

A faster algorithm for matching planar maps under the weak Fréchet distance

Daniel Chen Leonidas J. Guibas Qixing Huang Jian Sun *

December 1, 2008

Abstract

We consider the problem of matching a polygonal curve of complexity p with an arbitrary curve on an embedded planar graph of complexity q . With the large amount of GPS and location trace data that have become available in recent years, algorithms for such problems have become increasingly important. Direct applications include matching a GPS trace to a given roadmap and indirect applications including building and querying databases of polygonal curves. As problem sizes increase, improvements in asymptotic runtime may prove to be useful and we present an algorithm which solves the problem with respect to the weak Fréchet distance in $O(pq)$ time, which is an asymptotic improvement. In the process of doing so, we also provide a more intuitive and practical approach for the recursive division procedure used in Henzinger et al.'s shortest path algorithm for planar graphs. Unlike previous approaches to minimize the asymptotic complexity of calculating the Fréchet distance, our method does not involve parametric search, and hence is also more practical to implement.

*Computer Science Department, Stanford University, Palo Alto CA 94305.

1 Introduction

Consider a given polygonal curve P and an embedded planar graph G with line segment edges. We want to match P with a connected curve C on G where C is closest to P under some dissimilarity measure. This problem has been considered before by [3] in the context of map-matching vehicle tracking data and given the large amount of GPS data gathered in recent years, this seems to be a particularly interesting application. However, this problem has many other less direct applications: [1] mentions that it may be useful in an incremental construction of road networks from GPS traces and also extends it to matching geometric patterns given as embedded planar graphs. Our particular motivation for this problem comes from the design of databases to organize polygonal curves. We believe that map-matching may be an useful subroutine in both the construction and querying of such a database.

The particular dissimilarity measure we consider is the weak Fréchet distance. Given two polygonal curves P and Q , the weak Fréchet distance can be thought of as the length of the shortest leash required for a man to walk from the start of P to the end of P while staying on P and for his dog to walk from the start of Q to the end of Q while staying on Q . It is a variant of the Fréchet distance, which is more restrictive in that the man and the dog cannot walk backwards on P and Q . [3] provide experimental evidence which shows that matching with respect to the Fréchet distance and with respect to the weak Fréchet distance provide identical results for matching vehicle tracking data on road networks.

Map-matching with respect to the Fréchet distance has been studied by [1], who present an $O(pq \log pq \log q)$ algorithm where p is the number of edges of the given polygonal curve and q is the number of edges of the embedded planar graph. The case for the weak Fréchet distance has been studied before by [1], who give an $O(pq \log pq)$ algorithm. Our result is to improve this slightly to $O(pq)$; there may be applications both in theory and in practice where this reduction in asymptotic runtime would be useful. Furthermore, unlike previous approaches to minimize the asymptotic complexity of calculating the Fréchet distance, our method does not involve parametric search, and hence is also more practical to implement. Our algorithm also yields an $O(pq)$ algorithm for computing the weak Fréchet distance between polygonal curves where one curve is of complexity p and the other is of complexity q , which improves the $O(pq \log pq)$ result given by [2].

In the process of doing so, we also provided a more intuitive approach for the recursive division procedure in [7] that uses the the concept of an H -partition as used in [9]. However, as directly applied to planar graphs, their procedure has a constant of 2^{30} which we replace with the more practical constant 26. We also show a class of graphs called stacks, which permit linear time shortest path algorithms. We believe that such graphs are typical in many dynamic programming problems. Hence, our results for asymptotic improvement may also be applied to such problems.

2 Problem Definition and Main Result

The input to our problem are a polygonal curve P represented as p connected segments $\overline{u_0u_1}, \overline{u_1u_2}, \dots, \overline{u_{p-1}u_p}$ where u_0, u_1, \dots, u_p are points in \mathbb{R}^d , and a planar graph $G = (V, E)$ embedded in \mathbb{R}^d with line segment edges. To keep the discussion simple, we sometimes abuse notation and refer to embedded vertex points as the vertices themselves and line segments as edges. We define the weak Fréchet distance as follows:

Definition 2.1 (weak Fréchet distance). *For two curves given as continuous maps $P, Q : [0, 1] \rightarrow \mathbb{R}^d$, the weak Fréchet distance*

$$d_{wF}(P, Q) = \inf_{\substack{\alpha: [0,1] \rightarrow [0,1] \\ \beta: [0,1] \rightarrow [0,1]}} \max_{t \in [0,1]} \|P(\alpha(t)) - Q(\beta(t))\|$$

where α, β range over all continuous function with $\alpha(0) = 0, \alpha(1) = 1, \beta(0) = 0$ and $\beta(1) = 1$.

An intuitive way to look at the weak Fréchet distance is the length a shortest leash necessary for a man to walk on one curve and his dog to walk on the other curve from the beginning of the curves to the end, while allowing them to backtrack. The problem we wish to solve then, is to find a curve C on G minimizing the weak Fréchet distance $d_{wF}(P, C)$. We wish to output both C and $d_{wF}(P, C)$. Note that there may be many such curves C , but any one of them would suffice. Our main result is as follows:

Theorem 2.2 (Main theorem). *Given a polygonal planar graph G and a polygonal curve P embedded in \mathbb{R}^d , there is an algorithm of time complexity $O(pq)$ for computing a curve C in G that is closest to P under the weak Fréchet distance where p and q are the number of edges in P and H respectively.*

We remark that the curve we find on the graph also satisfies some simplifying properties allowing it to be output in time $O(pq)$. In order to describe them we first introduce the notion of a *fractional edge*.

