Distance Measures for Embedded Graphs

Hugo A. Akitaya\textsuperscript{1}, Maike Buchin\textsuperscript{2}, Bernhard Kilgus\textsuperscript{3}, Stef Sijben\textsuperscript{3}, and Carola Wenk\textsuperscript{4}

\textsuperscript{1}Department of Computer Science, Tufts University, Medford, MA, USA, hugo.alves.akitaya@tufts.edu
\textsuperscript{2}Department of Computer Science, Technical University Dortmund, Dortmund, Germany, Maike.Buchin@tu-dortmund.de
\textsuperscript{3}Department of Mathematics, Ruhr University Bochum, Bochum, Germany, Stef.Sijben|Bernhard.Kilgus@rub.de
\textsuperscript{4}Department of Computer Science, Tulane University, New Orleans, USA, cwenk@tulane.edu

March 26, 2019

Abstract

We introduce new distance measures for comparing embedded graphs based on the Fréchet distance and the weak Fréchet distance. These distances take the combinatorial structure as well as the geometric embeddings of the graphs into account. We present a general algorithmic approach for computing these 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 plane. Furthermore, we prove that deciding the distances based on the Fréchet distance remains NP-hard for plane graphs and show how our general algorithmic approach yields an exponential time algorithm and a polynomial time approximation algorithm for this case.

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, being able to compare two road networks is necessary for assessing the quality of map construction algorithms that construct road networks from trajectory data \cite{3, 4}, see Figure 1.

For assessing the similarity between such graphs, it is desirable to determine which parts of the graphs correspond to each other, i.e., to specify an explicit mapping between corresponding parts of the graphs. In road network reconstruction, 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 \cite{14}. An appropriate graph mapping can capture such differences, for example by mapping two unmerged parallel roads in one graph to a single road in the other graph. We

\textsuperscript{*}Supported by National Science Foundation grants CCF-1422311 and CCF-1423615, and the Science Without Borders scholarship program.
\textsuperscript{†}Supported by the Deutsche Forschungsgemeinschaft (DFG), project BU 2419/3-1
\textsuperscript{‡}Supported by National Science Foundation grant CCF-1618469
are also interested in modeling subgraph similarity. Typically only parts of a road network are reconstructed from a given data set, and thus a reconstructed graph should be similar to a subgraph of a ground-truth graph.

**Related work** A few different approaches have been proposed for comparing such graphs, however none of them suitable for comparing embedded graphs such as road networks. Subgraph-isomorphism considers only the combinatorial structure of the graphs, it is NP-hard to compute in general, although in linear time for constant size pattern graphs [15]. Armiti et al. [8] suggest a probabilistic approach for the inexact graph matching problem (graphs are not necessarily required to be isomorphic) based on spatial properties of the vertices and their neighbors. They require vertices to be mapped to vertices of the other graph, whereas we relax this requirement by allowing vertices to be mapped to a point on an edge. Furthermore, for comparing embedded graphs, a geometric edit distance has been suggested, however computing it is also NP-hard [12], and it gives a large distance when one road is represented by two parallel roads (e.g., two lanes). Other distance measures that have been used to assess map construction algorithms compare all paths [11] or random samples of shortest paths [16]. 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 [9] and Ahmed et al. introduced the local persistent homology distance [2]. 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. The sampling-based distance, however, captures the number of matched edges, which is useful when comparing reconstructed road networks. Alt et al. defined a distance based on mapping graph traversals [6], which is
similar to our measures but takes the combinatorial structure of the graphs less into account. For instance, the map reconstruction and the ground truth shown in Figure 1 (b) have a small traversal distance. See also Section 2.2 for further details.

Contributions
We present new distance measures that compare graphs based on their geometric embeddings while respecting their combinatorial structure. Our graph distance measures are based on mapping one graph 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 distance between the edges and the paths they are mapped to.

Since we wish to allow for subgraph similarity, we naturally obtain a directed (i.e., asymmetric) distance by mapping one graph to a subgraph of the other. We define the undirected distance as the maximum of both directed distances. In essence, our distance measures are natural generalizations of the Fréchet distance [7] to graphs without the strict constraint that the graphs have to be homeomorphic to each other. Our distance measures capture more topology than the path-based distance [1], but they capture differences in geometry better than the local persistent homology distance [2]. Also our distances are well-defined distance measures that do not require specific parameters to be set, unlike the sampling-based distance [9] which crucially depends on input parameters.

In Section 2 we define several variants of the graph distance (weak, strong, directed, undirected) and study properties of these distances. 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, and for the weak Fréchet distance for plane graphs. In Section 4 we focus on plane graphs and the strong Fréchet distance. For this case, we present a hardness result and show how to obtain an approximate result in polynomial time and an exact result in exponential time. An interesting insight is that for plane graphs the distance is polynomial time computable for the weak Fréchet distance while it is NP-hard for the Fréchet distance. Due to space limitation, we omit most of the proofs or present only proof sketches in Sections 3 and 4. Detailed proofs can be found in the appendix.

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. A plane graph is a planar embedded graph.

