Road segment selection with strokes and stability

T. C. van Dijk*
Chair for Computer Science I
Am Hubland, 97074
Würzburg, Germany
thomas.van.dijk@uni-wuerzburg.de

K. Fleszar
Chair for Computer Science I
Am Hubland, 97074
Würzburg, Germany
krzysztof.fleszar@uni-wuerzburg.de

J.-H. Haunert
Chair for Computer Science I
Am Hubland, 97074
Würzburg, Germany
jan.haunert@uni-wuerzburg.de

J. Spoerhase
Chair for Computer Science I
Am Hubland, 97074
Würzburg, Germany
joachim.spoerhase@uni-wuerzburg.de

ABSTRACT

In order to visualize a road network without producing visual clutter, a subset of all road segments needs to be selected. Many algorithms for road segment selection are based on a relevance score for edges in a network (for example betweenness centrality) and proceed by taking a greedy selection based on these weights. This can give dissatisfactory results. In order to improve readability, we introduce a stroke-based constraint and provide an efficient dynamic program that makes an optimal selection given this constraint. Next, we consider the computation of animated road selections from changing edge weights (for example a focus area that follows a moving user). Handling each time step of the animation individually can lead to distracting flickering effects. Here we introduce an optimization objective to achieve a more stable selection and provide a polynomial-time algorithm for solving it. While separately solvable in polynomial time, we show that the combination of the stroke constraints and stability optimization is $\mathcal{NP}$-hard.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Theory

*Corresponding author.

MAPINTERACT '13:, November 05 - 08 2013, Orlando, FL, USA

http://dx.doi.org/10.1145/2534931.2534936

Keywords

Map generalization, Road segment selection, Strokes, Algorithm design, Complexity

1. INTRODUCTION

The selection of a good subset of features from a geographic data set is a classical problem in map generalization. Today, this problem is particular relevant in mobile cartography, where smartphones and other small-screen devices are widely used for wayfinding, orientation, and exploration.

Many road segment selection systems can conceptually be split into two parts. First, determine which road segments are, in some way, important. A typical choice for this is betweenness centrality. Second, select segments based on this measure. Much research has gone into the first step (for example [9, 8, 13, 3]) with a simple implementation for the latter. Among the more advanced selection methods is the one of Kopf et al. [11], which is tailored to destination maps. Mackaness and Beard [12] have emphasized the importance of connectivity; to generalize a road network as much as possible while keeping all nodes connected, they have suggested reducing the network to a minimum spanning tree.

In this paper we look specifically at the second part, the selection, assuming the importance of the individual edges is already determined. In terms of graph algorithms, we are then looking at a subgraph problem in a weighted graph. To improve the quality of the results, we can impose constraints on the subgraph and modify the objective function. In this paper we propose two such improvements.

Our first improvement is based on *strokes*, a concept that has seen some use for detecting structure in road networks [14, 9, 16, 15]. Basically, a stroke is a path of almost collinear road segments. In large data sets strokes may be very long, so we do not insist on selecting entire strokes but instead require that the edges selected from the same stroke constitute a connected path. We give a polynomial-time algorithm to find optimal selections satisfying this constraint.

Our second proposal is for making animated selections. This can be used, for example, in a navigation system where the user focus traverses a predetermined route and thus the

importance of the road segments changes over time. Handling each time step of the animation individually can lead to distracting flickering effects. This problem has been observed in more aspects of animated map drawing, for example label placement. In order to avoid that labels flicker on and off when a user zooms in or out, Been et al. [1] have introduced *active ranges*. In the model of Been et al., an active range is an interval of scales. By requiring that each label is visible in exactly one active range, flickering effects are avoided. Similarly, active ranges (as intervals of rotation angles) have been defined for maps that can be rotated [6] and (as intervals of time) for maps that are centered on the position of a user who follows a precomputed route [5].