Definition 2.3 (Fractional edge). *Let an edge \overline{uv} in our embedded planar graph correspond to a line segment. Then, the fractional edge $\overline{uv}_\alpha^\beta$ would be the line segment from $\alpha u + (1 - \alpha)v$ to $\beta u + (1 - \beta)v$.*

We note that under the weak Fréchet distance, it is sufficient to consider curves that consist of fractional edges joined at vertices because any connected curve on the embedded planar graph is of weak Fréchet distance zero from such a curve. Our algorithm outputs such a curve with at most $O(pq)$ fractional edges.

The method we use to find C and $d_{wF}(P, C)$ in $O(pq)$ time is to reduce the problem to single source bottleneck shortest paths on a certain graph and use the solution to extract C and $d_{wF}(P, C)$. The reduction takes time $O(pq)$ and the bottleneck shortest paths computation on the graph also takes time $O(pq)$. We explain in detail the reduction in Section 4 and the bottleneck shortest paths computation in the remaining sections.

3 Reduction to Bottleneck Shortest Paths

In order to explain the reduction, we refer to the notions of free space and free space diagram introduced in [2] for computation of Fréchet distance between two polygonal curves. Subsequently, the free space diagram was extended in [1] for planar map matching.

3.1 Simplified example for weak Fréchet distance between polygonal curves

To provide intuition, we begin with a simplified case of calculating weak Fréchet distance between two polygonal curves P and Q with vertices u_0, u_1, \dots, u_p and vertices v_0, v_1, \dots, v_q respectively. We equip P with a parameterization $\pi : [0, p] \rightarrow \mathbb{R}^d$ such that $\pi(i) = u_i$ and for $0 < \alpha < 1$, $\pi(i + \alpha) = (1 - \alpha)u_i + \alpha u_{i+1}$ and Q with a parameterization $\kappa : [0, q] \rightarrow \mathbb{R}^d$ such that $\kappa(i) = v_i$ and for $0 < \alpha < 1$, $\kappa(i + \alpha) = (1 - \alpha)v_i + \alpha v_{i+1}$.

Then, given $\varepsilon > 0$ the *free space* between P and Q is defined as $F_\varepsilon(P, Q) := \{(s, t) \in [0, p] \times [0, q] \mid \|\pi(s) - \kappa(t)\| \leq \varepsilon\}$. Call the partition of $[0, p] \times [0, q]$ into white regions belonging and black regions not belonging to $F_\varepsilon(P, G)$ the *free space diagram* $FD_\varepsilon(P, G)$, see Figure 1(b). Then, we have the following obvious lemma:

Lemma 3.1. *There is a continuous curve from the lower left corner of $FD_\varepsilon(P, Q)$ to the upper right corner of $FD_\varepsilon(P, Q)$ only traversing through the white regions if and only if $d_{wF}(P, Q) \leq \varepsilon$.*

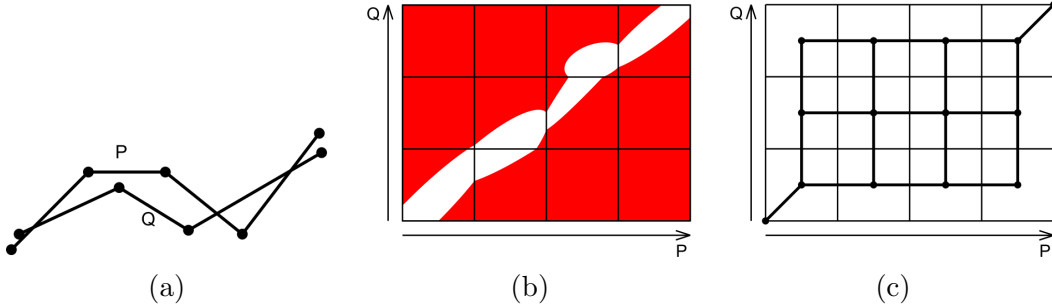


Figure 1: free space diagram and dual graph.

The idea is to find the smallest ε such that there exists such a curve in the free space from the lower left corner of $FD_\varepsilon(P, Q)$ to the upper right corner of $FD_\varepsilon(P, Q)$. One way to do this is to run parametric search on ε . However, there are $O(pq)$ parameters, and hence it is reasonable that parametric search adds more than constant overhead. Therefore, we instead construct a graph such that the bottleneck shortest distance between two vertices is equal to ε .