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 map \( s : G_1 \to G_2 \) a graph mapping if

1. it maps each vertex \( v \in V_1 \) to a point \( s(u) \) 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. Our graph distances use the Fréchet distance to measure similarity between edges.
and mapped paths. The Fréchet distance and the weak Fréchet distance are popular distance measures for curves [7]. For two curves \( f, g : [0, 1] \to \mathbb{R}^d \) their Fréchet distance is defined as

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

where \( \sigma \) ranges over all orientation preserving homeomorphisms.

For the weak Fréchet distance \( \delta_{wF} \), both \( f \) and \( g \) are reparameterized as

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

where \( \alpha, \beta \) range over all continuous onto functions that keep the endpoints fixed. For the weak Fréchet distance one can drop the requirement that the reparameterizations \( \alpha, \beta \) keep the endpoints fixed, also called boundary restriction [11].

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 [7].

Now, we are ready to define our graph distance measures.

**Definition 2** (Graph Distances). We define the directed (strong) graph distance \( \tilde{d}_G \) as

\[
\tilde{d}_G(G_1, G_2) = \inf_{s : G_1 \to G_2 \in E_1} \max_{e \in E_1} \delta_F(e, s(e))
\]

and the directed weak graph distance \( \tilde{d}_{wG} \) as

\[
\tilde{d}_{wG}(G_1, G_2) = \inf_{s : G_1 \to G_2 \in E_1} \max_{e \in E_1} \delta_{wF}(e, s(e)),
\]

where \( s \) ranges over all graph mappings from \( G_1 \) to \( G_2 \), and \( e \) and its image \( s(e) \) are interpreted as curves in the plane. We define the undirected graph distances as the maximum of their directed distances: \( \tilde{d}_G(G_1, G_2) = \max(\tilde{d}_G(G_1, G_2), \tilde{d}_G(G_2, G_1)) \) and \( \tilde{d}_{wG} = \max(\tilde{d}_{wG}(G_1, G_2), \tilde{d}_{wG}(G_2, G_1)) \).

**Lemma 1.** For embedded graphs, the strong graph distances and weak graph 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 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 \( \hat{p} \) from \( S \) 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 \( \tilde{d}_G(G_1, G_2) + \tilde{d}_G(G_2, G_3) \geq \tilde{d}_G(G_1, G_3) \) and \( \tilde{d}_{wG}(G_1, G_2) + \tilde{d}_{wG}(G_2, G_3) \geq \tilde{d}_{wG}(G_1, G_3) \) 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.

\[\square\]
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): $\delta(G_1, G_2) = \delta(G_2, G_1) = \epsilon_1$, $s_1(u_1) = v_1, s_1(u_2) = u_2', s_1(u_3) = v_2$ and $s_2 = s_1^{-1}$. (b): $\delta(G_1, G_2) = \epsilon_1 < \epsilon_2 = \delta(G_2, G_1)$. 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): $\delta(G_1, G_2) = \epsilon_3 > \epsilon_4 = \delta(G_2, G_1)$. $s_1(u_i) = v_i$ 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): $\delta(G_1, G_2) = \delta(G_2, G_1) = \epsilon_5 < \epsilon_6 = \tilde{\delta}(G_2, G_1)$. 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 $u'_2$ to all points on the edge from $u_2$ to $u_3$. The mappings attaining the weak graph distances are $s'^w_1 = s_1$ and $s'^w_2 = s_1^{-1}$.

2.2 Relation to Traversal Distance

A related distance measure for graphs was proposed by Alt et al. [6]. They define the traversal distance of two connected embedded graphs $G_1, G_2$ as

$$\delta_T(G_1, G_2) = \inf_{f, g} \max_{t \in [0, 1]} ||f(t) - g(t)||,$$

where $f$ ranges over all traversals of $G_1$ and $g$ over all partial traversals of $G_2$. A traversal of $G_1$ is a continuous, surjective map $f : [0, 1] \to G_1$, and a partial traversal of $G_2$ is a continuous map $g : [0, 1] \to G_2$.

Thus, graphs $G_1, G_2$ have small traversal distance if there is a traversal of $G_1$ and a partial traversal of $G_2$ that stay close together. This could also be used for comparing a graph $G_1$ to a larger graph $G_2$. However, as we observe below, the traversals do not require to maintain the combinatorial structure of $G_1$ within $G_2$. First, we observe that our distance measures are stronger distances in the sense that

$$\delta_T(G_1, G_2) \leq \delta_w(G_1, G_2) \leq \delta_G(G_1, G_2).$$

This follows because a graph mapping that realizes $\delta_G(G_1, G_2) \leq \epsilon$ maps any traversal of $G_1$ to a partial traversal of $G_2$ with distance at most $\epsilon$. For the weak graph distance, the traversal might need to be adjusted, so that it moves back along an already traversed path where the weak Fréchet matching requires it. Note that a traversal need not be injective.

However, the traversal distance captures the combinatorial structure of the graphs to a lesser extent than our measures. Figure 2 shows two graphs that have large graph distance (in particular the directed distance from $G_1$ to $G_2$ is large) but small traversal distance. If indeed $G_1$ is a map reconstruction and $G_2$ the ground truth we are comparing to, then the distance from $G_1$ to $G_2$ should be large.

