Fast Generation of Many Shortest Path Alternatives

Stéphanie Vanhove[1], Veerle Fack[2]

[1,2]Department of Applied Mathematics and Computer Science
Ghent University
Krijgslaan 281 – S9, 9000 Ghent, Belgium

Email: [1]Stephanie.Vanhove@UGent.be, [2]Veerle.Fack@UGent.be

1. Introduction

Route planning applications rely heavily on shortest path algorithms. However, in some cases a ranking of several possible "short" paths is desired, rather than a single shortest path. For example, when trying to optimize multiple parameters, say beautiful scenery as well as travel time, a possible solution is to calculate a ranking of the fastest routes and then pick the most scenic one by hand. Also techniques for finding dissimilar paths (Akgün and Erkut 2000, Marti et al. 2009) often generate a large set of candidate paths, from which a suitable collection is selected. Figure 1 shows different feasible routes from Ghent to Antwerp, where it is up to the driver to choose one.

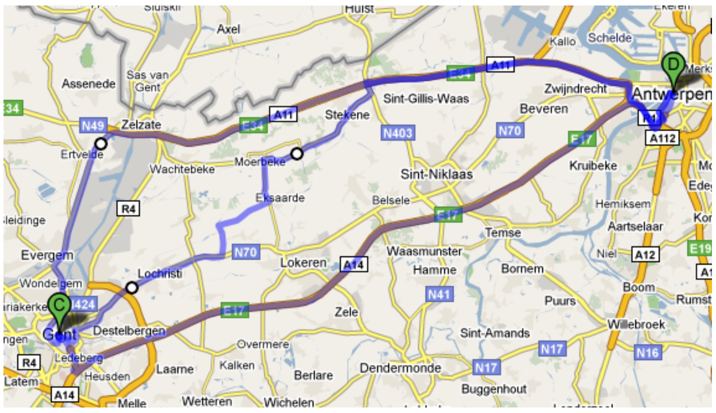

Figure 1. Different possible routes from Ghent to Antwerp.
(Image source: Google Maps)

Determining a ranking of shortest paths is called the *k* shortest paths problem, where *k* is the number of paths to be calculated. The road network is modeled as a graph, with road segments represented by arcs and intersections by nodes. Arcs have non-negative weight, representing distance or travel time. In this paper we focus on calculating the *k simple (*i.e. loopless) shortest paths, since loops are usually not useful for path finding in road networks. Yen's algorithm (Yen 1971) is the basis for many of the currently known algorithms (e.g. Herschberger et al. 2007, Gotthilf and Lewenstein 2009).

Path finding algorithms are often used in interactive applications, which makes them time-critical: a query time of e.g. 12 milliseconds versus 14 seconds can make a big difference. This fact motivates looking for a faster method which does not aim at finding the exact *k* shortest paths, but still misses only a few of them. Recently some theoretical approximation algorithms were developed (Roddity 2007, Bernstein 2010). In this paper we present a new heuristic that works very well in practice, i.e. which finds a majority of the paths, with only a slight increase in path length, and which is much faster than the exact algorithm.

2. Deviation Path Algorithms

Our heuristic is based on Yen's algorithm, which in its turn is an example of a deviation path algorithm. These are based on the idea that the *i*-th shortest path will always deviate at some node from a path previously found. Considering a path *P* of *n* nodes, there are only *n*-1 possible deviation positions: a deviating path *D* either differs from *P* immediately from the first node, or it coincides with *P* up to the 2nd, 3rd, … or *n*-1th node and then deviates from there on. Figure 2 sketches all possible situations for a path *P* with 5 nodes (note that *D* can deviate from a certain node to join *P* again later).

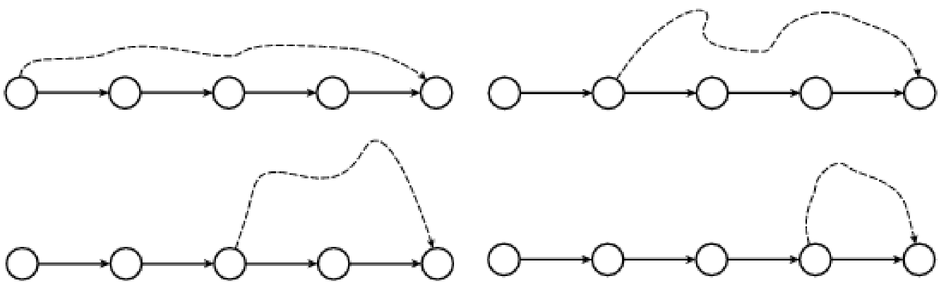

Figure 2. All possible deviations (dashed lines) from a 5-node path (solid lines).

Deviation path algorithms maintain a priority queue *Q* of candidate paths. In every iteration the next shortest path is fetched from *Q* and added to the results. This path is then used as a basis for calculating new candidate paths which are added to *Q*. This process is repeated until *k* shortest paths are found.

