PhD Showcase: Applications of Graph Algorithms in GIS [*]

PhD Student: Stéphanie Vanhove
Department of Applied Mathematics and Computer Science
Ghent University
Krijgslaan 281 - S9
9000 Ghent
Belgium
stephanie.vanhove@ugent.be

PhD Supervisor: Veerle Fack
Department of Applied Mathematics and Computer Science
Ghent University
Krijgslaan 281 - S9
9000 Ghent
Belgium
veerle.fack@ugent.be

ABSTRACT

This paper describes ongoing PhD research on applications of graph algorithms in Geographical Information Systems. Many GIS problems can be translated into a graph problem, especially in the domain of routing in road networks. Our research aims to evaluate and develop efficient methods for different variants of the routing problem.
Standard existing shortest path algorithms are not always suited for use in road networks, e.g. in a realistic situation forbidden turns and turn penalties need to be taken into account. An experimental evaluation of different methods for this purpose is presented.
Another interesting problem is the generation of alternative routes. This can be modelled as a k shortest paths problem, where a ranking of k paths is desired rather than only the shortest path itself. A new heuristic approach for generating alternative routes is presented and evaluated.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and Layout*; E.1 [**Data Structures**]: Graphs and Networks; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms, Network problems*

General Terms

Algorithms, Experimentation, Performance

Keywords

Graphs, road networks, routing, turn penalties, alternative shortest paths, k shortest paths, heuristics

[*]The SIGSPATIAL Special, Volume 2, Number 3, November 2010. Copyright is held by the author/owner(s).

1. INTRODUCTION

Road networks can easily be modelled as a graph where nodes represent intersections and dead ends and arcs represent directed road segments. Arc weights usually represent either distances or travel times. The recent popularity of route planners and navigation systems has renewed the interest in the applications of graph algorithms to road networks, especially routing algorithms. However, there are some additional requirements for these applications. On the one hand, the used models must represent the real world as realistically as possible. On the other hand, the algorithms must be very fast, since users prefer short query times and servers need to answer many queries. Even though the shortest path problem is a classic problem in graph theory, the existing standard algorithms such as the algorithm of Dijkstra [2] cannot always immediately be used for route planning. One example is the presence of forbidden turns in road networks. Standard shortest path algorithms do not take this into account at all, even though it would be unacceptable if a navigation system instructs a driver to take an illegal turn. Moreover, turns can imply additional waiting times, e.g. at stoplights, another issue which is not handled by standard shortest path algorithms. Figure 1 shows two situations where a standard shortest path algorithm would calculate either an incorrect or suboptimal route. Different

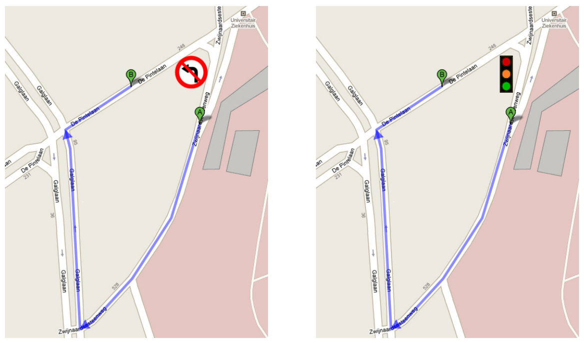


Figure 1: Two situations where a standard shortest path algorithm would take the obvious left turn to go from A to B. In reality however, the best route is the detour shown in the pictures because of either a forbidden turn (left) or a long waiting time at stoplights (right).

Figure 2: Alternative routes from Ghent to Antwerp in Google Maps.

methods have been proposed for this problem, but up to now it has remained unclear how well these algorithms perform on real-life road networks and how these algorithms compete with each other. A study of these algorithms is presented in Section 2.

