Candidate Sets for Alternative Routes in Road Networks* (Extended Abstract)

Dennis Luxen and Dennis Schieferdecker
{*luxen,schieferdecker*}*@kit.edu*
Karlsruhe Institute of Technology
Karlsruhe, Germany

The following pages present an extended abstract of a previous publication at the 11th International Symposium on Experimental Algorithms (Luxen and Schieferdecker 2012).

Introduction and Related Work

Routing services have evolved over the past years. Providing only a shortest path is no longer enough. Users expect to be presented with a set of reasonable alternatives.

We show how to engineer previous algorithms to provide reasonable alternative paths with better efficiency. Building on these results, we introduce the concept of *candidate via nodes* to further speed up the computation by an order of magnitude. We show how to perform preprocessing and query variants efficiently. Finally, we provide an extensive experimental evaluation of our method.

The shortest path problem can be solved by Dijkstra's seminal algorithm (Dijkstra 1959). Heuristics to prune the search space provide *goal direction* and help with scaling. An early goal-directed technique with substantial speedups is *arc flags* (Lauther 1997; Möhring et al. 2007). The road network is partitioned into regions and each edge stores a flag to indicate if there is a shortest path into a region over the edge. Techniques exploiting the *hierarchy* inherent to a road network assume that sufficiently long routes enter the arterial network at some point, e.g. a national road or highway. *Contraction Hierarchies (CH)* (Geisberger et al. 2012) is probably the most prominent of these techniques. The road network is augmentend by carefully chosen *shortcut* edges while all unnecessary edges are removed. Road networks of continental size can be preprocessed within minutes and (bidirectional Djkstra) queries run in the order of one hundred microseconds. Reconstructing the complete shortest path requires roughly the same time. The fastest CH variant is *CHASE* (Bauer et al. 2010) that combines CH with arc flags. Its queries run in the order of ten microseconds. (Abraham et al. 2010; 2011a) give analyses of the theoretical performance of speedup techniques to Dijkstra's algorithm and present an efficient implementation (Abraham et al. 2011b) with query times below one microsecond.

*Supported by the German Research Foundation (DFG) within the Research Training Group GRK 1194 "Self-organizing Sensor-Actuator-Networks" and by DFG grant SA 933/5-2.

Alternative paths that combine two shortest paths over a *via node* are used by *Choice Routing* (Cambridge Vehicle Information Tech. Ltd), also referred to as *plateau method*. Even though not entirely published, this method yields good alternatives in practice. The road network is modelled as a graph $G = (V, E)$. Queries grow shortest path trees from origin s and destination t. *Plateaus* $\langle u, \ldots, v \rangle$ on these trees yield candidates for alternative paths, i.e. follow the forward tree from s to u, the plateau from u to v, and finally the reverse tree from v to t. Formally, a plateau is a maximal path on both trees and unambiguously described by any of its nodes – called via node. The identified candidate paths feel natural as each sufficiently small sub-path of the described routes remains a shortest path. Actual alternatives are chosen from them according to an unspecified heuristic.

This general approach received further discussion by (Abraham et al. 2013). The authors formally define good alternatives –called *admissible*– and report on how to compute them efficiently for the first time. *Stretch* to the shortest path, *overlap* to previously chosen paths and *local optimality* are introduced as quality criteria. Meeting nodes of both shortest path trees are considered as via nodes and define candidate alternative paths. These paths are ranked with respect to the quality criteria and a best one is chosen.

Algorithmic Approach

We build upon the approach of (Abraham et al. 2013) and show how to find admissible alternatives even faster and with an improved success rate. Their algorithms *X-BDV* and *X-CHV* are based on bidirected Dijkstra and CH, respectively. Simple engineering allows to roughly half query times compared to X-CHV while retaining all other quality measures: We exchange CH by CHASE and store shortcuts pre-unpacked. The resulting algorithm, *X-CHASEV*, is used as our baseline henceforth.

The analyses in (Abraham et al. 2010) show that speedup techniques to Dijkstra's algorithm work well for graphs in which all shortest paths leaving a region are *covered* by a small node set. This leads to the following assumption:

Assumption 1 (small number of alternatives) *If the number of shortest paths between two sufficiently far away regions of a road network is small, so is the number of plateaus for Choice Routing. Likewise, the number of admissible alternatives is small and they can be covered by a few nodes.*

Table 1: Query performance for the 1st and 2nd alternative.

	p=1		p=2	
algorithm	time [ms]	success rate[%]	time [ms]	success rate[%]
X-BDV	11.5 s	94.51	12.3 s	80.60
X-CHV	3.488	88.59	1.771	64.75
X-CHASEV	2.756	88.59	0.797	64.75
single-level	0.254	90.05	0.438	70.22
multi-level	0.188	90.06	0.386	70.40