To do this, we first observe that the free space diagram consists of pq cells connected as a grid. Moreover, the free space in each cell is convex. This is because the free space in each cell is the free space for two line segments and some calculation shows that it is the intersection of the preimage of a circle under an affine map with a square. Hence if $F_\varepsilon(f, g)$ has a non empty overlap with any two boundary sides of a cell, then there exists a curve within the cell in $F_\varepsilon(f, g)$ from one boundary side to the other. Thus, we construct a graph where the vertices correspond to the cells and edges correspond to the junction of cells. Let each edge weight the minimum ε such that $F_\varepsilon(f, g)$ has non empty intersection with the corresponding junction. This ε is easy to calculate because it is simply the distance between a point and a line segment. We also add a source vertex, a target vertex, an edge with weight $\|u_1 - v_1\|$ connecting the source to the vertex corresponding to the lower left cell and an edge with weight $\|u_p - v_q\|$ connecting the vertex corresponding to the upper right cell with the target, see Figure 1(c). We claim that the weak Fréchet distance between P and Q is the bottleneck shortest distance between the source and the target. Obviously the constructed weighted graph is planar and the bottleneck shortest distance can be solved in linear time [7, 9]. Hence we obtain an algorithm to compute the weak Fréchet distance between P and Q in $O(pq)$.

3.2 Constructing the graph

We extend this idea to the *free space surface* which was introduced in [1] for planar map matching. The free space surface $FS_\varepsilon(P, G)$ for a polygonal curve P and a graph G is also composed of connected cells where each cell corresponding to the free space diagram of two segments: one from the curve P and the other corresponding to an edge of G see Figure 2(b). We have the following obvious lemma:

Lemma 3.2. *Let P be a polygonal curves with vertices $u_1, u_2 \dots u_p$. Then, there is a continuous curve in the free space $FS_\varepsilon(P, G)$ from the boundary corresponding to u_1 to the boundary corresponding to u_p only traversing through white regions if and only if there exists a curve C on G with $d_{wF}(P, C) \leq \varepsilon$.*

Like in the case of Fréchet distance between curves, we construct a graph on the free space surface. First, we note that the edges of P be $\overline{u_0u_1}, \overline{u_1u_2}, \dots, \overline{u_{p-1}u_p}$. Then, we arbitrarily label the edges of G as e_1, e_2, \dots, e_q . Then, the free space surface consists of cells $C_{i,j}$ where $C_{i,j}$ is the free space between $\overline{u_{i-1}u_i}$ and e_j . It is easy to see that $C_{i,j}$ shares a junction with $C_{i+1,j}$ for all $1 \leq i \leq p-1$ and $C_{i,j}$ shares a junction $C_{i,k}$ if and only if e_j is adjacent to e_k . Like before, we add a vertex $v_{i,j}$ for each cell

$C_{i,j}$ and would like to connect vertices whose corresponding cells share a junction with an edge whose weight w is the minimum ε such that $FS_\varepsilon(P, G)$ has non-empty intersection with the junction. However, in this case, junctions may be shared by several cells. This may be the case when we are considering the junction between a cell $C_{i,j}$ and a cell $C_{i,k}$. Therefore, for all such junctions, we add a new vertex and connect the new vertex to the vertices corresponding to neighboring cells with an edge of weight w . If a junction is only shared by two cells, we simply connect their corresponding vertices together with an edge of weight w , see Figure 2(c).

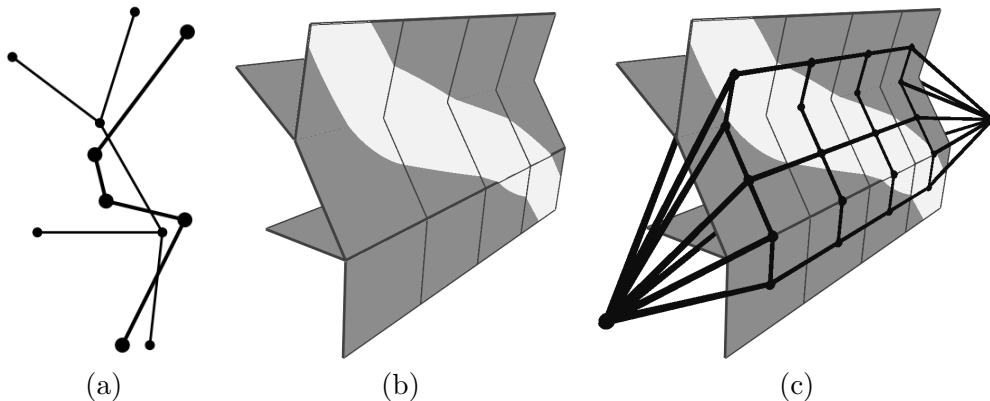


Figure 2: free space surface and dual graph.

Now, recall we want our bottleneck shortest path to correspond to a path from the boundary corresponding to u_1 to the boundary corresponding to u_p . Then, it is easy to see that we simply need to add a source vertex s , and edges $(s, v_{1,j})$ with weight being the distance from u_1 to e_j and a target vertex t with edges $(v_{p,j}, t)$ with weight being the distance from u_p to e_j , as drawn in Figure 2(c). Then, it is clear that the bottleneck shortest distance from s to t is the smallest ε for which there is a continuous curve in the free space $FS_\varepsilon(P, G)$ from the boundary corresponding to u_1 to the boundary corresponding to u_p only traversing through white regions.