Different deviation path algorithms differ in their strategy for calculating new paths. Yen's algorithm calculates new paths from start node *s* to target node *t* based on a path *P* as follows:

```
1:  For every arc u-v on P:
2:      Remove u-v and all nodes preceding u in P from graph.
3:      $P'_1$ ← subpath from s to u in P.
4:      $P'_2$ ← shortest path from u to t in modified graph.
5:      P' ← append $P'_2$ to $P'_1$
6:      Add P' to Q.
7:      Restore graph.
```

The shortest path in line 4 can be calculated using any shortest path algorithm, such as the well-known algorithm of Dijkstra. Since this line is executed many times, this can be very time-consuming. The heuristic we describe in the next section aims at speeding up the algorithm by avoiding these shortest path calculations.

3. Heuristic for Calculating Deviations

At the start of the algorithm, we construct a backward shortest path tree *T*, which stores a shortest path from every node in the graph to the target node *t*, thus allowing fast retrieval of these paths.

Instead of actually computing the shortest path from *u* to *t* (line 4) we obtain an approximation for this shortest path by looking at the possible deviations obtained by concatenating every outgoing arc *u*-*x* (except *u*-*v*) with the shortest path from *x* to *t* fetched from *T*. However, since such a path may contain nodes and arcs which were in the meantime removed from the graph, it is necessary to check if the entire path still exists in the graph. Only if such is the case the path is appended to P'_1 and added to Q.

Figure 3 illustrates this idea. On the current path *P* from *s* to *t* (solid lines), red crosses indicate forbidden nodes and arcs. To find a detour from *u* to *t*, we consider the other outgoing arcs from *u* (dashed lines) and look up the paths from these neighbours to *t* (dotted lines) in the shortest path tree *T*.

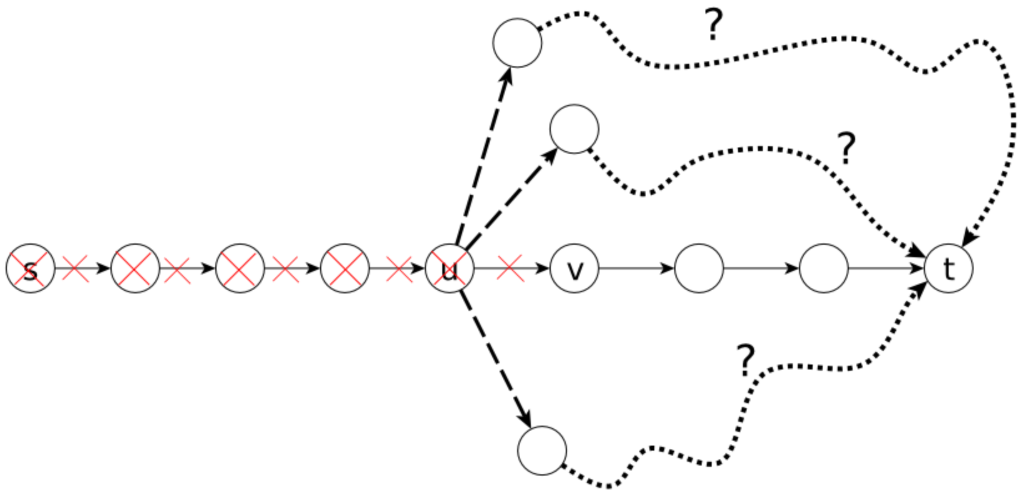

Figure 3. How the heuristic works.

Complexity

The algorithm of Yen has a time complexity of O(*k n* (*m* + *n* log *n*)), with *n* the number of nodes and *m* the number of arcs. However, road networks are sparse so $m = \Theta(n)$, resulting in a time complexity of O($k\ n^2$ log *n*). Our heuristic reduces this time complexity to O($k\ n^2$). However, since this upper bound is hardly ever reached, the speed-up is much better in practice, as the results will show. Details are omitted here for space reasons.

4. Results and discussion

We compared results of the heuristic with exact results for several European road networks. In this paper we present only results for the Navteq Belgian road network, but similar results for other road networks were obtained.

4.1 Quality of the paths found

Since our heuristic approach does not guarantee an exact set of *k* shortest paths, a comparison of the paths found is necessary. The results can be seen in Figure 4. The value *e(k)* represents the ranking of the path in an exact set of *k* shortest paths. E.g. if *e(k)* = 103 for a given query with *k* = 100, then the 100th path found by the heuristic is actually the 103rd shortest path, meaning that it “missed” three paths. The value *p(k)* shows the percentual weight increase of the path. E.g. if *p(k)* = 0.68% for *k* = 100, then the 100th shortest path found by the heuristic is 0.68% longer than the “real” 100th shortest path.

The heuristic performs surprisingly well in some cases, missing very few or even no paths at all. In other cases, more paths are missed, but even then the values for *p(k)* remain very small, typically less than 1%. Especially for routing applications, this is very acceptable since a travel time increase of less than 1% can almost be neglected.

4.2 Time performance

Of course a heuristic approach is only beneficial if it is significantly faster than the exact algorithm. Time measurements were performed for both the algorithm of Yen and the heuristic on an Intel dual core 2.13 GHz machine with 2 Gigabyte RAM running Linux. The algorithms were implemented, compiled and executed in Java version 1.6.0_16. The speed-up of the heuristic compared to the exact algorithm of Yen can also be seen in Figure 4.