We argue that requiring only one active range for each road segment is too strict, since the user may have to visit the same or similar positions multiple times. Consider for example an entire day's route for a delivery truck, or convoluted road networks. Therefore, we allow an arbitrary number of active ranges for each road segment, but we consider a cost for changing the on/off state of an edge. We call this *stability* optimization. We give a polynomial-time algorithm to select an optimal animation given all weights in advance. Lastly we consider the combination of stroke constraints and stability optimization. If we allow the strokes to change over time, the *combined* problem becomes $\mathcal{NP}$-hard. The same holds for the case of fixed strokes and a slightly generalized version of stability (that our previously mentioned algorithm can handle). The complexity of the combined version when using the most basic version of stability remains unknown.

2. STROKE CONSTRAINTS

First we look at *stroke constraints*. We start by identifying a problem with greedy selection and propose a solution.

2.1 Modelling

As input we take a score, say a real number, for every edge in the road network. Let w_i be this score of road segment i.

A typical approach would be to select all edges with a score higher than a certain cutoff value. That gives a reasonable monotonicity property: any edge that we select has a higher score than any edge we do not select. One way to do this would be to select a fixed amount of edges and maximize the selected weight.

MaxWeight

Instance:	Set E of n edges. Weight w_i for each $i \in E$. Number k.
Question:	Subset $S \subseteq E$ of maximum weight such that $\lvert S\rvert = k$.

Alternatively, we could select edges such that the sum of their scores is at least a fixed fraction of the sum of all scores. We think this approach has a particularly elegant application when the scores represent probabilities. This is the case in the random traveler model proposed by Van Dijk and Haunert [3], where the score of a road segment represents the probability that, within a certain time, the user visits that segment. Selecting at least a fixed fraction of the sum of all scores then means that we select a certain amount of probability mass. We would want to satisfy this constraint with the minimum number of selected edges. Call this MinEdges. Both these problems can be solved by greedily selecting the edge with the highest score until the constraint is satisfied. For $\mathcal{O}(n)$ runtime without requiring sorted input, use linear-time selection [2].

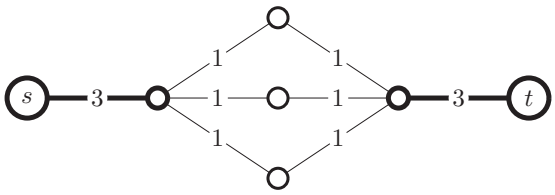


Figure 1: Selecting edges by using a score cutoff might select a disconnected set of edges. This might be undesirable, for example if the application needs to show a route.

(a) Greedy (b) Dynamic programming

Figure 2: Requiring that the selected network is connected does not guarantee that it is connected 'nicely.' See for example the gaps indicated in Figure 2(a). Figure 2(b) shows the corresponding solution found by our dynamic-programming algorithm. The weights are computed as in Section 5.

Throughout the rest of the paper we refer to this solution as the *greedy algorithm*. The solution to these problems might induce a disconnected graph. See Figure 1 for an example. It seems that we might want to guarantee connectedness of the selected edges. First, we note that by itself this is not exactly the condition we are looking for. See for example Figure 2: the selected network is connected, but the indicated gaps may mislead the map reader and are also visually displeasing. Furthermore, finding a maximum-weight connected subgraph of a given cardinality is $\mathcal{NP}$-hard [4]. These two issues lead us to pursue a different approach.

Our specific complaint about Figure 2 is about gaps in strokes. Therefore, we will introduce a constraint regarding strokes. For our purposes, a stroke is an ordered list of edges and every edge is in at most one stroke. We might consider just selecting entire strokes instead of individual edges, but this is not a good idea. Consider a long highway, which could easily be a single stroke. We do not want to force drawing the entire highway if we are just trying to draw a local selection that just happens to use a few of the highway's edges. For an example in an actual data set, see Figure 3. We say, instead, that a good selection of edges picks only a single interval on a stroke.