3.3 Extracting the solution

Now suppose we have a bottleneck shortest path from s to t . Then, it passes through a sequence of vertices $v_{i_1, j_1}, v_{i_2, j_2}, \dots, v_{i_k, j_k}$ that correspond to cells $C_{i_1, j_1}, C_{i_2, j_2}, \dots, C_{i_k, j_k}$ in the free space surface. Then, we output the polygonal curve which is a connected sequence of fractional edges $\overline{uv}_\alpha^\beta$. Then the bottleneck shortest path corresponds to a curve D in $FS_\varepsilon(P, G)$ that passes through those cells where ε is the bottleneck shortest distance between s and t . Moreover, we can choose a D that is a line segment within each cell. Then the curve C on G that corresponds to D is obtained by the image of D obtained by flattening the free space surface into the planar graph G . It is then obvious that C is a concatenation of at most $O(pq)$ fractional edges because D passes through at most $O(pq)$ cells and the curve's path through each cell gets flattened into a fractional edge. Then, we argue that C is a curve on G that is closest to P under the weak Fréchet distance.

Theorem 3.3. *The polygonal curve C output by the above algorithm is a curve in G that is the closest to P under the weak Fréchet distance.*

Proof. The above discussion shows that C is a curve on G that is within weak Fréchet distance ε to P where ε is the bottleneck shortest distance between s and t in our graph. We argue that there is no curve

C' on G that is closer. For the purpose of contradiction, suppose there was. Then C' is of weak Fréchet distance $\epsilon' < \epsilon$ away from P . Therefore, C' corresponds to a curve from the boundary of free space surface corresponding to u_1 to the boundary of the free space surface corresponding to u_p while staying in the white region defined by $FS_{\epsilon'}(P, G)$. Thus, there is a path in the graph from s to t of bottleneck shortest distance ϵ' which contradicts our assumption that ϵ was the bottleneck shortest distance. \square

4 Computing bottleneck shortest paths in $O(pq)$ time

In the previous section, we reduced the map-matching problem to finding a bottleneck shortest path in a certain graph of $O(pq)$ vertices and edges. If we use Dijkstra's with Fibonacci heaps, we only obtain an $O(pq \log pq)$ algorithm. However, we can modify the graph so that it belongs to a certain class of graphs we call *stacks*. Then, we show that single source bottleneck shortest paths can be solved in linear time on stacks and hence we obtain a $O(pq)$ algorithm for our problem.

4.1 Stacks

We define stacks as follows:

Definition 4.1 ((p, q) -stack). *Let a graph G be a (p, q) -stack if it is the Cartesian product of the path P of p vertices with a planar graph H of q vertices and a labeling of its vertices as follows: given a labeling (w_1, w_2, \dots, w_p) of the vertices of P from left to right, and an arbitrary labeling (u_1, u_2, \dots, u_q) of H , we let vertex $v_{i,j}$ of G be (w_i, u_j) . We say that G is a stack if it is a (p, q) -stack for some p, q .*

As presented, our graph is not yet a stack, but we easily modify it to be a stack. Looking at Figure 2(c), we observe that we have a subgraph of a stack that is the Cartesian product of P_q with a planar graph H if we remove the source and target vertex. Then, we reintroduce the source and target by expanding them into layers $\{v_{0,j}\}_j$ and $\{v_{q+1,j}\}_j$ that are copies of H . For each edge $(s, v_{1,j})$ out of the original source, we have an edge $(v_{0,j}, v_{1,j})$ and we assign all edges within the layer $\{v_{0,j}\}_j$ to have zero weight. Similarly, we connect and assign edge weights to $\{v_{q+1,j}\}_j$. We now have a dense subgraph of a stack and it is clear that the bottleneck shortest paths from an arbitrary vertex in the source layer to an arbitrary vertex in the target layer corresponds to bottleneck shortest paths from the source to the target in our original graph. Moreover, we can convert it into a stack by adding edges of infinite weight without affecting the bottleneck shortest paths problem. It is also clear we do not add more than $O(pq)$ edges and vertices under this transformation.

We also note the following separation theorem for stacks that would prove useful later on:

Theorem 4.2. *Let G be a (p, q) -stack of $n = pq$ vertices and U be a subset of $V(G)$. Then, the vertices of G can be partitioned into three sets A, B, C such that there is no edge between a vertex in A and a vertex in B , $|A \cap U|, |B \cap U| \leq 2|U|/3$, and $|C| \leq 2n^{2/3}$. Furthermore, A, B, C are stacks and this partition can be computed in $O(n)$ time.*

Proof. If $q \leq 8p^2$, we let $U(k) = |\{v_{i,j}\}_{i \leq k,j} \cap U|$. We let $U(0) = -1$. Then, by counting the elements of U while traversing G from left to right, we find k such that $U(k-1) < |U|/2 \leq U(k)$. Then, we set $A = \{v_{i,j}\}_{i < k,j}$, $B = \{v_{i,j}\}_{i > k,j}$ and $C = \{v_{k,j}\}_j$. Clearly, A and B have no edges between each other. Furthermore, $|A \cap U|, |B \cap U| \leq |U|/2 \leq 2|U|/3$, $|C| = q \leq 2n^{2/3}$, A, B, C are stacks and this partitioning can be done in linear time.

