Distance Measures for Embedded Graphs

Hugo A. Akitaya
Department of Computer Science, Tufts University, Medford, MA, USA
hugo.alves_akitaya@tufts.edu

Maike Buchin
Department of Mathematics, Ruhr University Bochum, Bochum, Germany
Maike.Buchin@rub.de

Bernhard Kilgus
Department of Mathematics, Ruhr University Bochum, Bochum, Germany
Bernhard.Kilgus@rub.de

Stef Sijben
Department of Mathematics, Ruhr University Bochum, Bochum, Germany
Stef.Sijben@rub.de

Carola Wenk
Department of Computer Science, Tulane University, New Orleans, LA, USA
cwenk@tulane.edu

Abstract

We introduce new distance measures for comparing straight-line embedded graphs based on the Fréchet distance and the weak Fréchet distance. These graph distances are defined using continuous mappings and thus take the combinatorial structure as well as the geometric embeddings of the graphs into account. We present a general algorithmic approach for computing these graph distances. Although we show that deciding the distances is NP-hard for general embedded graphs, we prove that our approach yields polynomial time algorithms if the graphs are trees, and for the distance based on the weak Fréchet distance if the graphs are planar embedded. Moreover, we prove that deciding the distances based on the Fréchet distance remains NP-hard for planar embedded graphs and show how our general algorithmic approach yields an exponential time algorithm and a polynomial time approximation algorithm for this case. Our work combines and extends the work of Buchin et al. [13] and Akitaya et al. [7] presented at EuroCG.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, graph comparison, embedded graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.55

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1812.09095.pdf`.

Funding *Hugo A. Akitaya*: Supported by National Science Foundation grants CCF-1422311 and CCF-1423615, and the Science Without Borders scholarship program.
Bernhard Kilgus: Supported by the Deutsche Forschungsgemeinschaft (DFG), project BU 2419/3-1.
Carola Wenk: Supported by National Science Foundation grant CCF-1618469.

1 Introduction

There are many applications that work with graphs that are embedded in Euclidean space. One task that arises in such applications is comparing two embedded graphs. For instance, the two graphs to be compared could be two different representations of a geographic network (e.g., roads or rivers). Oftentimes these networks are not isomorphic, nor is one interested in subgraph isomorphism, but one would like to have a mapping of one graph to the other, and ideally such a mapping would be continuous. For instance, this occurs when we have a ground truth of a road network and a simplification or reconstruction of the same network and we would like to measure the error of the latter. In this case, a mapping would identify the parts of the ground truth that are reconstructed/simplified and would allow to study the local error.

We present new graph distance measures that are well-suited for comparing such graphs. Our distance measures are natural generalizations of the Fréchet distance [10] to graphs and require a continuous mapping, but they don't require graphs to be homeomorphic. One graph is mapped continuously to a portion of the other, in such a way that edges are mapped to paths in the other graph. The graph distance is then defined as the maximum of the strong (or weak) Fréchet distances between the edges and the paths they are mapped to. This results in a directed or asymmetric notion of distance, and we define the corresponding undirected distances as the maximum of both directed distances. The directed distances naturally arise when seeking to measure subgraph similarity, which requires mapping one graph to a subgraph of the other.

For comparing two not necessarily isomorphic graphs only few measures were known previously. One such measure is the traversal distance suggested by Alt et al. [9] and another is the geometric edit distance suggested by Cheong et al. [14]. The traversal distance converts graphs into curves by traversing the graphs continuously and comparing the resulting curves using the Fréchet distance. It is also a directed distance that compares the traversal of one graph with the traversal of a part of the other graph. However, an explicit mapping between the two graphs is not established, and part of the connectivity information of the graphs is lost due to the conversion to curves. The geometric edit distance minimizes the cost of edit operations from one graph to another, where cost is measured by Euclidean lengths and distances. But again, connectivity is not well maintained. Figure 1 shows some examples of graphs where our graph distances, the traversal distance, and the geometric edit distance differ. In particular, only our graph distances capture the difference in connectivity between graphs G_1 and G_2, as well as between H_1 and H_2.

Our graph distances map one graph onto a subgraph of the other and they measure the Fréchet distance between the mapped parts (see Section 2.1 for a formal definition). Hence connectivity information is preserved and an explicit mapping between the two (sub-)graphs is established. One possible application of these new graph distances is the comparison of geographic networks, for instance evaluating the quality of map reconstructions and map simplification. In Section 5, we show first experimental results on graphs of map reconstructions that illustrate that our approach considers both, geometry and connectivity.

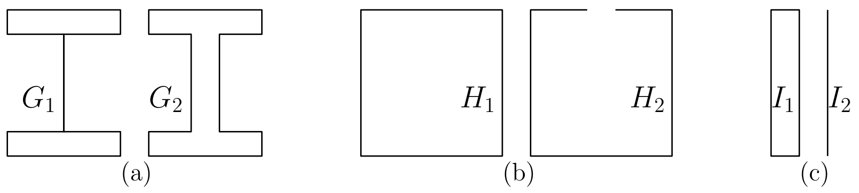


Figure 1 Examples where our graph distances, the traversal distance, and the geometric edit distance differ. For clarity the graphs are shown side-by-side, but in the embedding they lie on top of each other. (a): Graphs G_1 and G_2 have large graph distance (because G_1 needs to mapped to one side of G_2), large edit distance (because a long edge needs to be added), but small traversal distance. (b): Graphs H_1 and H_2 have large graph distance (because all of H_1 needs to mapped to only one side of H_2), but small traversal distance and small edit distance. (c): Graphs I_1 and I_2 have small graph distance and small traversal distance, but a large edit distance (because a long edge needs to be added).

Related work

A few approaches have been proposed in the literature for comparing geometric embedded graphs. Subgraph-isomorphism considers only the combinatorial structure of the graphs and not its geometric embedding. It is NP-hard to compute in general, although it can be computed in linear time if both graphs are planar and the pattern graph has constant size [17]. If we consider the graphs as metric spaces, the Gromov-Hausdorff distance (GH) between two graphs is the minimum Hausdorff distance between isometric embeddings of the graphs into a common metric space. While it is unknown how to compute GH for general graphs, recently Agarwal et al. [1] gave a polynomial time approximation algorithm for the GH between a pair of metric trees. We are however interested in measuring the similarity between two specific embeddings of the graphs. Armiti et al. [11] suggest a probabilistic approach for comparing graphs that are not required to be isomorphic, using spatial properties of the vertices and their neighbors. However, they require vertices to be matched to vertices, which can result in a large graph distance when an edge in one graph is subdived in the other graph. Furthermore, the spatial properties used are invariant to translation and rotation, whereas we consider a fixed embedding. Cheong et al. [14] proposed the geometric edit distance for comparing embedded graphs, however it is NP-hard to compute. Alt et al. [9] defined the traversal distance, which is most similar to our graph distance measures, but it does not preserve connectivity. comparison with the traversal distance.