2.3 Graph Distance for Paths

Consider the simple case that the graphs are paths embedded as polygonal curves. In this case, the (weak) graph distance is closely related to the (weak) Fréchet distance. If the graphs are
single edges embedded as polygonal curves, the graph and curve distances are in fact equal except for orientation of the curves. If the graphs are paths, such that each edge is embedded as a straight segment, the curve and graph distances are related, but not identical, as we show next.

A graph mapping between two paths maps vertices from one path to points on the other path, and it maps edges to the corresponding subpaths. In this case, we can characterize the graph distance in the free space, the geometric structure used for computing the Fréchet distance. Recall that for curves \( f, g : [0, 1] \to \mathbb{R}^d \) the free space is defined as \( F_\varepsilon(f,g) = \{(s,t) \mid d(f(s), g(t)) \leq \varepsilon \} \), i.e., the subset of the product of parameter spaces such that the corresponding points in the image space have distance at most \( \varepsilon \).

**Observation 1.** Let \( P_1, P_2 \) be two polygonal curves parameterized over \([0, m]\) and \([0, n]\), respectively. A graph mapping realizing \( \delta_G(P_1, P_2) \leq \varepsilon \) can be characterized as an \( x \)-monotone path in \([0, m] \times [0, n] \) from the left boundary to the right boundary that is \( y \)-monotone (either increasing or decreasing) in each column of the free space. A graph mapping realizing \( \delta_wG(P_1, P_2) \leq \varepsilon \) is characterized by a path in \([0, m] \times [0, n] \) from the left boundary to the right boundary that is vertex-\( x \)-monotone, i.e., it is monotone in the traversal of the vertices on the \( x \)-axis.

This observation implies relationships between graph distance and (weak) Fréchet distance that are summarized in Lemma 2. In this lemma we use the non-standard variant of (weak) Fréchet distance that does not require the homeomorphism to be orientation preserving, but allows to choose an orientation. Our graph distance does this naturally by choosing where to map. This variant of the Fréchet distance for curves can be computed by running the standard algorithm twice, i.e., searching for a path from bottom-left to top-right corner, as well as from top-left to bottom-right corner in the free space. Alternatively, we could enforce an orientation for the graph distance, e.g., using directions on the graphs.

**Lemma 2.** Let \( P_1, P_2 \) be paths embedded as polygonal curves. Then
\[
\delta_{wF}(P_1, P_2) \leq \delta_{wG}(P_1, P_2) \leq \delta_G(P_1, P_2) \leq \delta_F(P_1, P_2),
\]
where \( \delta_{wF} \) denotes the weak Fréchet distance without boundary restriction.

If \( P_1, P_2 \) are single edges embedded as polygonal paths, equality holds for both the weak and the strong distances.

**Proof.** The last inequality holds because a path in the free space realizing the Fréchet distance is a monotone path in \( x \) and \( y \) from the lower left to upper right corner, hence also realizes both undirected graph distances. For the first inequality, observe that two paths in the free space realizing the two directed weak graph distances can be combined to a path from the left to the right boundary realizing the weak Fréchet distance without boundary restriction.

If \( P_1, P_2 \) are single edges embedded as polygonal paths, this is essentially the same as a parameterization, hence both distances are equal (now with boundary restriction). \( \square \)

Note that if we require mapping endpoints to themselves then the weak graph distance is also lower bounded by the weak Fréchet distance with boundary restriction. A 1D-example of two paths where \( \delta_{wF} \) is strictly smaller than \( \delta_{wG} \) when enforcing to map endpoints is the following: \( P_1 = (0, 2, 0, 2) \) and \( P_2 = (0, 1, 2) \). Here, \( \delta_{wF}(P_1, P_2) = 0 < 1 = \delta_{wG}(P_1, P_2) \).

Intuitively, and confirmed by the above lemma, our graph distance measures are at least as hard to compute as the Fréchet distance. It is known that the Fréchet distance cannot be computed in less than subquadratic time unless the strong exponential time hypothesis fail [10]. Hence we do not expect to compute our graph distance measures more efficiently than quadratic time.
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 $m_2 = |E_2|$.

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 $\overrightarrow{\delta}(G_1, G_2) \leq \varepsilon$ ($\overrightarrow{\delta}(G_1, G_2) \leq \varepsilon$, resp.). Equivalently, this amounts to determining whether there exists a graph mapping from $G_1$ to $G_2$ realizing $\overrightarrow{\delta}(G_1, G_2) \leq \varepsilon$ (resp., $\overrightarrow{\delta}(G_1, G_2) \leq \varepsilon$). 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 $\varepsilon$ can be optimized by parametric search.

In Section 3.1 we describe a general algorithmic approach for solving the decision problems by computing valid $\varepsilon$-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.

3.1 Algorithmic Approach

Recall, that a graph mapping that realizes a given distance $\varepsilon$ 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 $\varepsilon$-placements of vertices and edges; see Figures 3 and 4.