Definition 1 (Stroke constraint). *For all pairs of edges on the same stroke, if both are selected, then any edges inbetween them on the same stroke must also be selected.*

Again we can consider the two different objectives, of which we state one here.

(a) Edges (b) Strokes

Figure 3: Example of selecting edges versus selecting strokes. The weights have been calculated to display a route from ○ to ● [3]. Here we have greedily selected 90% of the score, either by selecting individual edges or entire strokes at once. When selecting strokes we end up selecting edges far away from the intended focus area, in this case particularly the long stroke leading south. We have clipped this stroke for this diagram: it actually continues for approximately twice as long.

MaxWeight-Strokes

Instance:	Set E of n edges, partitioned into strokes. Weight w_i for each $i \in E$. Number k.
Question:	Subset $S \subseteq E$ of maximum weight satisfying the stroke constraints, such that $\|S\| = k$.

2.2 Dynamic programming algorithm

We give a dynamic programming algorithm for these problems. First we solve MaxWeight-Strokes. Then we observe that this gives a solution to MinEdges-Strokes in the same time and space.

The algorithm is based on mutual recurrence relations F_{include} and F_{exclude}. Let 1 to n be a numbering on the edges with the following properties. The edges are grouped by stroke, that is, for every stroke s there should exist integers ℓ_s and r_s such that edge i is on stroke s if and only if $\ell_s \leq i \leq r_s$. Furthermore, on each stroke, the edges should be numbered in the same order as they occur on the stroke.

Let E_i be the set of edges numbered at most i. We define the value of $F_{\text{include}}(i, K)$ as the maximum score of a subset of E_i that:

1. includes edge i,
2. has cardinality exactly K,
3. satisfies the stroke constraints.

The definition of $F_{\text{exclude}}(i, K)$ is the same, except condition 1 says: does *not* include edge i. For notational convenience, let $F_*(i, K) \stackrel{def}{=} \max\{ F_{\text{include}}(i, K),\ F_{\text{exclude}}(i, K) \}$. The optimum is then given by $F_*(n, k)$.

Lastly, we introduce a 'previous stroke' function $\pi(\cdot)$. For an edge, it gives the highest-numbered edge in the preceding stroke. That is, with S as the set of strokes, $\pi(i) = \max\{r_s \mid s \in S \wedge r_s < i\}$. Note that this can simply be precomputed in linear time and space, and later queried in constant time.

Lemma 2. *If $K = 0$, $F_{\text{include}}(i, K) = F_{\text{exclude}}(i, K) = 0$. Otherwise, let $1 \leq i \leq n$ and $1 \leq K \leq i$. Then*

$$F_{\text{exclude}}(i, K) = F_*(i - 1, K) \tag{1}$$

$$F_{\text{include}}(i, K) = w_i + \max \begin{cases} F_{\text{include}}(\ i - 1,\ K - 1\ ) \\ F_*(\ \pi(i),\ K - 1\ ) \end{cases} \tag{2}$$

Proof. If $K = 0$, then we take no edges and get no score at all. Therefore, the value is 0.

The value of $F_{\text{exclude}}(i, K)$ considers only the edges in E_i and, in particular, does not use edge i. Then effectively we are considering only the edges in E_{i-1}, without restriction on whether we take edge $i - 1$. This is stated in (1).