Another interesting variant of the shortest path problem is the generation of alternative routes. This is very commonly used, as can be seen in Figure 2. In graph theory, the problem of calculating a ranking of shortest paths is called the *k shortest paths* problem, where k is the number of paths to be calculated. Not only can this be useful for generating alternative routes, but k shortest paths algorithms also serve as a basis for methods for optimizing multiple parameters. Such methods (e.g. Mooney et al. [8]) can be used to find e.g. a fast but scenic route. The most scenic route can be chosen from a ranking of the k shortest paths. Similarly, a set of dissimilar paths can be selected from a ranking of k shortest paths. This can be used for generating dissimilar routes for the transportation of hazardous materials in order to spread the risk. Such a method is presented by Dell'Olmo et al. [1]. However, algorithms for the k shortest paths problem tend to be very time-consuming. This is a major issue in interactive routing applications. On the other hand, in routing applications obtaining a *good* solution very fast can be more interesting than obtaining the *exact* solution a lot slower. In Section 3 we present a new heuristic which calculates an approximation of the k shortest paths with results of good quality much faster than the exact algorithm. Section 4 outlines the possibilities for future work.

2. TURN RESTRICTIONS

There are two kinds of turn restrictions in road networks: turns can either be forbidden (*turn prohibitions*) or imply an additional cost (*turn costs*). In our examples we will assume that U-turns are always forbidden. The next sections describe the different methods for the shortest path problem with turn restrictions and an evaluation of these methods.

2.1 Modelling turns

Direct method

Gutiérrez and Medaglia [5] present an adaptation of the algorithm of Dijkstra which takes turn prohibitions into account. We will call this method the *direct method* since it operates directly on the original graph, unlike the other considered methods. While the algorithm of Dijkstra assigns

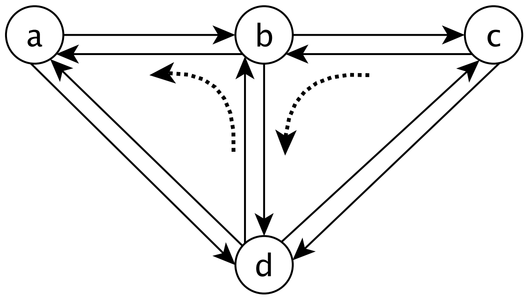


Figure 3: Graph with forbidden turns. Dashed arrows indicate forbidden turns: it is legal to move from arc (d, b) to (b, c) but it is illegal to move from arc (d, b) to (b, a).

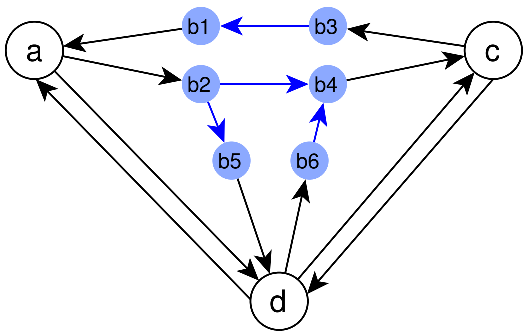


Figure 4: Node splitting: transformed graph for the graph in Figure 3. Node b is split into 6 nodes: b_1 .. b_6, each representing an arc incident with node b. Arcs between the split nodes represent legal turns. Split nodes and the arcs between them are shown in blue.

labels to nodes, the direct method assigns labels to arcs. Also, for every transition from one arc to another, a check is performed to make sure that the turn is not prohibited. Since the graph itself does not model turn prohibitions, this information is stored separately in a data structure outside the graph. Turn costs are not considered in their work. We have adapted the direct method in order to consider turn costs as well. Further details of this method are omitted for space reasons.

Node splitting

Kirkby and Potts [7] and Speičys et al [10] present another method called *node splitting*. This method requires a graph transformation. Every node in the graph with a turn cost or turn prohibition is split into several nodes: one for every incoming or outgoing arc. Then, for every legal turn, an arc is added between the two nodes representing the arcs of the turn. The weight of this new arc is the turn cost, which can be zero or more. Illegal turns can no longer be taken in the graph since there is no arc connecting the corresponding nodes. Figure 3 shows a graph with two forbidden turns. Figure 4 shows its transformed graph. The main ad-