**Definition 3 ($\varepsilon$-Placement).** An $\varepsilon$-placement of a vertex $v$ is a maximally connected part of $G_2$ restricted to the $\varepsilon$-ball $B_\varepsilon(v)$ around $v$. An $\varepsilon$-placement of an edge $e = \{u, v\}$ is a path $P$ in $G_2$ with endpoints on $\varepsilon$-placements $C_u$ of $u$ and $C_v$ of $v$ such that $\delta_P(\varepsilon, P) \leq \varepsilon$. In that case we say that $C_u$ and $C_v$ are reachable from each other. An $\varepsilon$-placement of $G_1$ is a graph mapping $s: G_1 \rightarrow G_2$ such that $s$ maps each edge $e$ of $G_1$ to an $\varepsilon$-placement.

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

Note that an $\varepsilon$-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). Furthermore note that two graph mappings $s_1$ and $s_2$ from $G_1$ to $G_2$ are 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 $\varepsilon$-placement of $v$.

**General Decision Algorithm.** Our algorithm consists of the following four steps, which we describe in more detail below. We assume $\varepsilon$ is fixed and use the term placement for an $\varepsilon$-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 $\varepsilon$, exists, where both $G_1$ and $G_2$ are connected graphs. In particular, this implies that the complexity of the graphs is dominated by their number of edges.
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. First, 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 3. 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 $\varepsilon$-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 $\varepsilon$-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.

For the strong graph distance, existence of a path inside the $\varepsilon$-tube is not sufficient to describe the connectivity between placements. We must ensure that the Fréchet distance between $e$ and $P$ is at most $\varepsilon$, 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 [7]. 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 $\varepsilon$-tube or moving backwards along $e$ 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 3. 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 in step 1 and step 2, 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 4 (Valid Placement). An $\epsilon$-placement $C_v$ of a vertex $v$ is (weakly) valid if for every neighbor $u$ of $v$ there exists an $\epsilon$-placement $C_u$ of $u$ such that $C_v$ and $C_u$ are connected by a (weak) $\epsilon$-placement of the edge $\{u, v\}$. Otherwise, $C_v$ is (weakly) invalid.

See Figure 4 for an illustration of valid and invalid 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 $\epsilon$-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 takes $O(\deg(v) \cdot m_2)$ time and pruning all invalid placements runs in $O(m_1 \cdot m_2^2)$ time.

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 $\delta_G(G_1, G_2)$ (and $\delta_wG(G_1, G_2)$) in general. See Figure 7.

Lemma 4. 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.

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 yield a placement of the whole graph $G_1$. This is not tractable for general graphs. In fact, we show in Section 3.2 that deciding whether $\delta_G(G_1, G_2) \leq \epsilon$ and deciding whether $\delta_wG(G_1, G_2) \leq \epsilon$ are both NP-hard for general graphs. However, in Section 4.3 we prove that in the cases of (1) the strong graph distance if $G_1$ is a tree, and (2) the weak graph distance if $G_1$ is a tree or if both are plane
graphs, we can decide in polynomial time whether the directed graph distance is less than $\varepsilon$ based solely on the existence of valid placements. In the latter scenario ($G_1$ and $G_2$ plane graphs) deciding if $\delta(G_1, G_2)$ remains NP-hard.

For an intuition about the complexity, note that for plane graphs we can concatenate weakly valid placements of two adjacent faces (Lemma 3), whereas this is not possible for the directed strong graph distance in this setting (Theorem 1) or for general graph for both, the directed weak and strong graph distances (Theorem 2).

Although deciding the (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 $\delta(G_1, G_2) > \varepsilon$ (or $\delta_w(G_1, G_2) > \varepsilon$) if there is a vertex that has no (weakly) valid $\varepsilon$-placement. Furthermore, we have:

**Lemma 5.** If, after running steps 1-3 of Algorithm 1, each internal vertex (degree at least two) has exactly one valid $\varepsilon$-placement (resp., weakly valid $\varepsilon$-placement) and each vertex of degree one has at least one valid $\varepsilon$-placement (resp., weakly valid $\varepsilon$-placement), then $G_1$ has an $\varepsilon$-placement (resp., weak $\varepsilon$-placement).

**Proof.** Map each internal vertex $v$ to a point $s(v)$ on its unique (weakly) valid placement $C_v$. Consider an edge $e = \{u, v\} \in E_1$. In the previous step, at least one (weak) placement $P_e$ of $e$ was discovered that connects points $p_u$ and $p_v$ on $C_u$ and $C_v$, respectively, since otherwise $C_u$ and $C_v$ would be invalid. If $p_u \neq s(e)$, $P_e$ can be adapted by shortening it and/or concatenating a path on $C_v$ (i.e., inside $B_2(v')$) without causing its (weak) Fréchet distance to $e$ to become $> \varepsilon$. Adapt $P_e$ to a path $P'_e$ that has endpoints $p_0 = s(u)$ and $p_k' = s(v)$ and define $s(e) = P'_e$. Now, $s$ is a graph mapping from $G_1$ to $G_2$ and each edge is mapped to a path with (weak) Fréchet distance at most $\varepsilon$, so $\delta(G_1, G_2) \leq \varepsilon$ (or $\delta_w(G_1, G_2) \leq \varepsilon$). Recall, that each vertex $w$ of degree one is either connected to an internal vertex $i$, or $G_1$ consists of only one edge $(w, x)$. The first case is already covered, since the unique valid (weak) placement $C_i$ for $i$ is reachable from any valid (weak) placement of $w$. The latter case follows because every vertex placement for $w$ is valid, i.e., for each placement $C_w$ for $w$ there is a (weakly) reachable placement $C_w$ for $x$ and any combination of two reachable placements $C_w$ and $C_x$ yields a valid (weak) placement of $G_1$.