For $F_{\text{include}}(i, k)$ we consider the edges in E_i and, in particular, do include edge i. In any case we add the score of edge i. We make a case distinction based on whether we also select edge $i-1$. If we do, then we have $F_{\text{include}}(i-1, K-1)$. If we do not, then we need to worry about the stroke constraint: we should not take any other edges in this stroke either. Then we are looking to use only the edges in E_{i-1} that don't belong to the current stroke. The set of those edges is, by definition, $E_{\pi(i)}$. This gives (2). □

Theorem 3. *The problem* MaxWeight-Strokes *can be solved in $\mathcal{O}(\,nk\,)$ time and $\mathcal{O}(\,k\,)$ working space, where k is the requested amount of edges.*

Proof sketch. Use dynamic programming on Equations (1) and (2). There is no need to evaluate dynamic-programming states with K larger than requested. To achieve the space bound, use space-saving dynamic programming (see for example [7, 10]). □

Theorem 4. *The problem* MinEdges-Strokes *can be solved in $\mathcal{O}(\,nk\,)$ time and $\mathcal{O}(\,k\,)$ working space, where k is the amount of edges in the optimum.*

Proof sketch. Use the algorithm from Theorem 3 and use doubling search on K. □

3. STABILITY

We now turn to time-dependent road segment selection. We assume that the weight of an edge may change over time and our goal is to adjust the selection of the segments accordingly. We assume time proceeds in unit steps within an interval $[0, T]$.

A naive approach is to apply the algorithms from the previous section at each time $t = 0, \ldots, T$ independently. The drawback of this approach is that it may lead to highly unstable selections that undergo substantial changes at every time step even if the changes of the edge weights are relatively small. To obtain selections that are easy to grasp by humans we would like to have a solution that does not only select good segments at each individual time step but is also relatively stable over time. We achieve this by penalizing the changes in the selection.

In this section, we consider a time-dependent extension of MaxWeight. Let i be an edge. At every time step

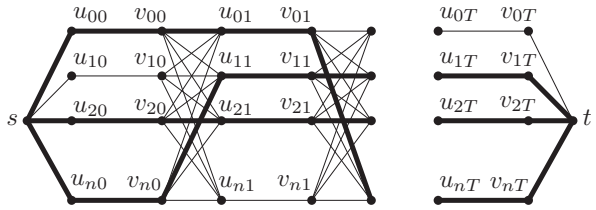


Figure 4: Layered flow network for segment set $E = \{0, \ldots, n\}$. All edges are directed from left to right and have unit capacity. Edges incident on s and t and edges $(v_{it}, u_{i,t+1})$ have cost zero. Edges $(v_{it}, u_{j,t+1})$ with $i \neq j$ have cost p. Edges (u_{it}, v_{it}) have cost $-w_{it}$. The integral s-t flow marked by the thick edges can be seen as a collection of k edge-disjoint s-t paths.

$t = 0, \ldots, T$, we are given a weight w_{it}. Moreover, we are given a positive integer k specifying the number of edges to be selected per time step. Finally, we fix a penalty p for changing the selection: whenever an edge is switched on or off from one time step to the next, we incur a penalty.

Let S_t be some edge selection of cardinality k at time step $t = 0, \ldots, T$. The *switching cost* $c(S_t, S_{t+1})$ at time step t is then given by $p \cdot |S_{t+1} \setminus S_t|$. Intuitively, we penalize each edge that changes at t by the amount p. Note that it is crucial here that $|S_t| = |S_{t+1}| = k$.

MaxWeight-Time

Instance: Set E of edges. Weight w_{it} for each edge $i \in E$ and each $t = 0, \ldots, T$. Numbers k, T and p.

Question: Set $S_t \subseteq E$ of cardinality k for each $t = 0, \ldots, T$ such that the quantity $\sum_{t=0}^{T} \sum_{i \in S_t} w_{it} - \sum_{t=0}^{T-1} p \cdot |S_{t+1} \setminus S_t|$ is maximized.

We now show that MaxWeight-Time can be solved in polynomial time by modeling it as a min-cost flow problem in a directed layered network N. See Figure 4 for an illustration.