For assessing the quality of map construction algorithms, several approaches have been proposed. One approach is to compare all paths [2] or random samples of shortest paths [18]. However, these measures ignore the local structure of the graphs. In order to capture more topological information, Biagioni and Eriksson developed a sampling-based distance [12] and Ahmed et al. introduced the local persistent homology distance [3]. The latter distance measure focuses on comparing the topology and does not encode geometric distances between the graphs. The sampling-based distance is not a formally defined distance measure, and it crucially depends on parameters (in particular *matched_distance*, to decide if points are sufficiently close to be matched); in practice it is unclear how these parameters should be chosen. However, it captures the number of matched edges, which is useful when comparing reconstructed road networks. In contrast to these measures, our graph distances capture more topology than the path-based distance [2], and capture differences in geometry better than the local persistent homology distance [3]. Also our graph distances are well-defined distance measures that do not require specific parameters to be set, unlike [12].

Contributions

We present new graph distance measures that compare graphs based on their geometric embeddings while respecting their combinatorial structure. To the best of our knowledge, our graph distances are the first to establish a continuous mapping between the embedded graphs. In Section 2 we define several variants of our graph distances (weak, strong, directed, undirected) and study their properties. In Section 3 we develop an algorithmic approach for computing the graph distances. On the one hand, we prove that for general embedded graphs, deciding these distances is NP-hard. On the other hand, we also show that our algorithmic approach gives polynomial time algorithms in several cases, e.g., when one graph is a tree. The most interesting case is when both graphs are plane. Here, we show that our algorithmic approach yields a quadratic time algorithm for the weak Fréchet distance. In Section 4 we focus on plane graphs and the strong Fréchet distance. For this case, we show that the problem is NP-hard, even though it is polynomial time solvable for the weak Fréchet distance. Furthermore, we show how to obtain an approximation, that depends on the angle between incident edges, in polynomial time and an exact result in exponential time. For this version, we omit some of the proofs or present only proof sketches. Detailed proofs can be found within the version published on arXiv [8].

2 Graph Distance Definition and Properties

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs with vertices embedded as points in $\mathbb{R}^d$ (typically $\mathbb{R}^2$) that are connected by straight-line edges. We refer to such graphs as *(straight-line) embedded graphs.* Generally, we do not require the graphs to be planar. We denote a crossing free embedding of a planar graph shortly as a *plane graph.* Note that for plane graphs G_1 and G_2, crossings between edges of G_1 and edges of G_2 are still allowed.

2.1 Strong and Weak Graph Distance

We define distance measures between embedded graphs that are based on mapping one graph to the other. We consider a particular type of graph mappings, as defined below:

▶ **Definition 1** (Graph Mapping). *We call a mapping $s\colon G_1 \to G_2$ a* graph mapping *if*
1. *it maps each vertex $v \in V_1$ to a point $s(v)$ on an edge of G_2, and*
2. *it maps each edge $\{u, v\} \in E_1$ to a simple path from $s(u)$ to $s(v)$ in the embedding of G_2.*

Note that a graph mapping results in a continuous map if we consider the graphs as topological spaces. To measure similarity between edges and mapped paths, our graph distances use the Fréchet distance or the weak Fréchet distance, which are popular distance measures for curves [10]. For two curves $f, g\colon [0,1] \to \mathbb{R}^d$ their *Fréchet distance* is defined as

$$\delta_F(f,g) = \inf_{\sigma\colon [0,1]\to[0,1]} \max_{t\in[0,1]} ||f(t) - g(\sigma(t))||,$$

where σ ranges over orientation preserving homeomorphisms. The *weak Fréchet distance* is

$$\delta_{wF}(f,g) = \inf_{\alpha,\beta\colon [0,1]\to[0,1]} \max_{t\in[0,1]} ||f(\alpha(t)) - g(\beta(t))|| \;,$$

where α, β range over all continuous onto functions that keep the endpoints fixed.

Typically, the Fréchet distance is illustrated by a man walking his dog. Here, the Fréchet distance equals the shortest length of a leash that allows the man and the dog to walk on their curves from beginning to end. For the weak Fréchet distance man and dog may walk backwards on their curves, for the Fréchet distance they may not. The Fréchet distance and weak Fréchet distance between two polygonal curves of complexity n can be computed in $O(n^2 \log n)$ time [10]. Now, we are ready to define our graph distance measures.

▶ **Definition 2** (Graph Distances). *We define the* directed (strong) graph distance $\vec{\delta}_G$ *as*

$$\vec{\delta}_G(G_1, G_2) = \inf_{s: G_1 \to G_2} \max_{e\in E_1} \delta_F(e, s(e))$$

and the directed weak graph distance $\vec{\delta}_{wG}$ *as*

$$\vec{\delta}_{wG}(G_1, G_2) = \inf_{s: G_1 \to G_2} \max_{e\in E_1} \delta_{wF}(e, s(e)) \;,$$

where s ranges over graph mappings from G_1 to G_2, and e and its image $s(e)$ are interpreted as curves in the plane. The undirected graph distances *are*

$$\delta_G(G_1, G_2) = \max(\vec{\delta}_G(G_1, G_2), \vec{\delta}_G(G_2, G_1)) \qquad \textit{and}$$

$$\delta_{wG}(G_1, G_2) = \max(\vec{\delta}_{wG}(G_1, G_2), \vec{\delta}_{wG}(G_2, G_1)).$$

According to Definition 1, a graph mapping s maps each edge of G_1 to a simple path $s(e)$ in G_2. This is justified by the following observation: Mapping e to a non-simple path $s'(e)$, where $s(e)$ and $s'(e)$ have the same endpoints and $s(e) \subset s'(e)$, does not decrease the (weak) graph distance because $\delta_{(w)F}(e, s(e)) \leq \delta_{(w)F}(e, s'(e))$. From this observation also follows that we cannot decrease $\vec{\delta}_G(G_1, G_2)$ by adding additional vertices to subdivide an edge e of G_1: While the concatenation of the resulting mapped paths in G_2 may not be simple, it can be replaced by the image of the entire edge e, which by the observation has to be simple.