Lemma 3, Lemma 4 and Lemma 5 imply the following Theorem.

**Theorem 1.** If there is a vertex that has no valid $\varepsilon$-placement or if each vertex has exactly one valid $\varepsilon$-placement after running steps 1-3 of Algorithm 1 the (directed) graph distance can be decided in $O(n^1 \cdot m^2_1)$ time and space. Analogously, if there is a vertex that has no weakly valid $\varepsilon$-placement or if each vertex has exactly one weakly valid $\varepsilon$-placement after running steps 1-3 of Algorithm 1 the (directed) weak graph distance can be decided in $O(n^1 \cdot m^2_1)$ time and space.

### 3.2 NP-Hardness for the General Case

Notwithstanding the special cases in Theorem 1, deciding the (weak) graph distance is not tractable for general graphs.

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

**Proof.** We show NP-hardness with a reduction from binary constraint satisfaction problem (CSP), which is defined as follows:

**Problem 1.** Binary Constraint Satisfaction Problem (CSP)

**Instance:** A set of variables $X = \{x_1, \ldots, x_n\}$, variable domains $D = \{D_1, \ldots, D_n\}$ and a set of constraints $C = \{C_1, \ldots, C_m\}$, where each constraint $C$ has
two variables \(x_i, x_j\) and a relation \(R_C \subseteq D_i \times D_j\).

**Question:** Can each variable \(x_i\) be assigned a value \(d_i \in D_i\) such that for each constraint \(C\) on variables \(x_i, x_j\), their values \((d_i, d_j)\) satisfy \(R_C\)?

Consider an instance \(\langle X, D, C\rangle\). Set \(\varepsilon = 1\). Every variable \(x_i\) is represented by a vertex \(v_i\) in \(G_1\) and for each constraint \(C_k\) on variables \(x_i, x_j\), \(G_1\) has an edge \(\{v_i, v_j\}\) in \(G_1\). We embed \(G_1\) such that all adjacent \(\varepsilon\)-balls are separated by at least \(2\varepsilon\) and no \(\varepsilon\)-tube of an edge overlaps an \(\varepsilon\)-ball that does not belong to one of the edge’s endpoints. This can for example be realized by placing all vertices on a sufficiently large circle.

Every value \(d_{i,a} \in D_i\) is represented by a vertex \(u_{i,a}\) in \(G_2\) that is inside the \(\varepsilon\)-ball of \(v_i\). For each pair of values \(d_{i,a} \in D_i, d_{j,b} \in D_j\) allowed by a constraint on \(x_i\) and \(x_j\), \(G_2\) has an edge \(\{u_{i,a}, u_{j,b}\}\). This way, every vertex \(u_{i,a}\) of \(G_2\) defines exactly one \(\varepsilon\)-placement of the corresponding vertex \(v_i\) in \(G_1\). Figure 5 illustrates the construction.

A solution to the CSP consists of selecting a value \(d_i \in D_i\) for each variable \(x_i\), such that if there is a constraint on \(v_i\) and \(v_j\), the pair \(\{d_i, d_j\}\) satisfies the constraint. This is equivalent to selecting a placement \(d_i\) of each vertex \(v_i\), such that if \(G_1\) has an edge \(\{v_i, v_j\}\), then \(G_2\) has an edge connecting \(u_{i,a}\) representing \(d_i\) and \(u_{j,b}\) representing \(d_j\), respectively. The graph distance problem has the weaker requirement that there exists a path \(P_{i,j}\) between \(u_{i,a}\) and \(u_{j,b}\), such that \(\delta_P(\{v_i, v_j\}, P_{i,j}) \leq \varepsilon\). However, the construction is such that only paths consisting of a single edge are permitted for the strong distance, since \(\varepsilon\)-balls must be sufficiently separated and nonoverlapping with \(\varepsilon\)-tubes. So \(G_2\) must have an edge \(\{u_{i,a}, u_{j,b}\}\) if \(G_1\) has an edge \(\{v_i, v_j\}\). So, \(\delta_G(G_1, G_2) \leq \varepsilon\) if and only if \(\langle X, D, C\rangle\) is a satisfiable binary CSP.

For the weak graph distance, edges in \(G_1\) can be mapped to paths consisting of multiple edges in \(G_2\). In this case, there may be weak placements of \(G_1\) that do not represent a solution to the constraint satisfaction instance. To remedy this, we insert a vertex in the middle of each edge of \(G_1\). The vertex is placed such that its \(\varepsilon\)-ball is separated from the \(\varepsilon\)-balls of the original endpoints of the edge by at least \(2\varepsilon\), so each of the new edges is mapped to part of a single edge in \(G_2\). In this construction \(\delta_w(G_1, G_2) \leq \varepsilon\) if and only if \(\langle X, D, C\rangle\) is a satisfiable binary CSP.