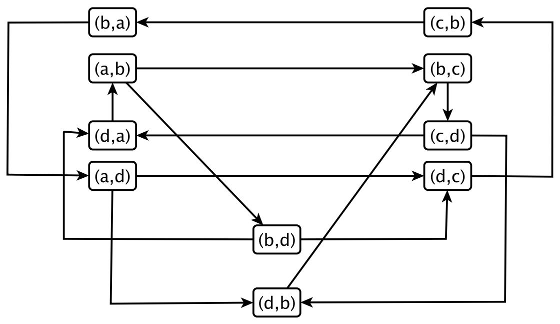


Figure 5: Line graph: transformed graph for the graph in Figure 3. Every node in the line graph represents an arc in the original graph. Arcs in the line graph represent legal turns in the original graph.

vantage of this method is that any standard shortest path algorithm, e.g. the algorithm of Dijkstra can be applied to the transformed graph, so no new algorithm is needed.

Line graph

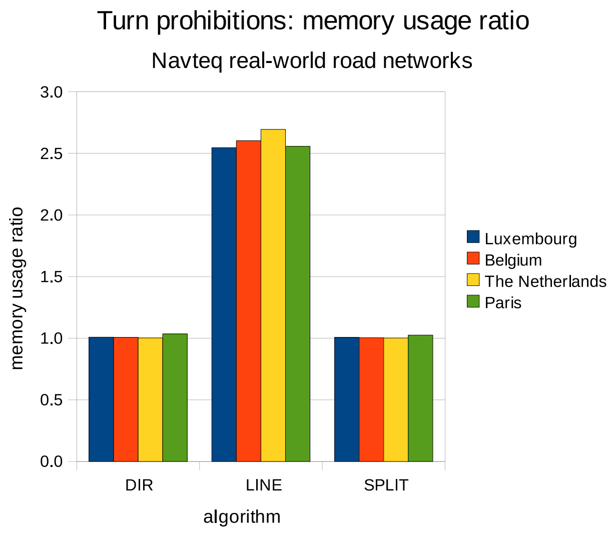
The *line graph* method is another graph transforming method presented by Winter [11]. While the node splitting method only affects those nodes with turn costs or turn prohibitions, the line graph method always transforms the entire graph. Nodes in the transformed graph represent arcs in the original graph. Arcs in the transformed graph represent legal turns in the original graph. Arc weights in the transformed graph represent arcs weights in the original graph as well as turn costs. The transformed graph is called the line graph. Figure 5 shows the transformed graph for the graph in Figure 3. Just like the node splitting method, this methods allows running any standard shortest path algorithm on the transformed graph.

2.2 Experimental evaluation

We performed several experiments aiming to evaluate the methods mentioned above on real-life road networks. Of course query time is important, so time measurements were performed. However, memory usage is important too. Two of the three methods require a graph transformation, which can possibly result in a much larger graph, while the direct method needs extra memory to store the information on turn restrictions separately. Therefore, memory usage was measured as well for the three methods. All algorithms were implemented, compiled and executed in Java version 1.6.0_03. All tests were run on an Intel dual core 2.13 GHz machine with 2 Gigabyte RAM running Linux. In the next paragraphs we make a distinction between turn prohibitions and turn costs, since different test data were used.

Turn prohibitions

For turn prohibitions, the experiments were performed on road networks provided by Navteq [9] Swith real-world turn prohibitions. The size of these road networks is in a range bounded by 39,883 (Luxembourg) nodes and 1,017,242 nodes (The Netherlands). As can be expected, only a small fraction of the turns is forbidden (less than 1%). The results can be seen in Figure 6. All results are ratios compared to the