We state a first important property of the graph distances:

▶ **Lemma 3.** *For embedded graphs, the strong graph distances and the weak graph distances fulfill the triangle inequality. The undirected distances are pseudo-metrics. For plane graphs they are metrics.*

Proof. Symmetry follows immediately for the undirected distances. The directed distances fulfill the triangle inequality because we can concatenate two maps and use the triangle inequality of $\mathbb{R}^d$: Let G_1, G_2 and G_3 be three embedded graphs. An edge e of G_1 is mapped to a simple path p in G_2. The segments of p are again mapped to a sequence of simple paths in G_3. Thus, when concatenating two maps, one possible mapping maps each edge e of G_1 to a sequence S of simple paths in G_3. Note, that S need not be simple. However, in that case we can instead map e to a shortest path $\hat{p}$ in S from beginning to end. As $\delta_{(w)F}(e, \hat{p}) \leq \delta_{(w)F}(e, S)$ for each edge of G_1, we have $\vec{\delta}_G(G_1, G_2) + \vec{\delta}_G(G_2, G_3) \geq \vec{\delta}_G(G_1, G_3)$ and $\vec{\delta}_{wG}(G_1, G_2) + \vec{\delta}_{wG}(G_2, G_3) \geq \vec{\delta}_G(G_1, G_3)$ by definition of the directed (weak) graph distance as the maximum Fréchet distance of an edge and its mapping. Analogously, the undirected distances fulfill the triangle inequality as well.

For plane graphs, their (weak) graph distance is zero iff their embeddings are the same, hence the distances are metrics. If the (weak) graph distance is zero, every edge needs to be mapped to itself, hence the embeddings are the same. If on the other hand, the embeddings are the same, a graph mapping may map every edge to itself in the embedding. Since there are no intersections or overlapping vertices, this mapping is continuous in the target graph, and the distance is zero. ◀

Note that for non-plane graphs the (weak) graph distance does not fulfill the identity of indiscernibles. For example, if G_1 consists of two crossing line segment edges, and G_2 has visually the same embedding but consists of four edges and includes the intersection point as a vertex, then both, $\vec{\delta}_G(G_1, G_2) = \vec{\delta}_{wG}(G_1, G_2) = 0$ and $\vec{\delta}_G(G_2, G_1) = \vec{\delta}_{wG}(G_2, G_1) = 0$, and therefore $\delta_G(G_1, G_2) = \delta_{wG}(G_1, G_2) = 0$. Also note that we do not require graph mappings to be injective or surjective. And an optimal graph mapping from G_1 to G_2 may be very different from an optimal graph mapping from G_2 to G_1. See Figure 2 for examples of graphs and their graph distances.

In [8], we show that the traversal distance between a graph G_1 and a graph G_2 is a lower bound for $\vec{\delta}_{wG}(G_1, G_2)$, which follows from the observation that the traversal distance captures the combinatorial structure of the graphs to a lesser extent than our graph distances. Furthermore, we apply the graph distances to measure the similarity between two polygonal paths to examine how these new definitions are generalizations of the (weak) Fréchet distance for curves to graphs.

3 Algorithms and Hardness for Embedded Graphs

Throughout this paper, let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two straight-line embedded graphs, and let $n_1 = |V_1|$, $m_1 = |E_1|$, $n_2 = |V_2|$ and $m_2 = |E_2|$.

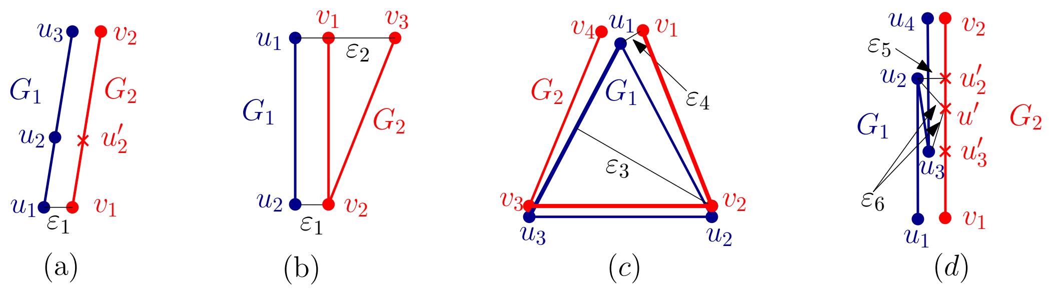


Figure 2 Examples of graph mappings $s_1 : G_1 \to G_2$ and $s_2 : G_2 \to G_1$, and the resulting graph distances. Mapped vertices are drawn with crosses and are not graph vertices. (a) $\vec{\delta}_G(G_1, G_2) = \vec{\delta}_G(G_2, G_1) = \varepsilon_1$. $s_1(u_1) = v_1, s_1(u_2) = u_2', s_1(u_3) = v_2$ and $s_2 = s_1^{-1}$. (b) $\vec{\delta}_G(G_1, G_2) = \varepsilon_1 < \varepsilon_2 = \vec{\delta}_G(G_2, G_1)$. The mapping $s_1(u_1) = v_1$ and $s_1(u_2) = v_2$ is not surjective, and $s_2(v_1) = s_2(v_3) = u_1$ and $s_2(v_2) = u_2$ is not injective. (c) $\vec{\delta}_G(G_1, G_2) = \varepsilon_3 > \varepsilon_4 = \vec{\delta}_G(G_2, G_1)$. $s_1(u_i) = v_i$ and $s_2(v_i) = u_i$ for $i = 1, 2, 3$; $s_2(v_4) = u_1$. (a)-(c) The weak graph distances equal the strong graph distances. (d) $\vec{\delta}_G(G_1, G_2) = \vec{\delta}_{wG}(G_1, G_2) = \vec{\delta}_{wG}(G_2, G_1) = \varepsilon_5 < \varepsilon_6 = \vec{\delta}_G(G_2, G_1)$. Here, the mappings that attain the strong graph distances are $s_1(u_1) = v_1, s_1(u_2) = u_2', s_1(u_3) = u_3', s_1(u_4) = v_2$ and $s_2(v_1) = u_1, s_2(v_2) = u_4$, where s_2 in the limit maps u' to all points on the edge from u_2 to u_3. The mappings attaining the weak graph distances are $s_1^w = s_1$ and $s_2^w = s_1^{-1}$.

First, we consider the decision variants for the different graph distances defined in Definition 2. Given G_1 and G_2 and a value $\varepsilon > 0$, the decision problem for the graph distances is to determine whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (resp., $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$). Equivalently, this amounts to determining whether there exists a graph mapping from G_1 to G_2 realizing $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (resp., $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$). Note that the undirected distances can be decided by answering two directed distance decision problems. As we show in Section 3.3, the value of ε can be optimized by parametric search.