![Figure 5: Illustration of the reduction from BCSP.](image.png)

### 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 4), and the weak graph distance if \(G_1\) is a tree or if both are plane graphs (Theorem 3). More precisely, we show that the existence of at least one (weakly) valid placement for each vertex is a sufficient condition for \(\delta_G(G_1, G_2) \leq \varepsilon\) or \(\delta_w(G_1, G_2) \leq \varepsilon\), making it trivial to decide the graph distance after computing (weakly) valid placements.

**Lemma 6.** If \(G_1\) is a tree and every vertex of \(G_1\) has at least one (weakly) valid \(\varepsilon\)-placement after running steps 1-3 of Algorithm 4 then \(G_1\) has a (weak) \(\varepsilon\)-placement. Thus, \(\delta_G(G_1, G_2) \leq \varepsilon\) (or \(\delta_w(G_1, G_2) \leq \varepsilon\)).
Figure 6: 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 $o_1$.

Proof. We view $G_1$ as a rooted tree, selecting an arbitrary vertex as the root. We map all vertices of $G_1$ from the root outwards. First, map the root to an arbitrary (weakly) valid placement. When processing a vertex $v$: Map $v$ to an arbitrary (weakly) valid placement that is reachable from the placement its parent $p$ is mapped to. Recall, that a (weak) $\varepsilon$-placement of a vertex is (weakly) valid if there is an $\varepsilon$-placement for every incident edge. Since $p$ was mapped to a (weakly) valid placement and there is an edge $\{p, v\}$ in $G_1$, there must be at least one such placement of $v$ by definition of a (weakly) valid placement. Since all edges in $G_1$ are tree edges, this ensures that every edge is mapped correctly.

Also, for plane graphs we have:

Lemma 7. If $G_1$ and $G_2$ are plane graphs and every vertex of $G_1$ has at least one weakly valid $\varepsilon$-placement after running steps 1-3 of Algorithm \ref{alg:outer}, then $G_1$ has a weak $\varepsilon$-placement. Thus, $\hat{\delta}_{wG}(G_1, G_2) \leq \varepsilon$.

Proof. First, we remove all tree-like substructures of $G_1$ and place these as described in the proof of Lemma \ref{lem:outer}. Next, we decompose the remainder of $G_1$ into cycles and show how to iteratively place them.

For this, consider a cycle $C$ bounding a face of $G_1$. We define an outer placement $O$ of $C$ in $G_2$. For this we consider the subgraph $H$ of $G_2$ that is restricted to weakly valid placements of the vertices on $C$ and the weak placements of the edges in $C$ connecting these weakly valid vertex-placements. We define $O$ by concatenating outermost placements for the vertices by outermost paths for the adjacent edges in the $\varepsilon$-neighborhood of each. If a vertex has multiple valid placements, the outermost one is the outermost to enter and leave the $\varepsilon$-disk around the vertex, see Figure \ref{fig:outer}(a). Note that if an edge is shorter than $\varepsilon$, and hence the $\varepsilon$-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 \ref{fig:outer}(a).

Note that if $C$ is sufficiently convex the outer placement is simply the cycle that bounds $H$. See Figure \ref{fig:outer}(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 $\varepsilon$-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
can construct a mapping \( O \) of \( C \). Let \( \langle O \rangle \) completely inside \( O \). Theorem 3 (Decision Algorithm for Weak Graph Distance). Note, that \( G \) already been mapped. Hence when adding a cycle \( C \) as a cycle, ignoring the part of it inside this cycle, and merge its mapping with \( G \) of \( \varepsilon \)-place of \( \varepsilon \) to obtain a weak \( \varepsilon \)-placement of these two adjacent cycles. Note that the mapping of \( C \) 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 \( C \) and \( C \) share multiple paths.

Now, we iteratively map the cycles bounding faces of \( G \) until \( G \) is completely mapped. Let \( (F_1, F_2, \ldots, F_k) \) be an ordering of the faces of \( G \) such that each \( F_i \), for \( i \geq 2 \) is on the outer face of the subgraph \( G_{i-1} := C_1 \cup C_2 \cup \ldots \cup C_{i-1} \) 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 \( G_{i-1} \) such that the cycle bounding its outer face is mapped to its outer placement. Thus, we can treat \( G_{i-1} \) 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 \( G_{i-1} \) unchanged, hence this is still a weak \( \varepsilon \)-placement of \( G_{i-1} \). However, the mapping of \( C_i \) is now modified to be identical to that of \( G_{i-1} \) in the parts where they overlap. Thus, we can merge these mappings to obtain a weak \( \varepsilon \)-placement of \( G_i \). After mapping \( F_k \) we have completely mapped \( G_i \).

Lemma 3 and Lemma 4 together with Lemma 6 and Lemma 7 directly imply the following theorem. Note, that \( m_1 = O(n_1) \) for plane graphs and trees.

**Theorem 3** (Decision Algorithm for Weak Graph Distance). Let \( \varepsilon > 0 \). If \( G_1 \) is a tree, or \( G_1 \) and \( G_2 \) are plane graphs, then Algorithm 2 decides whether \( \delta_{\text{wG}}(G_1, G_2) \leq \varepsilon \) in \( O(n_1 \cdot m_2) \) time and space.