Otherwise, $q > 8p^2$ and we run Lipton-Tarjan's planar separator algorithm on the subgraph induced by $\{v_{1,j}\}_j$ with vertex costs $c(v_{1,j}) = |\{v_{i,j}\}_i \cap U|/|U|$. The planar separator algorithm will partition the subgraph into A', B', C' such that there are no edges between A' and B' , $c(A'), c(B') \leq 2/3$ and

$|C'| \leq 2\sqrt{2q}$. We then let A be $\{v_{i,j}\}_{i,v_{1,j} \in A'}$, B be $\{v_{i,j}\}_{i,v_{1,j} \in B'}$ and C be $\{v_{i,j}\}_{i,v_{1,j} \in C'}$. To see that $|A \cap U| \leq 2|U|/3$, let us suppose that $|A \cap U| > 2|U|/3$. Then, $c(A') = \sum_{v_{1,j} \in A'} c(v_{1,j}) = \sum_{v_{1,j} \in A'} |\{v_{i,j}\}_i \cap U|/|U| = |\bigcup_{v_{1,j} \in A'} (\{v_{i,j}\}_i \cap U)|/|U| = |(\bigcup_{v_{1,j} \in A'} \{v_{i,j}\}_i \cap U)|/|U| = |A \cap U|/|U| > 2/3$, violating the correctness of Lipton-Tarjan. The proof for showing $|B \cap U| \leq 2|U|/3$ is symmetric. Furthermore, we have $|C| \leq 2p\sqrt{2q} \leq 2n^{2/3}$

In both cases it is clear that A, B, C can be computed in linear time, there are no edges between A and B and A, B, C are stacks. \square

4.2 Finding single-source bottleneck shortest paths in linear time on stacks

For general graphs, there is no known algorithm to compute single-source bottleneck shortest paths in linear time. However, for the special case of planar graphs, Henzinger et al. [7] presented a linear time algorithm. Tazari and Müller-Hannemann generalized their algorithm to work for classes of graphs excluding a fixed minor, albeit super-exponential in the size of the minor. However, the following corollary shows that we cannot directly use either of these results:

Lemma 4.3. *For any l , there exists a (p, q) -stack with K_l as a minor. In particular, a (p, q) -stack need not be planar.*

Proof. Consider the Cartesian product of the path P_l with the star graph $S_{\binom{l}{2}+1}$. We label the center vertex of the star graph as u_0 and the rest of the vertices as $u_{\{i,j\}}$ for $1 \leq i < j \leq l$. Then we let the l vertices of the K_l minor be $v_{i,0}$ for $1 \leq i \leq l$. But then, there is a path $(v_{i,0}, v_{i,\{i,j\}}, v_{\dots,\{i,j\}}, v_{j,\{i,j\}}, v_{j,0})$ between $v_{i,0}$ and $v_{j,0}$ for $1 \leq i < j \leq l$ and all such paths are disjoint and therefore we have our K_l minor. \square

Hence, we are unable to directly use the results of [7] and [9]. Nevertheless, we are able to adapt the algorithm in [9] by exploiting the special structure of a stack to obtain a linear time bottleneck shortest paths algorithm. Dijkstra's requires a logarithmic overhead because the priority queue may contain a linear number of vertices. The idea of the algorithms in [7] and [9] is to decompose the graph into a recursive division satisfying certain properties and then use this division to relax the edges in a certain order. Although now edges may be relaxed more than once, the priority queues are now short, and a complicated charging scheme shows that the entire algorithm runs in linear time.

To explain the properties of this division, we first define the following:

Definition 4.4 ((r, f) -division). *A (r, f) -division of an n -vertex graph is a partition of the edges of the graph into $O(n/r)$ regions, each containing $r^{O(1)}$ vertices and each having at most $f(r)$ boundary vertices, where boundary vertices are those that are shared by more than one region.*

Definition 4.5 ((\bar{r}, f) -recursive division). *A (\bar{r}, f) -recursive division of an n -vertex graph and positive integer sequence $\bar{r} = (r_0, r_1, \dots, r_k)$ is a recursive partitioning of the graph where the entire graph is partitioned into a (r_k, f) -division and each region is then recursively partitioned into a (\bar{r}^l, f) -recursive division, where $\bar{r}^l = (r_0, r_1, \dots, r_{k-1})$*

Henzinger et al.[7] present a bottleneck shortest paths algorithm that runs in linear time given a graph with maximum degree 3 and a (\bar{r}, f) -recursive division satisfying:

$$\frac{r_i}{f(r_i)} \geq 8^i f(r_{i-1}) \log r_{i+1} \left(\sum_{j=1}^{i+1} \log r_j \right). \quad (1)$$

For the remaining part of the section, we will show how to find such a recursive division for stacks and establish the degree requirement in linear time and hence yield the following result:

Theorem 4.6. *For any (p, q) -stack, single-source shortest-paths with nonnegative edge-weights can be calculated in linear time $O(pq)$.*