The speed-up is clearly significant, even though it is very variable. For most queries the heuristic is more than 50 times faster than the exact algorithm, often even more than 100 or even 1,000 times faster. This shows that the heuristic is fast enough to be very useful in practice.

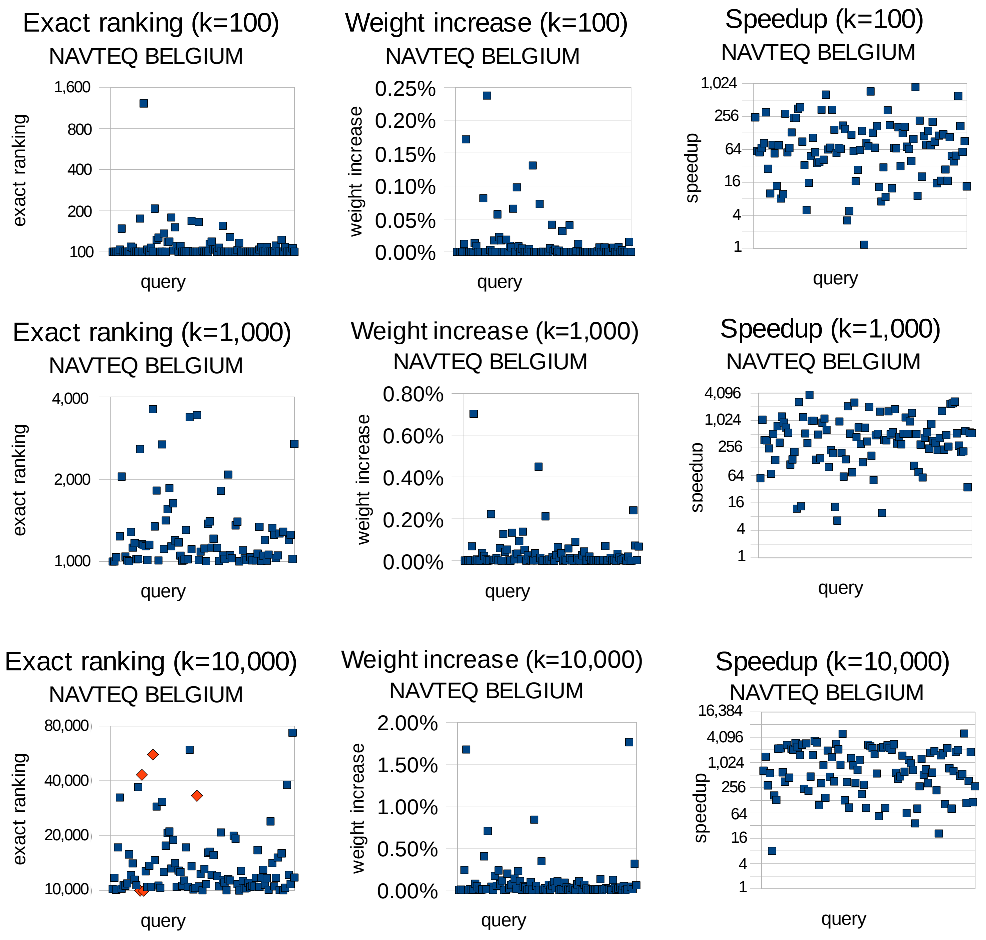

Figure 4. Results for the Navteq Belgium road network (564,477 vertices and 1,300,765 arcs). For k=100, 1,000 and 10,000 the exact ranking e(k) of the kth path found by the heuristic, the percentual weight increase p(k) and the speedup are shown, each time for 100 random queries. Dots marked in red indicate a lower bound instead of an exact value because of memory limitations.

4.3 Conclusion

The above experiments clearly show that significant speed-ups can be achieved by compromising only slightly on path quality. The new heuristic thus offers possibilities to serve as a basis for other algorithms and heuristics which make use of a large set of alternative shortest paths.

Acknowledgement

Stéphanie Vanhove is supported by a research grant of the Research Foundation - Flanders.

References

Akgün V and Erkut E, 2000, On finding dissimilar paths. *European Journal of Operational Research*, 121(2):232–246.

Bernstein A, 2010, A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, 742–755.

Gotthilf Z and Lewenstein M, 2009, Improved algorithms for the k shortest paths and the replacement paths problems. *Information Processing Letters*, 109:352–355.

Hershberger J, Maxel M, and Suri S, 2007, Finding the k shortest simple paths: a new algorithm and its implementation. *ACM Transactions on Algorithms*, 3(4):45.

Marti R, Velarde JLG and Duarte A, 2009, Heuristics for the biobjective path dissimilarity problem. *Computers and Operations Research*, 36:2905–2912.

Roddity L, 2007, On the k-simple shortest paths problem in weighted directed graphs. *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 920–928.

Yen JY, 1971, Finding the *k* shortest loopless paths in a network. *Management Science*, 17(11):712–716.