Impact of Shortcuts on Speedup Techniques*

Reinhard Bauer, Daniel Delling, and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany,
{rbauer,delling,wagner}@ira.uka.de

Abstract. In [2], we observed that SHARC benefits from adding shortcuts to the graph. In this preliminary work, we show that plain Arc-Flags [8] and reach-based routing [7] can be accelerated by simply adding additional shortcuts to the graph.
It turns out that arc-flags can be accelerated by a factor of up to 4, while reach-based routing benefits by a reduction of the settled nodes of a factor of up to 5.5 and with respect to runtime by a factor of up to 1.6.

1 Introduction

Computing shortest paths in networks is used in many real-world applications like routing in road networks, timetable information, or air-plane scheduling. In general, DIJKSTRA's algorithm [5] can solve this problem. Unfortunately, the algorithm is too slow to be used on huge datasets, e.g. the US road network has more than 20 million nodes. In order to reduce query times for typical instances like road or railway networks, several *speed-up techniques for* DIJKSTRA*'s algorithm* have been developed during the last years.

Related Work. A recent overview on speed-up techniques can be found in [3] and [12].

Overview. In Section 2, we briefly show how Arc-Flags and reach-based routing can benefit from additional edges. The key observation is that by simply favouring hop minimal shortest paths over other shortest paths in the preprocessing step, the query algorithm *automatically* uses shortcuts instead of the paths they represent. An experimental evaluation is located in Section 3 showing that the Arc-Flags approach is accelerated by a factor of up to 4, while reach-based routing benefits by a reduction of the settled nodes of factor up to 5.5 and of runtime by a factor up to 1.6. We conclude our work with possible future work in Section 4.

2 Speed-Up Techniques

Here, we briefly present those speed-up techniques—namely Arc-Flags and Reach—which are evaluated in Section 3 and explain why those techniques benefit from additional shortcuts.

* Partially supported by the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

Arc-Flags [9, 8, 10]. This approach uses a *pruning* strategy, i.e. by attaching additional data to edges, a modified DIJKSTRA checks whether an edge can or cannot be on the shortest path to the target. More precisely, the Arc-Flag approach partitions the graph into cells and attaches a label to each edge. A label contains a flag for each cell indicating whether a shortest path to the corresponding cell exists that starts with this edge. As a result, a bidirectional Arc-Flag DIJKSTRA often *only* visits those edges which lie on the shortest path of a long-range query. However, no speed-up can be achieved for queries within a cell and the effort of the preprocessing is very high. In this work, we use the variant as described in [10].

As we observed in [2], query performance of arc-flag based routing can be accelerated by adding additional shortcuts to the graph. During preprocessing, we favor paths with less hops over those with more hops. Then, an arc-flag query uses shortcuts instead of the paths they represent.

Reach is a centrality measure introduced in [7] that is used for the reach-based pruning speed-up technique. The version of reach described here corresponds to the one stated in [6]. The notion of reach derives from the observation that often, when having a long distance shortest path, only at the beginning and the end of the path 'local' edges are used. Intuitively speaking, the reach of an edge is high, if it lies in the middle of at least one long shortest path.

Given a path P_{st} from s to t and an edge (u, v) on P_{st}. The reach of (u, v) with respect to P_{st} is the minimum of the length of the two subpaths P_{sv} and P_{ut}, i.e

$$reach_{P_{st}}(u, v) = \min\{\text{length}(P_{sv}), \text{length}(P_{ut})\} \ .$$

The reach of an edge (u, v) is defined to be the maximum over all shortest paths P containing (u, v) of the reach values of (u, v) with respect to P, i.e.

$$reach(u, v) = \max_{P \in SP(u,v)} \{reach_P(u, v)\}$$

where $SP(u, v)$ denotes the set of all shortest paths containing (u, v). Note that we use the convention that the maximum over the empty set is zero.

3 Experiments

In this section, we present an experimental evaluation of shortcuts on road networks. Our implementation is written in C++ (using the STL at some points). As priority queue we use a binary heap. Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.3. The machine is clocked at 2.6 GHz, has 16 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3.

Input. We use two different road networks as input. One represents New York. It is taken from the DIMACS webpage [4], and has 264 346 nodes and 730 100 edges. The second network is a subgraph of the German road network, provided by PTV AG [1]

Table 1. Performance of Arc-Flags and Reach with and without additional shortcuts.

	Arc-Flags			Reach		
	PREPRO	QUERY		PREPRO	QUERY	
	time	#settled	time	time	#settled	time
inputs	[h:m]	nodes	[μs]	[h:m]	nodes	[μs]
New York w/o shortcuts	0:14	450	83.4	8:30	8293	2173
New York with shortcuts	0:16	56	21.0	8:41	2021	1331
Karlsruhe w/o shortcuts	0:12	253	52.7	9:34	5643	3497
Karlsruhe with shortcuts	0:13	48	17.9	10:17	1008	2379

for scientific use. It has 225 624 nodes and 572 182 edges and shows the city of Karlsruhe and surrounding areas.

Our external shortcuts derive from SHARC [2]. We run a full SHARC-preprocessing and extract the added shortcuts. Finally, we enrich the above mentioned networks by those shortcuts. For New York, the number of added shortcuts is 238 714, the corresponding figure for Karlsruhe is 112 107.

Arc-Flags. Table 1 shows the performance of bidirectional Arc-Flags on our inputs with and without additional shortcuts. As partitioning, we apply a SCOTCH [11] partition of 128 cells. We observe that adding shortcuts increase preprocessing time by $\approx 10\%$. This additional time is well spent because query performace increases by a factor of 3-4 in terms of query times and factor 5-8 in terms of search space. The reason for this gap is that the number of edges is higher for shortcut enriched graph. Hence, a query has to investigate more edges per node.

Reach. For reach based routing we compute exact node reach values as preprocessed data and a self-bounding distance balances query strategy. The results for reach are quite similar with respect to preprocessing time (increases only $\approx 7\%$) and number of settled nodes (decreases by factor 4-5.5). Unfortunately running time decreases only by factor 1.3 - 1.6.

4 Conclusion and Outlook

In this work, we showed that Arc-Flags and reach based pruning can benefit from externally computed shortcuts. We used shortcuts originating from the SHARC-technique and observed speed-ups up to factor 5.

There is plenty of future work on the problem. A theoretical analysis on the benefit of shortcuts would be helpful. For practical applications it would be good to develop and experimentally test shortcut-generating heuristics for other graph classes than road networks.

References

1. PTV AG - Planung Transport Verkehr, 1979. `http://www.ptv.de`.

2. R. Bauer and D. Delling. SHARC: Fast and Robust Unidirectional Routing. In I. Munro and D. Wagner, editors, *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08)*, pages 13–26. SIAM, 2008.
3. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. Submitted for publication, 2008.
4. C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors. *9th DIMACS Implementation Challenge - Shortest Paths*, November 2006.
5. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.
6. A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In *Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX'06)*, pages 129–143. SIAM, 2006.
7. R. J. Gutman. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 100–111. SIAM, 2004.
8. E. Köhler, R. H. Möhring, and H. Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, Lecture Notes in Computer Science, pages 126–138. Springer, 2005.
9. U. Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. volume 22, pages 219–230. IfGI prints, 2004.
10. R. H. Möhring, H. Schilling, B. Schütz, D. Wagner, and T. Willhalm. Partitioning Graphs to Speed Up Dijkstra's Algorithm. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, Lecture Notes in Computer Science, pages 189–202. Springer, 2005.
11. F. Pellegrini. SCOTCH: Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package, 2007.
12. D. Schultes. *Route Planning in Road Networks*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, February 2008.