Analogously, we have:

**Theorem 4** (Decision Algorithm for Graph Distance). Let \( \varepsilon > 0 \). If \( G_1 \) is a tree, then Algorithm 4 decides whether \( \delta_G(G_1, G_2) \leq \varepsilon \) 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 \( \varepsilon \) from a vertex in \( G_1 \).
2. Two vertex-placegments merge: The vertex in \( G_2 \) where they connect is at distance \( \varepsilon \) from a vertex in \( G_1 \).
3. The (weak) Fréchet distance between a path and an edge is \( \varepsilon \); these are described in [7].

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 \( O(m_1 \cdot m_2^2) \) of type three. Parametric search can be used to find the distance as described in [7], using the decision algorithms from Theorems 3 and 4. 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 \( O(n_1 \cdot m_2^2 \cdot \log(n_1 + n_2)) \).
Figure 7: 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 $\varepsilon$-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 Hardness Results and Algorithms for Plane Graphs

Lemma 7 does not hold for plane graphs and the directed strong graph distance because outer placements are not necessarily valid placements. See Figure 7 for a counterexample. In fact, we show that it is NP-hard to decide the directed strong graph distance for plane graphs.

4.1 NP-Hardness for the Strong Distance for Plane Graphs

Theorem 5. Deciding whether $\delta_G(G_1, G_2) \leq \varepsilon$ is NP-hard even if $G_1$ and $G_2$ are plane graphs.

Proof. 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. The overall idea is to construct two graphs $G_1$ and $G_2$ based on a MONOTONE-PLANAR-3-Sat instance $A$, such that $A$ is satisfiable if and only if $\delta_G(G_1, G_2) \leq \varepsilon$. That is, we construct subgraphs of $G_1$ and $G_2$, where some edges of $G_2$ are labeled TRUE or FALSE in a way, such that only certain combinations of TRUE and FALSE values can be realized by a placement of $G_1$ to $G_2$. To realize other combinations, one would have to walk back and forth at least along one edge of $G_1$, but this is not allowed for the strong graph distance. In the following, we describe the construction of the gadgets (subgraphs) for the variables and the clauses of a MONOTONE-PLANAR-3-Sat instance. Additionally, we need a gadget to split a variable if it is contained in several clauses and a gadget which connects the variable gadgets with the clause gadgets. Furthermore, we prove for each gadget which TRUE and FALSE combinations can be realized and which combinations are not possible. All constructed edges are straight line edges. The graph $G_1$ is shown in blue color and $G_2$ is shown in red color in the sketches used to illustrate the ideas of the proof. We denote $T_\varepsilon(e)$ as the $\varepsilon$-tube around the edge $e$. All vertices of the graph can be either placed arbitrarily within a given $\varepsilon$-surrounding and with a given minimal distance from each other or must lie at the intersection of two lines. Thus, we can ensure that the construction uses rational coordinates only and can be computed in polynomial time.

Following a reparametrization of an edge or a path (embedded curve) as described in Definition 2 we denote by the term "walking along an edge or a path". In this sense "walking back" means that the reparametization is not injective and thus, if walking back and forth is
necessary to stay within $\varepsilon$-distance to another path or edge one can conclude that the Fréchet distance distance between these paths or edges is great than $\varepsilon$ as in this case it is required that the reparameterization is injective.

Furthermore, we call a path labeled True (False) shortly a True (False) signal.

In the following, we give a detailed description of the construction and the properties of the gadgets. The Variable gadget: For each variable of $A$, we add a vertex $v$ and two edges, $e_1$ and $e_2$, of $G_1$ incident to a vertex $v$, where the angle between $e_1$ and $e_2$ is between $90^\circ$ and $120^\circ$. We add a vertex $w_1$ ($w_2$) of $G_2$ on the intersection of the outer boundaries of $T_\varepsilon(e_2)$ ($T_\varepsilon(e_1)$) and a line through $e_1$ ($e_2$). Furthermore, we add a vertex $w_3$ of $G_2$ at the intersection of the boundaries of $T_\varepsilon(e_1)$ and $T_\varepsilon(e_2)$. See the upper left sketch in Figure 8 for an illustration. We connect $w_1$ and $w_2$ with $w_3$ and draw an edge from $w_1$ and $w_2$ inside the $\varepsilon$-tubes around $e_1$ and $e_2$, with labels True. Analogously, we place two edges from $w_3$ with the label False. For the Variable gadget a True-True combination is not possible: There are two placements $p_1$ and $p_2$ of the vertex $v$. Assume we choose $p_1$. Note that one can map $e_1$ to a path containing the edge of $G_2$ with the True labeling inside $T_\varepsilon(e_1)$. Now, we want to map $e_2$ to a path $P$ starting at some point of $p_1$, where $P$ contains the edge of $G_2$ with the True labeling inside $T_\varepsilon(e_2)$. In this case, one has to walk along $e_2$ up to $q$ (the point on $e_2$ with distance $\varepsilon$ to $w_3$) while walking simultaneously to $w_3$ on $P$. But then, when walking along $P$ up to $w_1$, one must walk back along $e_2$ up to $v$ as any point along the interior of $e_1$ has distance greater than $\varepsilon$ to $w_1$. Thus, $\delta_F(e_2, P) > \varepsilon$. It is easy to see, for any other combination of labels there is a placement $p$ of $v$, such that $e_1$ and $e_2$ can be mapped to a path $P_1$ ($P_2$) starting at a point $a$ of $p$ with $\delta_F(e_2, P_1) \leq \varepsilon$ ($\delta_F(e_2, P_2) \leq \varepsilon$).