4.2.1 Recursive division in linear time

To generate the appropriate recursive division in linear time, we follow the approach of Tazari and Müller-Hannemann [9] instead of Henzinger et al. [7]. The reason we do this is because the recursive division in [7] applies a procedure *FindClusters* [5] recursively on minors of a planar graph. However, FindClusters requires that the graph be of degree at most 3. Note though finally we need to transform the graph to the one with maximum degree 3 in order to obtain a linear time algorithm for computing shortest paths, our recursive division algorithm does not rely on this degree requirement. Although it is easy to address the degree requirement at the highest level, planar graphs of degree at most 3 are not closed under minors and fixing the graph to be degree 3 after each contraction might alter the structure of the recursive division. Henzinger et al. does not mention this detail in the analysis and a full analysis would be likely much more complicated. Our recursive division algorithm only modifies that in [9] at two places: *H-Partition*, which we replace with *Stack-Partition* (see Lemma 4.10) and *Divide* (see Lemma 4.11), which we modify to work with stacks. For the purpose of completeness, we include the entire recursive division algorithm in Algorithm 1.

Algorithm 1 Recursive Division

```

1: Input: A  $(p, q)$ -stack  $G$ 
2: Output: A recursive division tree  $T$  for  $G$ 
3: //Partition and contract
4: let  $G_0 := G, z_0 := 2, i := 0$ ;
5: while the number of vertices in  $G_i > \frac{n}{\log n}$  do
6:   let  $G_{i+1} := \mathbf{Stack-Partition}(G_i, z_i)$ ;
7:   let  $z_{i+1} := 14^{z_i^{1/7}}, i := i + 1$ ;
8: end while
9:  $I := i + 1$ ;
10: //divide the graphs and build recursive tree
11: let  $v_G$  be the root of  $T$ ;
12: let  $D_{I+1}$  be the trivial division of  $G_{I+1}$  consisting of a single region;
13: for  $i := I$  down to 0 do
14:   for each region  $R$  of  $D_{i+1}$  do
15:     let  $S_R$  be the boundary vertices of  $R$  in the division  $D_{i+1}$ ;
16:     let  $D_R := \mathbf{Divide}(R, S_R, z_i)$ ;
17:     for each region  $R'$  of  $D_R$  do
18:       expand  $R'$  into a region  $R''$  of  $G_i$  by expanding every vertex;
19:       assign each boundary edge to one of the regions it occurs in ;
20:       create a child  $v_{R''}$  of  $v_R$  in  $T$ ;
21:     end for
22:   end for
23:   let  $D_i$  be the decomposition of  $G_i$  consisting of the regions  $R''$ ;
24: end for //add the leaves
25: for each edge  $e$  of each region  $R$  of  $D_0$  do
26:   create a child  $v_e$  of  $v_R$  in  $T$ ;
27: end for
28: return  $T$ ;

```

Once we have the Stack-Partition procedure and the Divide procedure adapted to work for stacks, it follows from the proof of [9] that Algorithm 1 is a linear-time algorithm that finds (\bar{r}, f) -recursive division

of G satisfying inequality (1) for all r_i . Below we describe the Stack-Partition Procedure and the Divide procedure. We start by defining a connected H -partition:

Definition 4.7 (connected H -partition). *Consider a partition $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$ of $V(G)$. Let H be the graph obtained by collapsing each block P_i into a single vertex v_i and getting rid of multiple edges. Then, we say that \mathcal{P} is a connected H -partition if $(v_i, v_j) \in E(H)$ if and only if there is an edge in G between every connected component of P_i and every connected component of P_j .*

Corollary 4.8. *If we collapse the individual partitions of a connected H -partition of a planar graph G into single vertices and get rid of multiple edges, we end up with a planar graph H .*

Proof. Consider the graph F obtained from G by contracting each connected component of the partitions in a connected H -partition into a vertex. Then F is a minor of G and is thus, planar. But then H is isometric to a subgraph of F , so it is also planar. \square

The following theorem and proof is similar to that of Reed and Wood [8], except we tailor it for planar graphs specifically to achieve much smaller constants. By using specific properties of planar graphs, we can take the original constant $c_0 = 2^{l^2+l} = 1073741824$ for K_5 exclusion down to 26, which makes the algorithm more practical.

Lemma 4.9. *There is a linear-time algorithm that given a constant z and a planar graph G of $n \geq 13z$ vertices, outputs a connected H -partition of size less than n/z and where each block is of size less than $26z$.*

Proof. The algorithm works as follows: We first greedily find a H -partition where each block is of size less than $13z$ and then modify it by merging blocks while maintaining the H -partition property so that the final H -partition has less than n/z blocks and each block is of size less than $26z$.

The first step of the algorithm is to use depth first search to find connected subgraphs of size $13z$ and remove their vertices as a block. We repeat this step until there are no more connected subgraphs of size $13z$. Let A be the blocks created in this step. We note that A is non-empty. Then, we let the remaining connected components of the graph form blocks B . Now, $A \cup B$ is an H -partition because the subgraphs induced by the blocks in $A \cup B$ are connected.

Let a block be *big* if it is of size at least $13z$ and *small* otherwise. Clearly, all blocks in A are big and all blocks in B are small. We say that two distinct blocks are *neighbors* if there is an edge from a vertex of one block to a vertex of another. Then, we say the *degree* of a block b is the number of neighbors of b . Let $N(b)$ be the neighboring blocks of b . Clearly for all $b \in B$, $N(b) \subseteq A$.