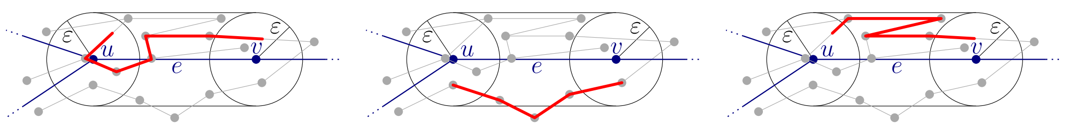
In Section 3.1 we describe a general algorithmic approach for solving the decision problems by computing *valid ε-placements* for vertices. We show that for general embedded graphs the decision problems for the strong and weak directed graph distances are NP-hard, see Section 3.2. However, we prove in Section 3.3 that our algorithmic approach yields polynomial-time algorithms for the strong graph distance if G_1 is a tree, and for the weak graph distance if G_1 is a tree or if both are plane graphs. In the latter scenario (G_1 and G_2 plane graphs), deciding if $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ remains NP-hard, see Section 4.1.

3.1 Algorithmic Approach

Recall, that a (directional) graph mapping that realizes a given distance ε maps each vertex of G_1 to a point in G_2 and each edge of G_1 to a simple path in G_2 within this distance. In order to determine whether such a graph mapping exists, we define the notion of *ε-placements* of vertices and edges; see Figures 3 and 4 (a).

▶ **Definition 4** (ε-Placement). *An* ε-placement *of a vertex v is a maximally connected part of G_2 restricted to the ε-ball $B_\varepsilon(v)$ around v. An* ε-placement *of an edge $e = \{u, v\} \in E_1$ is a path P in G_2 connecting placements of u and v such that $\delta_F(e, P) \leq \varepsilon$. In that case, we say that C_u and C_v are* reachable *from each other. An* ε-placement *of G_1 is a graph mapping $s\colon G_1 \to G_2$ such that s maps each edge e of G_1 to an ε-placement.*

A weak ε-placement *of an edge $e = \{u, v\}$ is a path P in G_2 connecting placements of u and v such that $\delta_{wF}(e, P) \leq \varepsilon$. A* weak ε-placement *of G_1 is a graph mapping $s\colon G_1 \to G_2$ such that s maps each edge e of G_1 to a weak ε-placement.*



(a) An ε-placement of e. **(b)** Not an ε-placement. **(c)** A weak ε-placement.

Figure 3 (a) Illustration of ε-placements of an edge e. (b) Not an ε-placement because the path leaves the ε-tube around e. (c) The Fréchet distance is too large, but e can be mapped to the path if backtracking is allowed. Thus, it is a weak ε-placement.

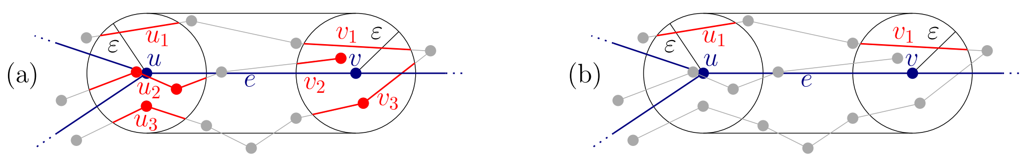


Figure 4 Illustration of valid and invalid vertex placements. (a) Placements u_3 (resp. v_3) are invalid because they are not connected to a placement of v (resp. u) by an ε-placement of the edge e. Placement v_2 is valid when considering e in isolation, but it cannot connect to a placement for the edge that leaves v to the right. Thus, it is also invalid. As a result of pruning v_2 (right), u_2 becomes invalid as well, leaving only u_1 and v_1 as potentially valid placements of u and v (b).

Note that an ε-placement of a vertex v consists of edges and portions of edges of G_2, depending whether $B_\varepsilon(v)$ contains both, one or zero endpoint(s) of the edge, see Figure 4. Also note that each vertex has $O(m_2)$ ε-placements, since an ε-placement is defined as a connected part of G_2 of maximal size inside $B_\varepsilon(v)$. Furthermore, we consider two graph mappings s_1 and s_2 from G_1 to G_2 to be equivalent in terms of the directed (weak) graph distance if for each vertex $v \in V_1$, $s_1(v)$ and $s_2(v)$ are points on the same ε-placement of v.

General Decision Algorithm

Our algorithm consists of the following four steps, which we describe in more detail below. We assume ε is fixed and use the term *placement* for an ε-placement.

Observe that each connected component of G_1 needs to be mapped to a connected component of G_2, and each connected component of G_1 can be mapped independently of the other components of G_1. Hence we can first determine the connected components of both graphs, and then consider mappings between connected components only. In the following we present an algorithm for determining if a mapping from G_1 to G_2, that realizes a given distance ε, exists, where both G_1 and G_2 are connected graphs.

Algorithm 1 General Decision Algorithm.

```
1: Compute vertex placements.
2: Compute reachability information for vertex placements.
3: Prune invalid placements.
4: Decide if there exists a placement for the whole graph $G_1$.
```

1. Compute vertex placements

We iterate over all vertices $v \in V_1$ and compute all their placements. Each vertex has $O(m_2)$ placements, so the total number of vertex-placements is $O(n_1 \cdot m_2)$, and they can be computed in $O(n_1 \cdot m_2)$ time using standard algorithms for computing connected components.

2. Compute reachability information of vertex placements

Next, we iterate over all edges $e = \{u, v\} \in E_1$ to determine all placements of its vertices that allow a placement of the edge. That is, we search for all pairs of vertex-placements C_u, C_v that are reachable from each other according to Definition 4.

For the weak graph distance, we need to find all pairs of placements of u and placements of v that can reach one another using paths contained in the ε-tube $T_\varepsilon(e)$ around e, i.e., the set of all points with distance $\leq \varepsilon$ to a point on e, see Figure 3 (c). If we restrict G_2 to its intersection with the ε-tube, all placements in the same connected component are mutually reachable. Thus, each edge is processed in time linear in the size of G_2 using linear space per edge: For each connected component a pair of lists containing the placements of u and v in that component, respectively, is computed. So, all reachability information can be computed in $O(m_1 \cdot m_2)$ time and space. Note that the weak Fréchet distance between a straight line edge $e \in E_1$ and a simple path $s(e)$ in G_2 is the maximum of the Hausdorff distance between e and $s(e)$ and the distances of the endpoints of e and $s(e)$.

For the strong graph distance, existence of a path inside the ε-tube is not sufficient to describe the connectivity between placements. We must ensure that the Fréchet distance between e and P is at most ε, i.e., a continuous and monotone map s must exist from e to P such that $\delta_F(t, s(t)) \leq \varepsilon$ for all $t \in e$. This can be decided in $O(|P|)$ time using the original dynamic programming algorithm for computing the Fréchet distance [10]. In order to determine whether such a path P exists, every placement of u stores a list of all placements of v that are reachable. The connectivity information can be computed by running a graph exploration, starting from each placement, which prunes a branch if the search leaves the ε-tube or backtracking on e is required to map it. This method runs a search for every placement of the start vertex and thus needs $O(m_2^2)$ time per edge of G_1. Since the connectivity is explicitly stored as pairs of placements that are mutually reachable, it also needs $O(m_2^2)$ space per edge. Hence, in total over all edges, $O(m_1 \cdot m_2^2)$ time and space are needed. Summing up, we have:

▶ **Lemma 5.** *To run step 1 and step 2 of Algorithm 1, we need $O(m_1 \cdot m_2)$ time and space for the weak graph distance and $O(m_1 \cdot m_2^2)$ time and space for the strong graph distance.*

3. Prune invalid placements

Now, after having processed all vertices and edges, it still needs to be decided whether G_1 as a whole can be mapped to G_2. To this end, we delete *invalid* placements of vertices.

