges of a node are scanned when settling and new minimum weights of the end nodes of any edge is noted as a tentative weight.

The correctness of the method follows from the order in which nodes are settled and that the resulting graph is directed and acyclic. The search does not relax any edges leading to nodes that are already settled. Note that the argument is symmetric for paths of maximum weight.

4. FLEXIBLE GUIDANCE

The above method can be used to generate a flexible and novel kind of guidance that also reflects local knowledge. Generally speaking, local knowledge implies fewer instructions, e.g. Fussell and Krauss [11] show that there is a substantial difference between the directions given by locals and non-locals. Consider two exemplary drivers with the same origin and destinations. The first does not know the area around the origin while the second one is a local taxi driver that knows his or her way around town very well.

Instead of giving explicit instructions in the form of "at junction x do maneuver y and continue for distance z" the taxi driver is happy with the simple instruction of "Drive to highway ramp x direction y and continue for distance z". Obviously, this only suffices for a driver with substantial local knowledge and it is fair to assume that the second one would be lost. The other driver does indeed need more detailed guidance or may be performing a random walk.

Turning away from this tongue-in-cheek argument, we assume that we have rules matching local knowledge or at least some abstract form of it. Hence, we can solve this problem by applying an additional layer of pattern matching that contracts the (sub-)guidance below a certain threshold of road importance. For example, it is reasonable to assume that not every single residential street needs to be mentioned on the way to the next highway ramp if the driver has sufficient local knowledge.

This locality based approach is also applicable to parts of the route that lie on the arterial network. Generally speaking, it is fair to assume that less guidance is necessary as soon as the driver enters the arterial network which is usually rather sparse and clearly arranged. Very detailed guidance can be necessary around origin and destination but not on the arterial network sufficiently far away from either. Consider the example shown in Figure 4. Node v is a big intersection between two major highways. The first route goes from u to v and it is of short range. Here it may make sense to indicate that intersection v is passed. Route u' to v' on the other hand is a long distance route and most probably passes by a number of larger intersections without actually leaving the highway. Here, it may make sense to *not* announce the passing of the intersection. This reflects the fact that humans tend to think of guidance in hierar-

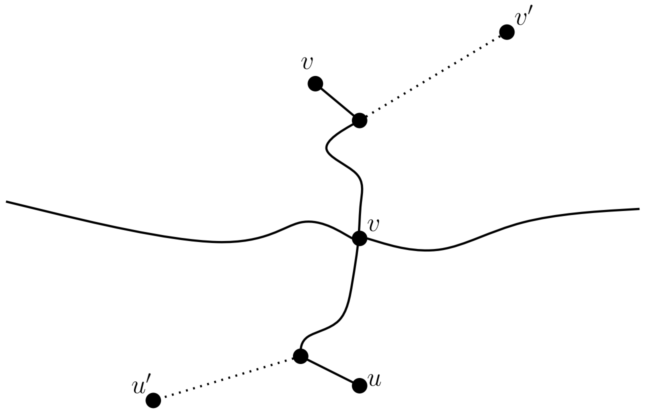

Figure 4: Intersection at Node v Need not be Announced in Every Case.

chical terms and that instructions need to provide necessary context along the route.

We solve this problem by adapting the guidance extraction sweep in the following intuitive way. Depending on the distance from origin and destination we simply discard instructions that are below a certain importance *threshold.* Recall that this is given by the weight of an edge in the TIG and if necessary rules must be adapted to do so. The threshold can be given as an absolute value that is applied once a certain distance from either origin or destination has been reached. Also, this can be done by more complex functions like Gaussian mixtures that reflect a threshold value as a function of the shortest distance to either origin or destination.

5. HANDLING AMBIGUOUS NAMES

So far, the (implied) assumption was that street names are not ambiguous and that each street has one and only one name. The names of streets may be ambiguous in the sense that a certain street has more than one name. For example, a locally used name and also a reference that identifies it as some part of the arterial network. Other examples are that distinctly referenced highways share some part of the same physical asphalt for some distance. We show how to handle this situation by (again) modeling the problem as a shortest path search in an edge-expanded graph.

The set of road names in the route (guidance) resembles the set of nodes in the search graph. The order of the route defines the order of the graph nodes, i.e. node i^0 is the i-th route segment. Road segments with multiple names are resembled by the same number of *duplicated* nodes, i.e. the k names of road segment j is represented by nodes $j^0, \ldots, j^{k-1}$. An edge (i^m, j^n) is inserted into the graph when it is possible to turn from node i to j with the following edge weights: The weight is 0 if and only if the names m and n are equal, otherwise the weight is 1. We further reduce the set of edges by only inserting (name change) edges when the name sets of two adjacent nodes actually differ. Instead of inserting all possible pairwise name changes only the edges are inserted that carry the previously chosen name forward. The search for a shortest path in this graph yields a path with the least possible number of name changes.