Let C be the set of blocks in B with degree less than 6 and F be the set of blocks with degree greater than or equal to 6. Now, for each block $c \in C$, we check if c has a pair of distinct neighbors $x, y \in A$ such that $\{x, y\}$ has not yet been assigned to any vertex in C . We can do this in linear time because each block in C has degree less than 6.

Now, let D be the set of blocks in C that have been assigned and $E = C \setminus D$. We argue that $|D| < 3|A|$. To see this, consider the graph H obtained by collapsing each block to single vertex. Since the blocks form a H -partition, H is planar. Now consider the subgraph I of H induced by the vertices corresponding to the blocks in $A \cup D$. For each block $d \in D$, let $\{x, y\}$ be the pair of blocks in A assigned to d . Then, we contract the edge between the vertices in I corresponding to d and x . After these contractions, the resulting graph has $|A|$ vertices and at least $|D|$ edges because all vertices corresponding to blocks in D are contracted with a vertex corresponding to a block in A and each contraction results in a unique edge being created. Since Euler's formula implies that in planar graphs, $|E| \leq 3|V| - 6$, then $|D| < 3|A|$ because a planar graph cannot have a nonplanar graph as a minor.

Then, we consider E the set of blocks in C that have not been assigned. Now, we partition E into P_1, \dots, P_s such that two blocks $u, v \in E$ are in the same partition if and only if $N(u) = N(v)$. We further partition each P_i into $P_{i,1}, \dots, P_{i,t_i}$ such that for $j < t_i$, $13z \leq \sum_{b \in P_{i,j}} |b| < 26z$ and $\sum_{b \in P_{i,t_i}} |b| < 13z$. This can be done greedily because $|b| < 13z$ for all $b \in E$. Now, we merge the blocks in each set $P_{i,j}$ into a single block. Our partition remains a connected H -partition because all blocks in $P_{i,j}$ have the same neighborhood. Moreover, the size of our blocks remains less than $26z$. Let E_{big} be the blocks formed by merging $P_{i,j}$ for $j < t_i$ and E_{small} be the blocks formed by merging P_{i,t_i} .

Next, we will show that $|E_{\text{small}}| < 8|A|$. Suppose for the purpose of contradiction that $|E_{\text{small}}| \geq 8|A|$. Let X be the subgraph of H induced by the vertices corresponding to blocks in A . Then, for each block $b \in E_{\text{small}}$ we add a clique on the vertices corresponding to $N(b)$. Since vertices in E_{small} have distinct neighborhoods, this process adds at least $|E_{\text{small}}| \geq 8|A| = 8|V(X)|$ cliques. A theorem of Wood [W] states that every graph with $n \geq 3$ vertices and no K_5 -minor has at most $8(n-2)$ cliques. It is easy to check then that every graph with no K_5 -minor has less than $8n$ cliques. Therefore, X must have K_5 as a minor. Consider every edge (x, y) in this K_5 minor not in the subgraph of H induced by the vertices corresponding to blocks in A . Then $b_x, b_y \in N(b)$ for $b \in E_{\text{small}}$ where b_x and b_y are blocks corresponding to x and y . But b is not assigned to any pair of neighbors, so there must be a block $d \in D$ assigned to $\{b_x, b_y\}$. Let u be the vertex in H corresponding to d . We can then replace the edge (x, y) with the path (x, u, y) to obtain a K_5 minor in the subgraph of H induced by the vertices corresponding to the blocks in $A \cup D$. But then, K_5 is a minor of H , which contradicts the assumption that H is planar.

We finally note that $|F| < |A|$. Suppose for the purpose of contradiction that $|F| \geq |A|$. Then the subgraph I of H induced by the vertices corresponding to blocks in $C \cup A$ has at least $6|C|$ edges and $|C| + |A| \leq 2|C|$ vertices. Since Euler's formula implies that in planar graphs, $|E| \leq 3|V| - 6$, I cannot be planar. But I is a minor of G and since planar graphs are minor-closed, G cannot be planar, so we have a contradiction.

Now, we have a connected H -partition with sets of $A \cup E_{\text{big}}$ of big blocks and sets $D \cup E_{\text{small}} \cup F$ of small blocks. We have also proved that $|D| < 3|A|$, $|E_{\text{small}}| < 8|A|$, and $|F| < |A|$. Thus, the number of small blocks is less than $12|A|$. By definition the number of big blocks is at most $n/13z$. In particular, $|A| \leq n/13z$. Therefore, the number of small blocks is less than $12n/13z$. Hence, the total number of blocks is less than $13n/13z = n/z$ and each block is of size less than $26z$ so we are done. \square

Lemma 4.10. *There is a linear-time algorithm $\text{Stack-Partition}(G, z)$ which given a constant z and a stack G of $n > z$ vertices, outputs a partitioning of size less than n/z and where each partition is of size less than $26z$. Furthermore, if H is the graph obtained from G by collapsing this is a partitioning into one vertex, H is a stack. If $n < z$, we have a trivial partitioning of one partition of size less than z .*

Proof. If $q \geq 13z$, we can find a H -partition of the graph induced by $\{v_{1,j}\}_j$ of size less than q/z where each partition is of size less than $26z$. Then, we simply copy this partition at all levels of the stack, resulting in at most $pq/z = n/z$ partitions of size less than $26z$. Then, when the partitions are collapsed to vertices to form H , the same planar graph is copied in all levels and between levels, the only partitions that have edges between them are copies of one another. Thus H is a stack.