We make use of this assumption by partitioning the road networks and computing a set of via node candidates for each pair of regions. During a query we identify the respective regions of origin and destination and look up the pre-computed candidates of this region pair. Then, we only need to check whether any of them yields an admissible alternative which is much cheaper than having to discover them first. Via node candiate sets of neighboring regions or within one region are usually too large to check efficiently. Thus, we either apply the baseline algorithm for such queries (*single-level approach*) or apply a second, fine-grained level of partitioning (*multi-level approach*). For neighboring regions or within one region of the fine partitioning we still default to the baseline algorithm. This only occurs for very local –and thus cheap– queries.

Precomputation of via node candidate sets is done by *bootstrapping*, i.e. the query (or fallback) algorithm is run between boundary nodes of region pairs to discover candidate nodes. We refer to the full paper for a more detailed explanation of efficient and parallel prepocessing.

Experimental Results

The algorithms are implemented in C++ and compiled with g++ 4.5 using full optimizations. Queries are performed on one core of an Intel Core i7-920 at 2.66 GHz with 12 GiB main memory. Parallel preprocessing is done on 4 AMD Opteron 6168 CPUs at 1.90 Ghz with 256 GiB memory.

Preprocessing of single-level via node candidates requires 2.38 h and produces 1.74 MiB additional data. Multi-level via node candidates take additonal 1.96 h to compute and add 7.17 MiB overhead. We identify 6.74 and 10.26 candidates for each region pair for the first and second alternative, respectively. Multi-level region pairs require 12.20 and 15.09 candidates on average.

Query results in Table 1 report on the average of 10 000 runs with origin and destination chosen at random. Testing about 2 candidates is sufficient on average for finding a first alternative while 4 have to be checked for the second alternative, for both single-level and multi-level approach. Note that query times of the "gold standard" X-BDV are given in seconds and not milliseconds. Figure 1 depicts two exemplary results of our algorithm.

One result, we take from our experimental evaluation is the low number of via node candidates. We regard this as an indication that Assumption 1 holds. For more extensive results and for further applications of our approach we again refer to the full paper (Luxen and Schieferdecker 2012).

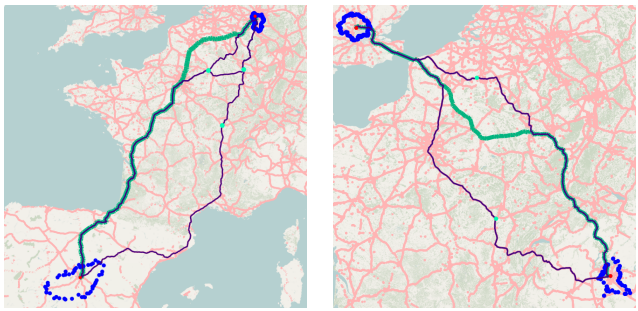

Figure 1: Alternative paths generated by our method. The shortest path is depicted by a bold line and alternative paths by thin lines. Via nodes are marked by light dots, boundary nodes of the origin and destination region by dark dots.

References

Abraham, I.; Fiat, A.; Goldberg, A. V.; and Werneck, R. F. 2010. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10)*.

Abraham, I.; Delling, D.; Fiat, A.; Goldberg, A. V.; and Werneck, R. F. 2011a. VC-Dimension and Shortest Path Algorithms. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP'11)*.

Abraham, I.; Delling, D.; Goldberg, A. V.; and Werneck, R. F. 2011b. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*.

Abraham, I.; Delling, D.; Goldberg, A. V.; and Werneck, R. F. 2013. Alternative Routes in Road Networks. *ACM Journal of Experimental Algorithmics* 18.

Bauer, R.; Delling, D.; Sanders, P.; Schieferdecker, D.; Schultes, D.; and Wagner, D. 2010. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics* 15.

Cambridge Vehicle Information Tech. Ltd. Choice Routing. http://camvit.com.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* 1.

Geisberger, R.; Sanders, P.; Schultes, D.; and Vetter, C. 2012. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science* 46.

Lauther, U. 1997. Slow Preprocessing of Graphs for Extremely Fast Shortest Path Calculations. Workshop on Computational Integer Programming at ZIB.

Luxen, D., and Schieferdecker, D. 2012. Candidate Sets for Alternative Routes in Road Networks. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA'12)*.

Möhring, R. H.; Schilling, H.; Schütz, B.; Wagner, D.; and Willhalm, T. 2007. Partitioning Graphs to Speedup Dijkstra's Algorithm. *Journal of Experimental Algorithmics* 11.