The directed network N contains a source s and a sink t. For each segment i and each time $t = 0, \ldots, T$ we introduce nodes u_{it}, v_{it}. Now we specify the edge set of N. All edges have unit capacity. For each $t = 0, \ldots, T$ and each $i \in E$ we introduce an edge (u_{it}, v_{it}) of cost $-w_{it}$. Sending one unit of flow along this edge models the selection of i at time t. We use negated edge cost $-w_{it}$ because we want to *minimize* the total cost (in contrast to the weight maximization in MaxWeight-Time). For each $\{i, j\} \in E$ and each $t = 1, \ldots, T-1$ we moreover introduce an edge $(v_{it}, u_{j,t+1})$. If $i = j$ then the cost of this edge is zero. This models the case where segment i is selected at t and remains so at $t+1$. If $i \neq j$ then the cost of this edge is p which models the case where segment i is replaced with segment j at the transition from time t to $t+1$. Finally, we introduce a zero-cost edge (s, u_{i0}) for each $i \in E$. Similarly, we introduce a zero-cost edge (v_{iT}, t) for each $i \in E$. This completes the description of the network N.

Now we compute an s-t flow of value k and minimum cost in the network N. It is not hard to see that a solution exists if $k \leq |E|$. Since all edges have unit capacity there is an optimum solution in which every edge carries either zero or one unit of flow. Such an integral solution can be computed in polynomial time.

We now argue that this solution can be transformed into an optimum solution to MaxWeight-Time. To this end consider an arbitrary optimum solution to MaxWeight-Time of weight OPT. We now construct an s-t flow for N of value k and cost $-\mathrm{OPT}$. For every $t = 0, \ldots, T$ and every $i \in S_t$ we send one unit of flow along edge (u_{it}, v_{it}). This partial flow incurs a total cost $-\sum_{t=0}^{T} \sum_{i \in S_t} w_{it}$. Note that this flow is not a feasible solution as nodes u_{it}, v_{it} may not conserve flow. Now consider some $t = 0, \ldots, T-1$. For each segment $i \in S_t \cap S_{t+1}$ we send one unit of flow along edge $(v_{it}, u_{i,t+1})$ which incurs zero cost. Find an arbitrary bijection π from $S_t \setminus S_{t+1}$ onto $S_{t+1} \setminus S_t$ (which exists since $|S_t| = |S_{t+1}|$). We send one unit of flow along edge $(v_{it}, u_{\pi(i),t+1})$ for each $i \in S_t \setminus S_{t+1}$. This incurs a cost of $p \cdot |S_{t+1} \setminus S_t|$. The flow constructed so far conserves flow except for possibly the nodes u_{i0}, v_{iT} where $i \in E$ and the source and the sink. To obtain an overall feasible s-t flow we send one unit of flow from s to each node u_{i0} where $i \in S_0$ and from each node v_{jT} to t for every $j \in S_T$. This incurs no additional cost. The flow has value k as $|S_0| = k$. The total cost of this flow is precisely $-\mathrm{OPT}$.

The above implies that our algorithm will find an integral flow of some cost $C \leq -\mathrm{OPT}$ for N. We now describe how to find a solution of weight at least $-C \geq \mathrm{OPT}$ to MaxWeight-Time. To this end, we let S_t (where $t = 0, \ldots, T$) be the set of $i \in E$ where edge (u_{it}, v_{it}) carries one unit of flow. Note that in fact $|S_t| = k$ since the flow value is k and the graph is layered. Moreover, the quantity $\sum_{t=0}^{T} \sum_{i \in S_t} w_{it}$ is exactly the negated flow costs induced by edges of the form (u_{it}, v_{it}). Regarding the switching cost consider some fixed $t = 0, \ldots, T-1$. The crucial observation is that the total flow cost along edges of the form $(v_{it}, u_{j,t+1})$ is at least $p \cdot |S_{t+1} \setminus S_t|$. Hence our solution has weight at least $-C$ which completes the argument.

It can be noted that for this algorithm, the switching penalty p does not have to be a global constant: it can in fact depend on which edge switches to which edge.

4. STROKES AND STABILITY

In the previous sections we have separately handled stroke constraints and stability, but not both at the same time. The complexity of this natural combination remains open. Here we discuss hardness for two slight generalizations.