If $z < q < 13z$, we simply partition the graph into its levels. Then clearly, the partition is of size less than n/z and each partition is of size less than $13z < 26z$.

If $q \leq z$, then $p \geq 2$, and we can greedily partition the levels into sets of total size in $(z, 2z]$ except for the last one which might be of size less than or equal to z . Then, we merge the last one with the second last one so we can partition of size in $(z, 3z]$ so we have less than n/z partitions where each partition is of size less than $26z$. \square

We now consider the procedure $\text{Divide}(R, S, r)$. We follow the Divide procedure in [5, 9] except using our separator algorithm described in Theorem 4.2 which works for stacks. Our Divide procedure can be

described as follows. Denoting n the number of the vertices in R , assign weight $\frac{1}{n}$ to each vertex of R and find a $O(n^{2/3})$ -separator in R using the algorithm described in Theorem 4.2. Since the resulting regions remain stack, we recursively apply this separation algorithm to each region with more than r vertices. Now each region has at most r vertices. However, there may be regions with more than $O(r^{2/3})$ boundary vertices (a boundary vertex is either in S or belong to more than one region). For such a region, do the following: let n' be the boundary vertices in the region, assign weight $\frac{1}{n'}$ to each of them and weight 0 to the other vertices, and apply our separation algorithm to the region. The performance of our Divide procedure is stated in the following lemma, which is the same as that of the Divide procedure in [9]. In fact, the proof of correctness is the same as that in [9], except we replace their separation theorem for minor-closed graphs with our separation theorem for stacks.

Lemma 4.11. *There is a $Divide(R, S, r)$ procedure that divides the stack R such that if n denotes the number of vertices in R ,*

- *it divides G into at most $c_2(\frac{|S|}{r^{2/3}} + \frac{n}{r})$ regions for some constant c_2 ;*
- *each region has at most r vertices;*
- *each region has at most $c_1 r^{2/3}$ boundary vertices for some constant c_1 , where the vertices in S also count as boundary;*
- *it takes time $O(n \log n)$*

4.2.2 Establishing the degree requirement

To apply the result by Henzinger et al. [7] to obtain a linear time algorithm for computing shortest path on a graph, we have to transform the graph to have maximum degree 3. We perform the transformation using the same procedure as presented by Tazari and Müller-hannemann [9] where it is performed on a graph excluding a fixed minor. The correctness of the procedure only exploits the fact that the number of edges in each subregion R in the recursive division is linear to the number of vertices, which obviously holds in our case where R is a stack. Below we just state the result without proof as it is essentially the same as described in [9].

Lemma 4.12. *Let G be a stack and let T be a recursive division tree representing an (\bar{r}, f) -recursive division of G obtained by Algorithm 1. Then one can replace every vertex of G with a zero-weight cycle to obtain a graph G' and at the same time modify T into a tree T' , so that G' has in-/outdegree at most 2 and T' represents an (\bar{r}, f) -recursive vision of G' . This procedure takes linear time.*

5 Conclusion and future work

We have shown an $O(pq)$ -time algorithm for matching a polygonal curve of complexity p to a curve on an embedded planar graph of complexity q . Our strategy is to reduce the problem to that of computing the bottleneck shortest paths on a particular class of graph called stacks. We adapt the results from graph separators [10] and linear time shortest path algorithms [7, 9] to develop a linear time shortest path algorithms for stack. We believe that such graphs are typical in many dynamic programming problems. Hence, our results for asymptotic improvement may also be applied to such problems.

Possible extensions to this work include algorithms and lower bounds for map matching with respect to other distance measures, and obtaining faster map matching algorithms in the case where we are allowed preprocess the embedded map, as in [6]. In addition, road networks may not be planar. One natural question to consider is if our work can be extended more general road network models such as disk intersection graphs [4], which have small separators but do not exclude a fixed minor. Finally, we would like to follow our original motivation and apply such results to other problems such as roadmap reconstruction and databases for polygonal curves.

References

- [1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. Algorithms*, 49(2):262–283, 2003.
- [2] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.
- [3] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 853–864. VLDB Endowment, 2005.
- [4] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. *CoRR*, abs/0808.3694, 2008.
- [5] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 252–257, New York, NY, USA, 1983. ACM.
- [6] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Efficient point-to-point shortest path algorithms. Technical report, October 2005.
- [7] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.
- [8] B. Reed and D. R. Wood. Fast separation in a graph with an excluded minor. In S. Felsner, editor, *2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05)*, volume AE of *DMTCS Proceedings*, pages 45–50. Discrete Mathematics and Theoretical Computer Science, 2005.
- [9] S. Tazari and M. M. Shortest paths in linear time on minor-closed graph classes with an application to steiner tree approximation. submitted for publication, Dec 2007.
- [10] D. R. Wood. On the maximum number of cliques in a graph. *Graph. Comb.*, 23(3):337–352, 2007.