▶ **Definition 6** (Valid Placement). *An ε-placement C_v of a vertex v is* (weakly) valid *if for every neighbor u of v there exists an ε-placement C_u of u such that C_v and C_u are connected by a (weak) ε-placement of the edge $\{u, v\}$. Otherwise, C_v is* (weakly) invalid.

See Figure 4 for an illustration of (in)valid placements. As shown in the Figure, deleting an invalid placement possibly sets former valid placements to be invalid. Thus, we need to process all placements recursively until all invalid placements are deleted and no new invalid placements occur. Note that the ordering of processing the placements does not affect the final result. To decide which placements of vertices u and v incident to an edge e are valid, we use the reachability information computed in Step 2.

Initially there are $O(n_1 \cdot m_2)$ vertex-placements, each of which may be deleted once. For the weak graph distance, connectivity is stored using connected components inside the ε-tube surrounding an edge $\{u, v\}$. On deleting a placement C_v of v, it is removed from the list containing placements of v. If a component no longer contains placements of v (i.e. its list becomes empty), then all placements of u in that component become invalid. A placement

C_v is deleted at most once and upon deletion it must be removed from one list for every edge incident to v. Thus, the time for pruning C_v is $O(\deg(v))$. Since the sum of all degrees is $2m_1$, all invalid placements can be pruned in $O(m_1 \cdot m_2)$ time. For the strong graph distance, every placement has a list of placements to which it is connected. On deleting C_v, it must be removed from the lists of all placements C_u to which C_v is connected. Each vertex has $O(m_2)$ placements which have to be removed from a list for each neighbor of v. Thus, pruning a placement runs in $O(\deg(v) \cdot m_2)$ time and pruning all invalid placements in $O(m_1 \cdot m_2^2)$ time.

▶ **Lemma 7.** *Pruning all invalid placements takes $O(m_1 \cdot m_2)$ time for the weak graph distance and $O(m_1 \cdot m_2^2)$ time for the strong graph distance.*

Note that after the pruning step all remaining vertex placements are (weakly) valid. However, the existence of a (weakly) valid placement for each vertex is not a sufficient criterion for $\vec{\delta}_G(G_1, G_2)$ ($\vec{\delta}_{wG}(G_1, G_2)$) in general, see Figure 6.

4. Decide if there exists a placement for the whole graph G_1

After pruning all invalid placements, we want to decide if the remaining valid vertex-placements allow a placement of the whole graph G_1. The complexity of this step depends on the graph and the distance measure: for plane graphs we show that we can concatenate weakly valid placements of two adjacent faces (Lemma 12), whereas this is not possible for the directed strong graph distance in this setting (Theorem 15) or for general graphs for both distances (Theorem 10). Although deciding the directed (weak) graph distance is NP-hard for general graphs, there are two settings which may occur after running steps 1-3 of Algorithm 1, making step 4 of the algorithm trivial. Clearly $\vec{\delta}_G(G_1, G_2) > \varepsilon$ ($\vec{\delta}_{wG}(G_1, G_2) > \varepsilon$) if there is a vertex that has no (weakly) valid ε-placement. Furthermore, we have the following:

▶ **Lemma 8.** *If, after running steps 1-3 of Algorithm 1, each internal vertex (degree at least two) has exactly one valid ε-placement (resp., weakly valid ε-placement) and each vertex of degree one has at least one valid ε-placement (resp., weakly valid ε-placement), then G_1 has an ε-placement (resp., weak ε-placement). Thus, $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (resp., $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$).*

Lemma 5, Lemma 7 and Lemma 8 imply the following Theorem.

▶ **Theorem 9.** *If there is a vertex that has no valid ε-placement or if each vertex has exactly one valid ε-placement after running steps 1-3 of Algorithm 1, the directed strong graph distance can be decided in $O(m_1 \cdot m_2^2)$ time and space. Analogously, if there is a vertex that has no weakly valid ε-placement or if each vertex has exactly one weakly valid ε-placement after running steps 1-3 of Algorithm 1, the directed weak graph distance can be decided in $O(m_1 \cdot m_2)$ time and space.*

3.2 NP-Hardness for the General Case

Notwithstanding the special cases in Theorem 9, deciding the (weak) graph distance is not tractable for general graphs.

▶ **Theorem 10.** *Deciding whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ and deciding whether $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$ for two graphs G_1 and G_2 embedded in $\mathbb{R}^2$ is NP-hard.*

Proof Sketch. We show NP-hardness reducing from binary constraint satisfaction problem (CSP) by identifying each variable x_i with a vertex v_i of G_1 and each constraint on two variables x_i, x_j with an edge incident to v_i and v_j. Furthermore, for every possible value for a variable, we add one vertex to G_2 and embed the vertex inside an ε-ball around the variable. We connect two of such vertices corresponding to two different variables with an edge iff the two values satisfy the constraint. Now, deciding the CSP is equivalent to decide if every edge of G_1 can be mapped to a path of G_2 consisting of a single edge. ◀

3.3 Efficient Algorithms for Plane Graphs and Trees

Here, we show that that Algorithm 1 yields polynomial-time algorithms for deciding the strong graph distance if G_1 is a tree (Theorem 14), and the weak graph distance if G_1 is a tree or if both are plane graphs (Theorem 13). More precisely, we show that the existence of at least one (weakly) valid placement for each vertex is a sufficient condition for $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ or $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$.