Theorem 5. *Edge selection with strokes and stability is $\mathcal{NP}$-hard if the strokes can vary over time, even when limited to 4 time steps.*

Proof. We show hardness by reducing from Maximum Weight Independent Set. Consider an independent-set instance $G = (V, E)$ and a weight w_v for each node v. We construct a corresponding MaxWeight-Strokes-Time instance with three main properties (see also Figure 5). First, for each node $v \in V$ our constructed instance has an edge e_v with weight w_v at all time steps. All further edges always have weight 0. Secondly, by making the switching cost p sufficiently large ($p > \sum_{v \in V} w_v$), we ensure that in an optimum solution the set of selected edges remains the same over all time steps. Lastly, we ensure that for any two nodes $\{u, v\} \in E$ at most one of their corresponding edges e_u or e_v is selected. We enforce this using strokes. For each such pair of edges $\{e_u, e_v\}$, we add $|V| - 1$ edges (of weight 0)

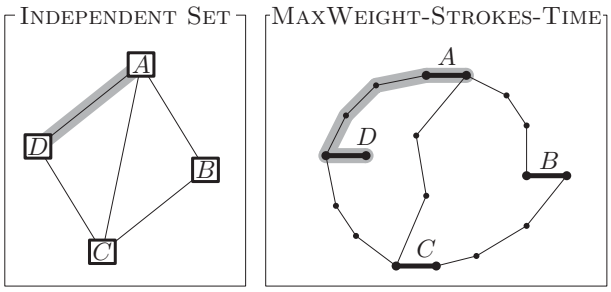


Figure 5: In the $\mathcal{NP}$-hardness proof, each node is replaced by an edge of the same weight, and each edge is replaced by a path of weight zero. For each edge there is a time step with a stroke corresponding to it. Here, the gray-shaded stroke corresponds to the edge AD. Since in this case we will set $k = 4$, no stroke can be selected in its entirety.

that form a path of length $|V| + 1$ from e_u to e_v. The constructed instance has a time step where we have a stroke equal to this path (and no other strokes). Thus, if e_u and e_v are selected at this time, then also all the edges on the path must be selected. By limiting the requested number of edges k to $|V|$, at most one of e_u or e_v, can be selected: there are too many edges between them in the stroke. By these three properties, any set of selected edges corresponds to an independent set of nodes of the same weight. Conversely, any edge set corresponding to a maximum weight independent set does not violate any stroke constraints and, hence, is selectable. This completes the reduction.

Note that the number of time steps now equals $|E|$. We can do better by putting multiple strokes into the same time step. This is possible as long as the involved strokes are edge disjoint, which is the case if the corresponding edges are a matching in G. By reducing from maximum-degree-3 independent set (also $\mathcal{NP}$-hard), it suffices to use 4 time steps: a max-degree 3 graph has a 4-edge-coloring. Then a time step per color suffices. □

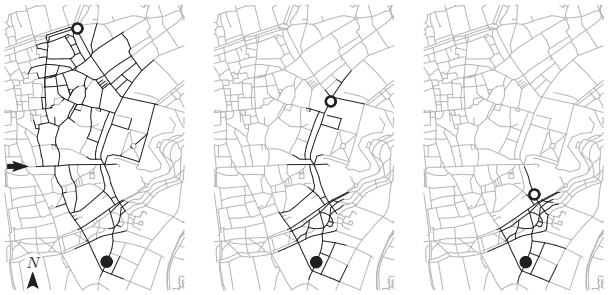
It seems reasonable that strokes should be the same at every time step. In that case, edge selection with strokes and stability is $\mathcal{NP}$-hard if the switching costs can depend on which edge switches to which edge. (The proof is similar and omitted for space.) The complexity of edge selection with strokes and stability remains open when the strokes are constant and the switching costs are uniform.