Assuming that the number of different names for a piece of the route (guidance) is bounded by k, the resulting search graph has the following worst-case space requirements. For each of the n pieces of route (guidance) at most k nodes are added to the graph. Assuming that the set of names changes with each piece we need to add at most $k \times k$ edges for each piece which yields and overall worst case space requirement of

$$\mathcal{O}(n \cdot k^2)$$

which is feasible in practice as the upper bound is a) not sharp in reality and b) the value k is a rather small single-digit constant. Furthermore, it is fair to assume that the observed ratio of the numbers of nodes and edges is close to 1. Thus, in practice, even a trivial implementation of Dijkstra's algorithm is bounded by a linear number of operations.

6. PRELIMINARY RESULTS

We already implemented a simplified variant the above system as a module for Project OSRM[1], an open-source and high-performance routing engine that facilitates OpenStreetMap data. We perform a greedy scan of the micro-instruction string using very simplistic rules. Every pair of adjacent streets with a common name is merged and we also implemented roundabout counting. This already leads to favorable results with running times that are dominated only by allocating output strings. Furthermore, in this setting we do not have to actually perform an explicit extraction step but can construct the guidance on-the-fly as we sweep through the string of micro instructions. As a result the run-time is dominated by the necessary memory allocation to provide enough room to actually store the final guidance.

7. CONCLUSION AND FUTURE WORK

We described the ongoing algorithmic research to provide flexible textual route guidance provided by a routing service. We detailed the graph model and gave a description of the algorithmic frame work. The preliminary results gathered at the time of writing look promising and we are hopeful to publish further results soon.

There are several directions for future work. First, we want to provide a full-fledged implementation of the described methods. Second, we want to conduct a thorough experimental evaluation using well-tuned recognizers to validate the efficiency and effectiveness of our method. Third, we would like to conduct a theoretical analysis in terms of computational complexity for all the sub-problems involved and also would like to establish tighter bounds. This has to be done perhaps empirically. Fourth, we aim to apply the algorithm engineering paradigm to fine-tune the results in terms of quality and efficiency and to identify the viable tuning parameters for our method. Last but certainly not least, we look forward to implementing the flexible guidance method as an open-source project. It will deliver a new and previously unseen way users interact with the map to generate route guidance that reflects local knowledge and preferences.

[1]`http://project-osrm.org`

8. REFERENCES

[1] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension and provably efficient shortest path algorithms. *submitted*, 2013.

[2] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In *International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *LNCS*, pages 230–241. Springer, 2011.

[3] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: improving usability through generalization. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'01)*, pages 241–249. ACM, 2001.

[4] Julian Arz, Dennis Luxen, and Peter Sanders. Transit Node Routing Reconsidered. In *International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *LNCS*, pages 55–66. Springer, 2013.

[5] Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum Time-Dependent Travel Times with Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4), 2013.

[6] Bing Maps. http://maps.bing.com.

[7] Ulrik Brandes and Barbara Pampel. Orthogonal-ordering constraints are tough. *Journal of Graph Algorithms and Applications*, 17(1):1–10, 2013.

[8] Daniel Delling, Andreas Gemsa, Martin Nöllenburg, and Thomas Pajor. Path schematization for route sketches. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'10)*, volume 6139, pages 285–296, 2010.

[9] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[10] Matt Duckham, Stephan Winter, and Michelle Robinson. Including landmarks in routing instructions. *Journal of Location Based Services*, 4(1):28–52, 2010.

[11] S. R. Fussell and R. M. Krauss. Coordination of knowledge in communication: effects of speakers' assumptions about what others know. *Journal of Personality and Social Psychology*, 62(3):378–91, 1992.

[12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[13] Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In *International Symposium on Experimental Algorithms (SEA'11)*, pages 100–111. Springer, 2011.

[14] Google Maps. http://maps.google.com.

[15] Peter E. Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[16] Hiroshi Imai and Masao Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36(1):31–41, 1986.

[17] Johannes Kopf, Maneesh Agrawala, and Michael F. Cohen. Automatic generation of destination maps. *ACM Transactions on Graphics (SIGGRAPH Asia 2010)*, 29(5):158:1–158:12, 2010.

[18] Dennis Luxen and Dennis Schieferdecker. Candidate Sets for Alternative Routes in Road Networks. In *International Symposium on Experimental Algorithms (SEA'12)*, volume 7276 of *LNCS*, pages 260–270. Springer, 2012.

[19] Dennis Luxen and Christian Vetter. Real-Time Routing with OpenStreetMap data. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'11)*, pages 513–516. ACM, 2011.

[20] Mark Ovenden. *Metro Maps of the World*. Capital Transport Publishing, 2003.

[21] Falko Schmid, Kai-Florian Richter, and Denise Peters. Route aware maps: Multigranular wayfinding assistance. *Spatial Cognition & Computation*, 10(2-3):184–206, 2010.

[22] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning EMEA, 2012.

[23] Christian Sommer. Shortest-Path Queries in Static Networks, 2012. submitted. Preprint available at `http://www.sommer.jp/spq-survey.htm`.

[24] Lars Volker. Route Planning in Road Networks with Turn Costs, 2008. Universität Karlsruhe, Fakultät für Informatik, Student Research Project. `http://algo2.iti.uni-karlsruhe.de/documents/routeplanning/volker_sa.pdf`.

[25] Matthias Westphal and Jochen Renz. Evaluating and minimizing ambiguities in qualitative route instructions. In *ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS '11)*, pages 171–180. ACM, 2011.

[26] Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.