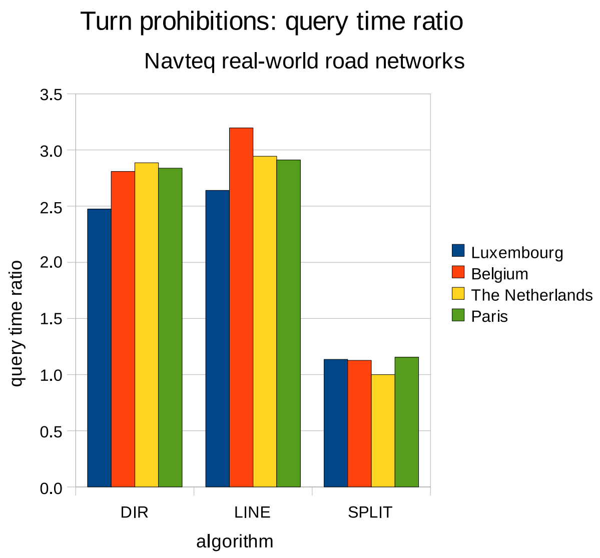
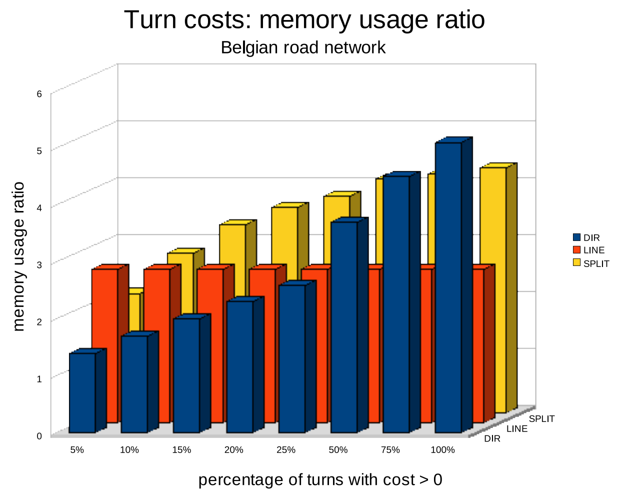


Figure 6: Turn prohibitions: results for Navteq real-world road networks. Memory usage ratio (top) and average query time ratio (bottom) are shown for 4 different road networks. Three methods are compared: the direct method (DIR), line graph (LINE) and node splitting (SPLIT).

algorithm of Dijkstra. E.g. if the memory usage ratio is 3, then the method needs 3 times more memory than the original graph representation without turn restrictions. If the query time ratio is 3, then the query time for this method is 3 times the query time of the algorithm of Dijkstra. The top chart shows that the direct method and node splitting take about the same amount of memory, while the line graph method has a much higher memory usage. When looking at the time measurements in the chart in the bottom however, the node splitting method is clearly the fastest. Hence, we can conclude that the node splitting method performs best on realistic road networks with turn prohibitions.



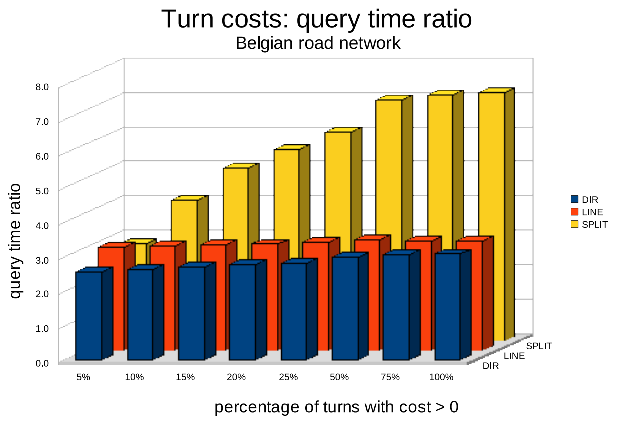


Figure 7: Turn costs: results for the Belgian road network. Memory usage ratio (top) and average query time ratio (bottom) for different percentages of turn costs are shown. Three methods are compared: the direct method (DIR), line graph (LINE) and node splitting (SPLIT). The graph has 458,403 nodes, 1,085,076 arcs and 2,922,504 turns. The original graph size is 57.72 MB. The average query time for a standard Dijkstra algorithm is 254.49 ms.

Turn costs

To the best of our knowledge, no real-life data with turn costs are available at this moment. To overcome this obstacle, real-world road networks were used but the turn costs were added randomly. The road networks are provided by the University of Karlsruhe in the DIMACS format [3] and represent the road networks for different European countries. In this abstract, results for the Belgian road network are shown, but results are similar for the other countries.