5. EXPERIMENTS

We have implemented the algorithm for stroke constraints. The following experiments have been run on a desktop PC with an Intel® Core™ i5-2400 CPU at 3.10 Ghz.

In our examples we consider the German city of Würzburg. We use a road network of the city and its surroundings, consisting of 3786 nodes and 4987 road segments in 677 strokes. This network is a crop of a much larger OpenStreetMap data set of Lower Franconia (approximately 10^5 nodes) that is available online.[1] We have calculated weights using the random traveler model of Van Dijk and Haunert [3]. The figures in this paper have been clipped to show only the

[1] `download.geofabrik.de/osm/`



(a) $t = 0$ (b) $t = 16$ (c) $t = 32$

Figure 6: Several edge selections following a user (○) travelling to a destination (●). Edges are selected using the dynamic program to select 90% of the weight. Note the horizontal stroke indicated with an arrow in (a): its middle part does not contain much weight, but a stroke constraint ensures the connecting edges are selected. This improves the readability of the selected map.

area where edges are actually selected. See Figure 6 for an example.

We have grouped road segments into strokes as follows. Consider a line segment e and one of its endpoints u. Let $C_u(e)$ be the line segment $f \neq e$ that is incident to u and minimizes the change of direction when traveling from e to f. (If e is the only line segment incident to u, we set $C_u(e) = nil$.) Two line segments e and f that share a node u are put into the same stroke if both $C_u(e) = f$ and $C_u(f) = e$, and the change of direction from e to f is smaller than a threshold θ. We have used $\theta = 30°$.

Now we evaluate the stroke constraints using 1000 random source-destination pairs in our map. As noted, the greedy algorithm may violate stroke constraints. Figure 2 shows an example where this happens. In our 1000 runs of the experiment, the greedy algorithm violated stroke constraints in 82% of the instances and in those cases violated 3.9 stroke constraints on average. The dynamic program guarantees a solution that does not violate stroke constraints at all.

Both the greedy algorithm and the dynamic program might select a disconnected set of edges. (As argued in Section 2, we do not require connectedness.) Of the selected edges, consider the connected component that contains the most weight. We have found that it is not uncommon for either algorithm to have some disconnected edges: 43% of instances with greedy and 29% with strokes constraints. However, there are typically not many such disconnected edges and they represent only a negligible amount of score (0.16% with greedy, 0.09% with strokes). We suggest dropping these edges.

Lastly, these experiments show that the dynamic program runs very quickly (though of course much slower than the greedy algorithm, which runs almost instantly): 0.4 ms on average for the greedy algorithm, versus 58 ms for the dynamic program. On very large maps, the space-saving version of the dynamic program becomes relevant. For simplicity, our implementation uses $\mathcal{O}(nk)$ space instead of $\mathcal{O}(k)$. This is no problem at all for any of the experiments presented in this paper. When running the algorithm on the entire Lower Franconia data set of 10^5 nodes, however, mem-

ory usage easily gets out of hand if k is large. Implementing the space-saving version would solve this.

Preliminary experimentation with the stability optimization suggests that this approach can provide significant improvement. Using the penalty p, we can influence the trade-off between the number of switches and the weight. With the same data set as before, and weights calculated for a user travelling across the city [3], we generally observe that with fixed k, and at the cost of losing 10% of the weight, the number of switches can be greatly reduced. In a typical instance, for example, 942 switches could be reduced to 150 while still getting 90% of the weight.

6. CONCLUSIONS AND FUTURE WORK

We have presented polynomial-time algorithms to select a preferably small number of road segments of a preferably large total weight. We can either prescribe the number of road segments in the selection or require a minimum total weight – and optimize the other criterion. A dynamic-programming approach allows us to ensure that from each stroke in the road network at most one connected path is selected. On the other hand, given multiple time frames in which the road segments have different weights, we can use a flow-based algorithm to select a given number of segments in each time frame such that a combined cost function is optimized. This combined cost function integrates our basic objective of maximizing the total weight of the selection and the objective of avoiding excessive changes between subsequent selections. With this we aim to increase the stability of the selection over time.