▶ **Lemma 11.** *If G_1 is a tree and every vertex of G_1 has at least one (weakly) valid ε-placement after running steps 1-3 of Algorithm 1, then G_1 has a (weak) ε-placement. Thus, $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (or $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$).*

Proof Sketch. We view G_1 as a rooted tree, selecting an arbitrary vertex as the root. Now we can greedily map all vertices of G_1 from the root outwards because all placements are valid and no cycle exists where we need to ensure that we start and end in the same valid placement when traversing the cycle. ◀

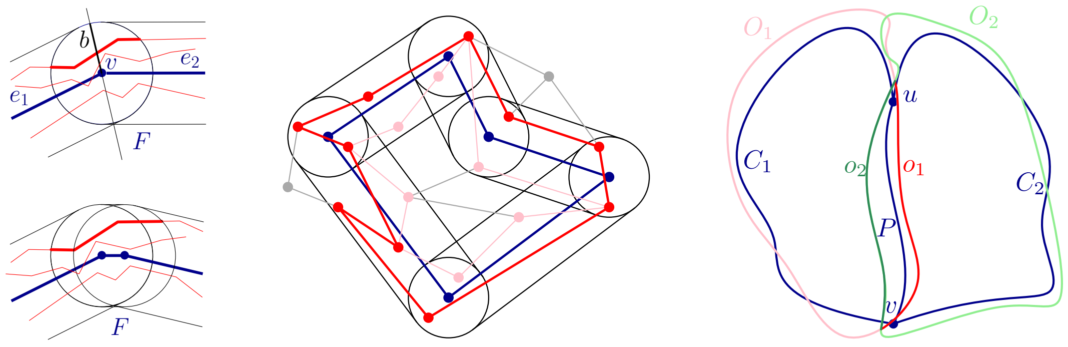
▶ **Lemma 12.** *If G_1 and G_2 are plane graphs and every vertex of G_1 has at least one weakly valid ε-placement after running steps 1-3 of Algorithm 1, then G_1 has a weak ε-placement. Thus, $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$.*

Proof. A *tree-substructure* of G_1 is a tree $T = (V_T, E_T)$ induced by the vertex set $V_T \subset V_1$ with a root vertex $r \in V_T$, such that for all vertices $v \in V_T$, $v \neq r$, v is not an endpoint of an edge $e \in E_1 \backslash E_T$ and such that T is maximal, in the sense that when adding one additional vertex, T contains a cycle. We first remove all tree-substructures of G_1 and map these as in the proof of Lemma 11. Next, we consider all faces of the remainder of G_1 and show how to iteratively map them.

Consider a cycle C bounding a face F and let e_1 and e_2 be two edges of C incident to a vertex v. Let b be the line segment of the bisector of e_1 and e_2 inside $B_\varepsilon(v)$. We define the *outermost placement* of v as the placement which intersects b at maximum distance to the endpoint of b inside F, see Figure 5 (a). Furthermore, we define an *outermost path* in G_2 of an edge $e = \{u, v\}$ of G_1 as the path P_{out} with maximum distance to F connecting the outermost placements of u and v. That is, no subpath Q of P_{out} can be replaced by a path R such that $\delta_H(R, B) \leq \delta_H(Q, B)$, where δ_H is the Hausdorff distance and B is the boundary of the tube $T_\varepsilon(e)$ which lies inside the face F. Note that if an edge is shorter than 2ε, and hence the ε-balls around the vertices overlap, then so possibly do the placements. In particular, in this case the outer placements may overlap, in which case the edge placements degenerate, see Figure 5 (a). Finally, we define an *outer placement* O of C in G_2 as the concatenation of all outermost paths of edges of C.

Note that if C is sufficiently convex the outer placement is simply the cycle that bounds H. See Figure 5 (b) for an example, where the red outer placement bounds the outer face of G_2 restricted to red and pink vertices and edges. The outer placement of C is a weak ε-placement of C.

Now, consider two cycles C_1 and C_2 bounding adjacent faces of G_1, which share a single (possibly degenerate) path P between vertices u and v. Let O_1 and O_2 be the outer placements of C_1 and C_2, respectively. By definition of an outermost placement, O_1 and O_2



(a) A vertex with its placement. **(b)** A cycle with its outer placement. **(c)** Merging two outer placements.

Figure 5 Illustration of outer placements and how to merge them. In (c) the outer placements of cycles C_1 and C_2 can be merged by mapping the shared path P through o_1.

must intersect inside the intersection of the ε-tubes of C_1 and C_2. Let o_1 and o_2 of O_1 and O_2 be the parts between the intersections of O_1 and O_2 containing the respective images of P. Again, by definition of an outermost placement, it holds that o_1 is completely inside O_2 and o_2 is completely inside O_1.

This is illustrated in Figure 5 (c). By planarity there must be a vertex at the intersections of O_1 and O_2. Thus, we can construct a mapping O_2' of C_2 that consists of o_1 and $O_2 \setminus o_2$. This is a weak ε-placement of C_2 for which the image of the shared path P is identical to its image in O_1. Thus, we can merge O_1 and O_2' to obtain a weak ε-placement of these two adjacent cycles. Note that the mapping of C_1 is not modified in this construction. Additionally, the image of the cycle bounding the outer face is its outer placement. The same argument can be applied iteratively when C_1 and C_2 share multiple paths.

If there are two cycles C_1 and C_2 which are connected by a path P such that one endpoint u of P lies on C_1, the other endpoint v of P lies on C_2 and all other vertices of P are no vertices of C_1 or C_2, we can still construct a common placement for C_1, C_2 and P: Let C_u, C_v be the outermost placements of u and v, respectively and let D_v be a vaild placement of v which is connected by a path Q in G_2 to C_u such that $\delta_{wF}(Q, P) \leq \varepsilon$. Such a placement D_v must exist as C_u is a valid placement. If $D_v = C_v$ we have found a common valid placement for C_1, C_2 and P. If $D_v \neq C_v$, by definition of an outermost placement, the path Q must intersect the outermost placement O of C_2 inside the intersection of the tubes $T_\varepsilon(P)$ and $T_\varepsilon(C_2)$. As G_2 is plane, there is a vertex w at the intersection and the resulting path $R = Q_{C_u \to w} + O_{w \to C_v}$ with $\delta_{wF}(R, P) \leq \varepsilon$ connects C_u and C_v.

Now, we iteratively map the cycles bounding faces of G_1 until G_1 is completely mapped. Let $\langle F_1, F_2, \ldots, F_k \rangle$ be an ordering of the faces of G_1 such that each F_i, for $i \geq 2$ is on the outer face of the subgraph $\mathbb{G}_{i-1} := C_1 \cup C_2 \cup \ldots \cup C_{i-1}$ of G_1, where C_j is the cycle bounding face F_j. Thus, let F_1 be an arbitrary face of G_1 and subsequently choose faces adjacent to what has already been mapped. Hence when adding a cycle C_i, we have already mapped $\mathbb{G}_{i-1}$ such that the cycle bounding its outer face is mapped to its outer placement. Thus, we can treat $\mathbb{G}_{i-1}$ as a cycle, ignoring the part of it inside this cycle, and merge its mapping with C_i using the procedure described above. This leaves the mapping of $\mathbb{G}_{i-1}$ unchanged, hence this is still a weak ε-placement of $\mathbb{G}_{i-1}$. However, the mapping of C_i is now modified to be identical to that of $\mathbb{G}_{i-1}$ in the parts where they overlap. Thus, we can merge these mappings to obtain a weak ε-placement of $\mathbb{G}_i$. After mapping F_k we have completely mapped G_1. ◀

Lemma 5 and Lemma 7 together with Lemma 11 and Lemma 12 directly imply the following theorems. Note, that $m_1 = O(n_1)$ for plane graphs and trees, in particular.