In theory, when applying turn costs to a road network, every turn has its own cost associated to it. However, in a real-life situation, it is very likely that a data provider does not provide a turn cost for every turn in the network, since visiting every turn would be an extremely expensive and time-consuming task. A data provider would probably focus initially on the busiest roads and intersections and possibly keep the less important roads for a later phase. The data could e.g. contain turn costs for 5% of the turns. For this reason, different percentages of available turn costs are considered in the experiments, namely 5%, 10%, 15%, 20%, 25%, 50%, 75% and 100%. It should be noted that 100% available turn costs can still be realistic if the turn costs are calculated automatically, e.g. based on the angle.

The results can be seen in Figure 7. The results are again ratios as explained in the previous section. The direct method and line graph method show very similar query times, which also seem to be independent of the percentage of turn costs. This can be expected since these methods transform the entire graph or perform no graph transformation at all, respectively, so the number of nodes in the final graph is independent of the percentage of turn costs for both methods. The node splitting method is never faster than the other two methods. On the other hand, memory usage seems to be independent of the percentage of turn costs for the line graph method but not for the direct method. This can be explained by the fact that the line graph always transforms the entire graph and doesn't store any additional information, while the direct method keeps the original graph but needs to store additional information for every turn cost. As can be seen in the chart, this leads to increasing memory usage for higher percentages of turn costs. The direct method appears to be more memory-efficient for lower percentages of turn costs. For higher percentages however, the line graph method is more memory-efficient than the direct method. So we can conclude that the direct method is best suited for graphs with fewer turn costs (up to about 25%) while the line graph performs better for graphs with many turn costs.

3. ALTERNATIVE ROUTES

3.1 K shortest paths

The second problem we discuss in this paper is the generation of alternative paths. We will assume that loops in the paths are forbidden, a natural assumption in road networks. For this problem - k shortest paths without loops - an influential algorithm was proposed by Yen [12], which was the basis for many of the currently known algorithms (e.g. Hershberger et al. [6], Gotthilf and Lewenstein [4]). However, as mentioned in the introduction, routing applications are very time-critical and the existing algorithms tend to be too slow for this purpose. Therefore, we developed a heuristic approach which does not aim to find an exact solution but is much faster than the exact algorithms while the results are still of good quality. In the next sections we describe the general principle (*deviation path algorithms*) on which most algorithms are based, present our heuristic approach and report the results.

3.2 Deviation path algorithms

Our heuristic is based on the algorithm of Yen [12]. Both are examples of *deviation path algorithms*, which are based on the fact that any shortest path in the ranking always deviates at some point from a path previously found. A path can either immediately deviate from a path from the start node, or coincide with a path up to some node and then deviate from it. Figure 8 shows all possible deviations from a path with 5 nodes (note that a path can deviate from another path to join it again later).

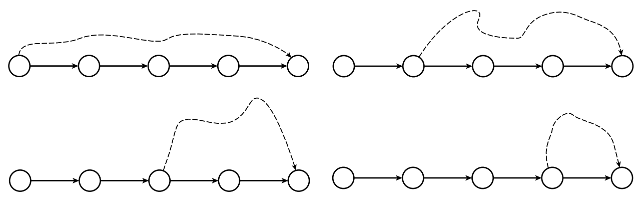


Figure 8: All possible deviations (dashed lines) from a path (solid lines) with 5 nodes.

Deviation path algorithms start by calculating the shortest path using any shortest path algorithm. In our work the algorithm of Dijkstra is used for this purpose. This shortest path is then added to a collection C (usually a priority queue). Then the algorithm fetches the shortest path P from C in every iteration, adds it to a list L containing the ranking of shortest paths found so far, and calculates deviations from P which are added to C. The algorithm is finished after k iterations. Algorithm 1 outlines this general principle, which is shared by all deviation path algorithms, who then differ in their method for calculating deviations.