Ultimately, we are interested in a maximum-weight solution that is stable *and* satisfies the stroke constraint. Though it is still open whether such a solution can be found in polynomial time, our $\mathcal{NP}$-hardness proof for a slightly more general variant of the problem leads us to be rather pessimistic. Therefore, we will also consider heuristic methods. An obvious idea is to use the stability optimization to compute an initial solution and to add some road segments to satisfy the stroke constraint. In order to assess whether this heuristic yields solutions of high quality, however, additional experiments are needed.

Our model for stability optimization assumes that the weights of the road segments are known in advance for all time frames. This requirement is realistic if the user follows a precomputed route, but not so if the user navigates freely in the environment. Therefore, we plan to relax that requirement and to develop online algorithms for stable selections of road segments.

Acknowledgments

This research was supported by grant Ha 5451/3-1: *Algorithms for Interactive Variable-Scale Maps* of the German Research Foundation (DFG).

7. REFERENCES

[1] K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry*, 43(3):312–328, 2010.

[2] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[3] T. C. v. Dijk and J.-H. Haunert. A probabilistic model for road selection in mobile maps. In *Proc. 12th Internat. Sympos. Web and Wireless Geographical Information Systems*, volume 7820 of *Lecture Notes in Computer Science*, pages 214–222. Springer-Verlag, Berlin, Germany, 2013.

[4] M. Fischetti, H. W. Hamacher, K. Jørnsten, and F. Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24(1):11–21, 1994.

[5] A. Gemsa, B. Niedermann, and M. Nöllenburg. Trajectory-based dynamic map labeling. In *Proc. 29th European Workshop Computational Geometry (EuroCG'13)*, 2013.

[6] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In *Proc. 12th Internat. Sympos. Algorithms and Data Structures (WADS'11)*, volume 6844 of *Lecture Notes in Computer Science*, pages 451–462. Springer-Verlag, Berlin, Germany, 2011.

[7] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[8] B. Jiang. Ranking spaces for predicting human movement in an urban environment. *International Journal of Geographical Information Science*, 23(7):823–837, 2009.

[9] B. Jiang and C. Claramunt. A structural approach to the model generalization of an urban street network. *Geoinformatica*, 8(2):157–171, 2004.

[10] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[11] J. Kopf, M. Agrawala, D. Bargeron, D. Salesin, and M. Cohen. Automatic generation of destination maps. *ACM Transactions on Graphics*, 29(6):158:1–158:12, 2010.

[12] W. A. Mackaness and M. K. Beard. Use of graph theory to support map generalization. *Cartography and Geographic Information Systems*, 20:210–221, 1993.

[13] F. Schmid, C. Kuntzsch, S. Winter, A. Kazerani, and B. Preisig. Situated local and global orientation in mobile you-are-here maps. In *Proc. 12th Internat. Conf. Human Computer Interaction with Mobile Devices and Services*, MobileHCI '10, pages 83–92. ACM, 2010.

[14] R. C. Thomson and R. Brooks. Exploiting perceptual grouping for map analysis, understanding and generalization: The case of road and river networks. In *Selected Papers from the Fourth Internat. Workshop Graphics Recognition Algorithms and Applications*, GREC '01, pages 148–157. Springer-Verlag, Berlin, Germany, 2002.

[15] B. Yang, X. Luan, and Q. Li. Generating hierarchical strokes from urban street networks based on spatial pattern recognition. *International Journal of Geographical Information Science*, 25(12):2025–2050, 2011.

[16] Q. Zhang. Road network generalization based on connection analysis. In *Developments in Spatial Data Handling – Proc. 11th Internat. Symposium Spatial Data Handling*, pages 343–353, 2005.