▶ **Theorem 13** (Decision Algorithm for Weak Graph Distance). *Let $\varepsilon > 0$. If G_1 is a tree, or if G_1 and G_2 are plane graphs, then Algorithm 1 decides whether $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$ in $O(n_1 \cdot m_2)$ time and space.*

▶ **Theorem 14** (Decision Algorithm for Graph Distance). *Let $\varepsilon > 0$. If G_1 is a tree, then Algorithm 1 decides whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ in $O(n_1 \cdot m_2^2)$ time and space.*

Computing the Distance

To compute the graph distance, we proceed as for computing the Fréchet distance between two curves: We search over a set of critical values and employ the decision algorithm in each step. The following types of critical values can occur:

1. A new vertex-placement emerges: An edge in G_2 is at distance ε from a vertex in G_1.
2. Two vertex-placements merge: The vertex in G_2 where they connect is at distance ε from a vertex in G_1.
3. The (weak) Fréchet distance between a path and an edge is ε: these are described in [10]. There are exponentially many paths in G_2, but each value the Fréchet distance may attain is defined by either a vertex and an edge, or two vertices and an edge.

There are $O(n_1 \cdot m_2)$ critical values of the first two types, and $O(m_1 \cdot n_2^2)$ of type three. Parametric search can be used to find the distance as described in [10], using the decision algorithms from Theorems 13 and 14. This leads to a running time of $O(n_1 \cdot m_2 \cdot \log(n_1 + n_2))$ for computing the weak graph distance if G_1 is a tree or both are plane graphs. And the total running time for computing the graph distance if G_1 is a tree is $O(n_1 \cdot m_2^2 \cdot \log(n_1 + n_2))$.

4 Hardness Results and Algorithms for Plane Graphs

Lemma 12 does not hold for plane graphs and the directed strong graph distance because in general outer placements of cycles cannot be combined to a placement of G_1 as shown in the proof of Lemma 12 , see Figure 6 for a counterexample. In fact we show that deciding the directed strong graph distance for plane graphs is NP-hard.

4.1 NP-Hardness for the Strong Distance for Plane Graphs

▶ **Theorem 15.** *For plane graphs G_1 and G_2, deciding whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ is NP-hard.*

Proof Sketch. We prove the NP-hardness by a reduction from MONOTONE-PLANAR-3-SAT. In this 3-SAT variant, the associated graph with edges between variables and clauses is planar and each clause contains only positive or only negative literals. We construct two graphs G_1 and G_2 where each vertex of G_1 has at least two valid placements. The equivalence of a MONOTONE-PLANAR-3-SAT solution and a valid mapping is obtained by zig-zag shapes of G_2 inside the ε-Ball of some of the vertices of G_1. See Figure 6 for an illustration. ◀

The following stronger result follows from the observation that characteristics of the subgraphs we constructed in the proof of Theorem 15 still hold for a slightly larger ε value.

▶ **Theorem 16.** *It is NP-hard to approximate $\vec{\delta}_G(G_1, G_2)$ within a 1.10566 factor.*

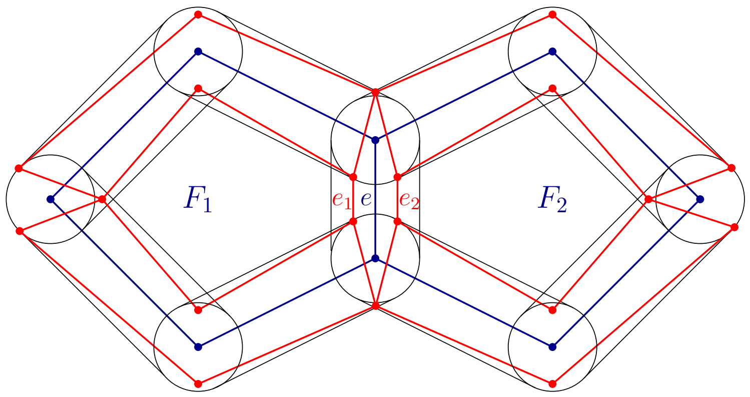


Figure 6 An example of plane graphs G_1 (blue) and G_2 (red) where every vertex of G_1 has two valid placements, but there is no ε-placement of G_1: If the central edge e is mapped to a path through e_1, there is no way to map the cycle bounding face F_2 on the right, and if e is mapped to a path through e_2, the cycle bounding F_1 cannot be mapped.

4.2 Deciding the Strong Graph Distance in Exponential Time

A brute-force method to decide the directed strong graph distance is to iterate over all possible combinations of valid vertex placements, which takes $O(m_1 \cdot m_2^{n_1})$. time. Another approach is to decompose G_1 into faces and merge the substructures bottom-up. This approach is exponential in the number of faces. For more details, see [8].

▶ **Theorem 17.** *For plane graphs, the strong graph distance can be decided in $O(Fm_2^{2F-1})$ time and $O(m_2^{2F-1})$ space, where F is the number of faces of G_1.*

Thus, this method is superior to the brute-force method if $2F - 1 \leq n_1$.

4.3 Approximation for Plane Graphs

For plane graphs, Algorithm 1 yields an approximation depending on the angle between the edges for deciding the strong graph distance. The decision is based on the existence of valid placements. Therefore, the runtime is the same as stated in Theorem 14.

▶ **Theorem 18.** *Let $G_1 := (V_1, E_1)$ and $G_2 := (V_2, E_2)$ be plane graphs. Assume that for all adjacent vertices v_1, $v_2 \in V_1$, $B_\varepsilon(v_1)$ and $B_\varepsilon(v_2)$ are disjoint. Let α_v be the smallest angle between two edges of G_1 incident to vertex v with $deg(v) \geq 3$, and let $\alpha := \frac{1}{2}\min_{v \in V_1}(\alpha_v)$. If there exists at least one valid ε-placement for each vertex of G_1, then $\vec{\delta}_G(G_1, G_2) \leq \frac{1}{\sin(\alpha)}\varepsilon$.*