Algorithm 1 Deviation path algorithms

```
Require: graph $G$, number of shortest paths $k$, source $s$, target $t$
Ensure: sorted collection $L$ of $k$ shortest paths
1: $P \leftarrow$ calculate shortest path from $s$ to $t$
2: add $P$ to a collection $C$
3: for $i$ from 1 to $k$ do
4:     $P \leftarrow$ shortest path in $C$
5:     remove $P$ from $C$
6:     add $P$ to $L$
7:     calculate deviations and add them to $C$ {algorithms differ here}
8: end for
```

3.3 Our approach

The method for calculating deviations used by our approach is similar to the method used in Yen's algorithm. The algorithm of Yen forbids every arc (v_i, v_{i+1}) on a path P from s to t one by one, and calculates the new shortest path P' from v_i to t. In this calculation, all the nodes on P preceding v_i are also forbidden. A new path from s to t is then created by appending P' to the subpath of P from s to v_i. This results in a very large amount of time-consuming shortest path calculations. Our heuristic aims to speed up the calculation of these shortest paths. Instead of performing so many shortest path calculations, the shortest paths are retrieved from precomputed information. The heuristic uses a backward shortest path tree T which is precomputed and thus computed only once. The shortest path from any node in the graph to the target node t can be looked up in T very fast. Instead of actually computing the shortest path from a node v_i to t, the heuristic calculates deviations by concatenating every outgoing arc (v_i, x) from v_i with $x \neq v_{i+1}$ to the shortest path from x to t fetched in T. Of course, the possibility exists that this path is no longer valid in the graph since some nodes and arcs have been forbidden in the meantime. This needs to be checked before creating the full $s-t$ path and adding it to C. Figure 9 illustrates this idea.

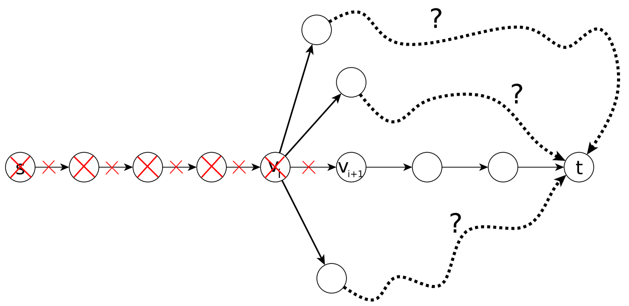


Figure 9: How the heuristic works. Solid lines indicate the current path P from s to t. Red crosses indicate forbidden nodes and arcs. A detour is necessary from v_i to t. Dashed lines indicate other outgoing arcs from v_i. Dotted lines indicate paths from these neighbours to t, which can immediately be looked up in the shortest path tree T. These paths are not allowed to pass through already forbidden nodes or arcs.

Complexity

When the algorithm of Dijkstra is used for shortest path calculations, Yen's algorithm has a time complexity of $O(kn(m + n \log n))$, with n the number of nodes and m the number of arcs in the graph. Since road networks are sparse, it can be assumed that $m = O(n)$, resulting in a time complexity of $O(kn^2 \log n)$ for the algorithm of Yen. Our heuristic reduces this time complexity to $O(n^2k)$ (details omitted for space reasons). Even though the complexities only differ by a logarithmic factor, the heuristic performs much better in practice. In the time complexity of the algorithm of Yen, a factor $O(n \log n)$ is attributed to shortest path calculations which can involve (almost) the entire graph. On the other hand, the heuristic does not perform these shortest path calculations, and the $O(n^2)$ factor is limited to iterating over the found paths. Although theoretically a path *can* have a length of n, this upper bound is never reached in practice, resulting in much faster running times. The results presented in the following section clearly confirm this statement.

3.4 Results

The heuristic was tested on several road networks. Figure 10 shows the results for the Navteq Belgian road network with k = 1,000. The results are similar for other road networks provided by Navteq and by the University of Karlsruhe. Three different parameters were tested. The first parameter (shown in the first chart of Figure 10) is the ranking of the k^{th} path found by the heuristic in an exact ranking of shortest paths. E.g. for k = 1,000, if the ranking of the 1,000th path found by the heuristic is 1,006, then the heuristic has missed 6 paths. The results show that the heuristic often misses very few paths or even no paths at all. In some other cases, more paths are missed, but always within acceptable bounds, as can be seen from the results for the second parameter in the second chart of Figure 10. This shows the weight increase for the k^{th} path found by the heuristic. E.g. if the weight increase is 0.80%, then the k^{th} shortest path found by the heuristic is 0.80% longer than the exact k^{th} shortest path. The results show that this value is always

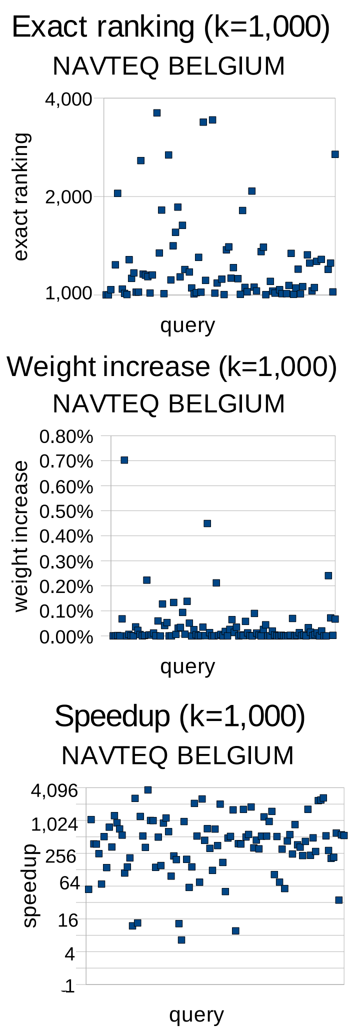


Figure 10: Results for the Navteq Belgium road network (564,477 nodes and 1,300,765 arcs) with $k = 1{,}000$. The exact ranking of the k^{th} path found by the heuristic, the percentual weight increase and the speedup are shown, for 100 random queries.

below 1%. A weight increase of less than 1% is neglectable in a road network, making the results of the heuristic very useful in practice. A third important parameter is query time, since of course a heuristic calculating an approximation is only useful if it is significantly faster than the exact algorithm. The speedup of our heuristic compared to the exact algorithm of Yen can be seen in the third chart. The results show that the heuristic is always faster than the exact algorithm, often hundreds or even thousands times faster. This is a very significant advantage of the heuristic for use in interactive routing applications.

4. FUTURE WORK

In the future we aim to further optimize our heuristic for the k shortest paths problem. Even though the heuristic performs well in most cases, there is currently still a small number of cases where the heuristic misses a substantial number of paths. We aim to further reduce this number or, ideally, eliminate these outliers. As mentioned in Section 3.1, k shortest paths algorithms can also serve as a basis for generating dissimilar paths. This can e.g. be interesting for the generation of alternative routes which can be used when weather conditions are not favorable on the usual route. In this case the alternative routes should coincide as little as possible with the first route, otherwise the alternative routes will suffer from the same weather conditions. We aim to develop a new method for this problem based on our heuristic for the k shortest paths problem. Eventually, it would be interesting to include turn restrictions in our methods for k shortest paths and dissimilar paths.

5. ACKNOWLEDGEMENT

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

6. REFERENCES

[1] P. Dell'Olmo, M. Gentili, and A. Scozzari. Finding dissimilar routes for the transportation of hazardous materials. In *Proceedings of the 13th Mini-EURO Conference on Handling Uncertainty in the Analysis of Traffic and Transportation Systems.*, pages 785–788, 2002.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[3] DIMACS. 9th dimacs implementation challenge on shortest paths. *http://www.dis.uniroma1.it/∼challenge9/*, 2005.

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

[5] E. Gutiérrez and A. L. Medaglia. Labeling algorithm for the shortest path problem with turn prohibitions with application to large-scale road networks. *Annals OR*, 157(1):169–182, 2008.

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

[7] R. F. Kirby and R. B. Potts. The minimum route problem for networks with turn penalties and prohibitions. *Transportation Research*, 3:397–408, 1969.

[8] P. Mooney and A. Winstanley. An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science*, 20:401–423, 2006.

[9] NAVTEQ. Navteq network for developers. *http://www.nn4d.com/*, 2007.

[10] L. Speičys, C. S. Jensen, and A. Kligys. Computational data modeling for network-constrained moving objects. *GIS '03: Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 118–125, 2003.

[11] S. Winter. Modeling costs of turns in route planning. *Geoinformatica*, 6(4):345–361, 2002.

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