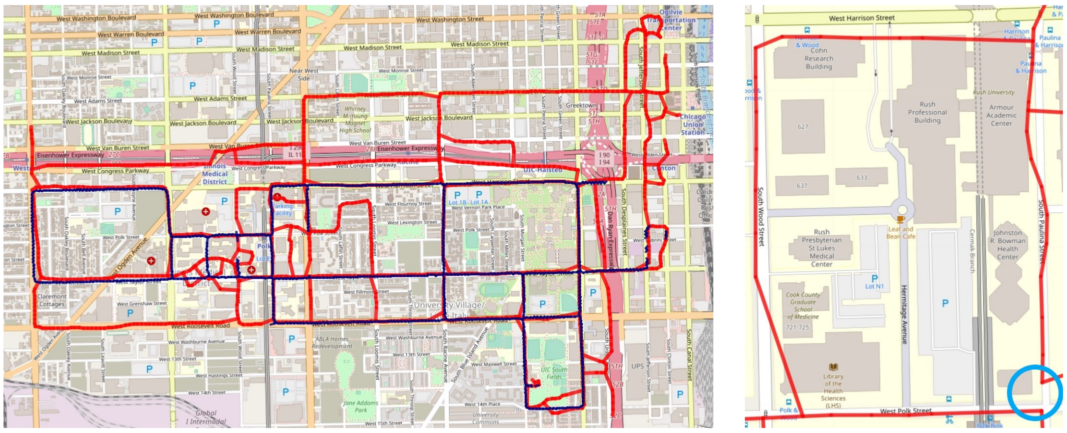
Proof Sketch. For $\hat{\varepsilon} := \frac{1}{\sin(\alpha)}\varepsilon$ the union of all ε-placements of a vertex v with $deg(v) \geq 3$ form a single connected component of G_2 inside $B_{\hat{\varepsilon}}$. Thus, when mapping two adjacent cycles separated by a path P, we ensure that for both mappings the same $\hat{\varepsilon}$-placements of the start and endpoints of P are used. ◀

5 Experiments on Road Networks

In the last decade several algorithms have been developed for reconstructing maps from the trajectories of entities moving on the network [4, 5]. This naturally asks to assess the quality of such reconstruction algorithms. Recently, Duran et al [16] compared several of these algorithms on hiking data, and found that inconsistencies often arise due to noise and low sampling of the input data, for example unmerged parallel roads or the addition of short off-roads.

When assessing the quality of a network reconstruction from trajectory data, several aspects have to be taken into account. Two important aspects are the *geometric* and *topological error* of the reconstruction. Another important aspect is the *coverage*, i.e., how much of the network is reconstructed from the data. We believe our measures to be well suited for assessing the geometric error while still maintaining connectivity information.

We have used the weak graph distance for measuring the distance between different reconstructions and a ground truth of a part of the road network of Chicago. Figure 7 (a) shows two reconstructed road map graphs R (red) and B (blue), overlayed on the underlying ground truth road map G from OpenStreetMap. The reconstruction R in red resulted from Ahmed et al.'s algorithm [6], whereas the reconstruction B in blue from Davies et al.'s [15] algorithm. Our directed graph distance from B to G is 25 meters, and from R to G it is 90 meters. This reflects the local geometric error of the reconstructions (note that it does not evaluate the difference in coverage). Figure 7 (b) shows an example where the topology of R and G differs (blue circle), affecting for instance navigation significantly. This is captured by our distance. Although the reconstruction approximates the geometry well, our measure computes a directed distance of 200 m from G (restricted to the part covered by R) to R.



(a) Two partial map reconstructions of Chicago. **(b)** Different topology.

Figure 7 Two reconstructed road map graphs R (red) and B (blue), overlayed on the underlying ground truth road map G from OpenStreetMap.

6 Conclusion

We developed new distances for comparing straight-line embedded graphs and presented efficient algorithms for computing these distances for several variants of the problem, as well as proving NP-hardness for other variants. Our distance measures are natural generalizations of the Fréchet distance and the weak Fréchet distance to graphs, without requiring the graphs to be homeomorphic. Although graphs are more complicated objects than curves, the runtimes of our algorithms are comparable to those for computing the Fréchet distance between polygonal curves. A large-scale comparison of our approach with existing graph similarity measures is left for future work.

References

1 Pankaj K. Agarwal, Kyle Fox, Abhinandan Nath, Anastasios Sidiropoulos, and Yusu Wang. Computing the Gromov-Hausdorff Distance for Metric Trees. *ACM Trans. Algorithms*, 14(2):24:1–24:20, April 2018. doi:10.1145/3185466.

2 Mahmuda Ahmed, Brittany T. Fasy, Kyle S. Hickmann, and Carola Wenk. Path-based distance for street map comparison. *ACM Transactions on Spatial Algorithms and Systems*, 28 pages, 2015.

3 Mahmuda Ahmed, Brittany Terese Fasy, and Carola Wenk. Local Persistent Homology Based Distance Between Maps. In *22nd ACM SIGSPATIAL GIS*, pages 43–52, 2014.

4 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.

5 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. *Map Construction Algorithms*. Springer, 2015.

6 Mahmuda Ahmed and Carola Wenk. Constructing Street Networks from GPS Trajectories. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 60–71, Berlin, Heidelberg, 2012. Springer-Verlag.

7 Hugo Akitaya, Maike Buchin, and Bernhard Kilgus. Distance Measures for Embedded Graphs - Revisited. In *35th European Workshop on Computational Geometry (EuroCG)*, 2019.

8 Hugo A. Akitaya, Maike Buchin, Bernhard Kilgus, Stef Sijben, and Carola Wenk. Distance Measures for Embedded Graphs. *CoRR*, abs/1812.09095, 2018. arXiv:1812.09095.

9 Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.

10 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1&2):75–91, 1995.

11 Ayser Armiti and Michael Gertz. Geometric graph matching and similarity: A probabilistic approach. *ACM International Conference Proceeding Series*, June 2014.

12 James Biagioni and Jakob Eriksson. Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. *Transportation Research Record: Journal of the Transportation Research Board*, 2291:61–71, 2012.

13 Maike Buchin, Stef Sijben, and Carola Wenk. Distance Measures for Embedded Graphs. In *Proc. 33rd European Workshop on Computational Geometry (EuroCG)*, pages 37–40, 2017.

14 Otfried Cheong, Joachim Gudmundsson, Hyo-Sil Kim, Daria Schymura, and Fabian Stehn. Measuring the similarity of geometric graphs. In *International Symposium on Experimental Algorithms*, pages 101–112, 2009.

15 Jonathan J. Davies, Alastair R. Beresford, and Andy Hopper. Scalable, Distributed, Real-Time Map Generation. *IEEE Pervasive Computing*, 5(4):47–54, 2006.

16 David Duran, Vera Sacristán, and Rodrigo I. Silveira. Map Construction Algorithms: An Evaluation Through Hiking Data. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '16, pages 74–83, 2016.

17 David Eppstein. Subgraph Isomorphism in Planar Graphs and Related Problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

18 Sophia Karagiorgou and Dieter Pfoser. On vehicle tracking data-based road network generation. In *20th ACM SIGSPATIAL GIS*, pages 89–98, 2012.