A Permute gadget is the same as the Variable gadget, but with a different labeling, see Figure 8. The construction of the Split gadget is similar to the Variable gadget. Additionally, we add a third edge $e_3$ of $G_1$ and edges of $G_2$ from $w_2$ and $w_3$ inside the $\varepsilon$-tube around $e_3$. For the labeling, see Figure 8.

The same argument as for the Variable gadget is used to proof the following statements:

- A False signal can never be converted to a True signal in the Split gadget.
- A False signal can never be converted to a True signal in the Permute gadget.

Furthermore, a True signal can, but need not to be converted to a False signal in the Permute gadget.

We construct the Wire gadget used to connect all the other gadgets by drawing two edges $e_1$ and $e_2$ of $G_1$ incident to a vertex $v$ (with arbitrary angle) and two vertices $w_1$ and $w_2$ of $G_2$ inside $B_\varepsilon(v)$ with non intersecting incident edges inside $T_\varepsilon(e_1)$ and $T_\varepsilon(e_2)$. Obviously, it is not possible to convert a True signal to a False signal or vice versa, here.

For the Clause gadget, we first introduce a NAE-Clause gadget. Here it is required that the three values in each clause are not all equal to each other. We start the construction of the NAE-Clause gadget by drawing three edges $e_1$, $e_2$ and $e_3$ incident to a vertex $v$ with a pairwise $120^\circ$ angle. We draw three vertices $w_1$, $w_2$ and $w_3$ on the intersections of $T_\varepsilon(e_1)$, $T_\varepsilon(e_2)$ and $T_\varepsilon(e_3)$ with a maximum distance to $v$. Furthermore, we draw two edges of $G_2$ inside the tubes for each vertex and label them as shown in Figure 8. Let $q_1$ be the point on $e_1$ with distance $\varepsilon$ to $w_1$ and $w_2$. Here, it is not enough to simply connect $w_1$ and $w_2$ with one edge as shown in the bottom left sketch of Figure 8. To force walking back and forth along $e_1$ for a combination of labels which we want to exclude, we have to ensure that a path from $w_1$ to $w_2$ leaves $B_\varepsilon(q_1)$ but stays, once entered, inside $B_\varepsilon(v)$. For the other pairs, ($w_1$, $w_3$) and ($w_2$, $w_3$) we do the same. A possible drawing of these paths maintaining the planarity of $G_2$ is shown in Figure 8. The three placements of $v$ are connected by the vertices $w_1$, $w_2$ and $w_3$ and it holds that there is no placement of $v$ such that an all-False or an all-True labeling can be realized: Suppose we map $v$ to $s(v)$ as shown in the Figure 8. Then, edges $e_2$ and $e_3$ can be mapped to paths through edges labeled True. But we cannot map $e_1$ to such a path $P$.
Figure 8: Building blocks to build graph-similarity instance given a MONOTONE-PLANAR-3-SAT instance.
Let \( q \) and \( v \) be two points of the gadgets. We give a detailed proof for the NAE-Clause gadget, as shown in Figure [8]. Figure [8] partially shows the constructed graphs for a given MONOTONE-PLANAR-3-SAT consisting of the subgraphs (gadgets) described above.

Now, given a MONOTONE-PLANAR-3-SAT instance \( A \), one can construct the graphs \( G_1 \) and \( G_2 \) with the gadgets described above. Note that all gadgets are plane subgraphs. By placing them next to each other with no overlap, we can ensure that \( G_1 \) and \( G_2 \) are plane graphs.

A valid placement of the whole graph \( G_1 \) induces a solution of \( A \): In the corresponding gadget for each positive NAE-clauses, at least one of the outgoing edges of \( G_1 \) must be mapped to a path through the edge labeled \( \text{True} \). By construction, this label cannot be converted to \( \text{False} \) in any of the gadgets and therefore the corresponding variable \( v \) gets the value \( \text{True} \). In this case, \( v \) cannot set any of the negative clauses \( \text{True} \), because the other outgoing edge must be mapped to a path through the edge of \( G_2 \) labeled \( \text{False} \) and this signal can never be switched to \( \text{True} \). The same holds for the case of negative NAE-clauses.

Conversely, given a solution \( S \) of the MONOTONE-PLANAR-3-SAT instance \( A \), it is easy to construct a placement of \( G_1 \). In the variable gadget of variable \( x_1 \) we chose placement \( p_1 \) and map \( e_1 \) to a path through the edge of \( G_2 \) labeled \( \text{True} \) and map \( e_2 \) to a path through the edge labeled \( \text{False} \) if \( x_1 \) is positive in \( S \). If \( x_1 \) is negative, we chose placement \( p_2 \) and map \( e_1 \) and \( e_2 \) accordingly. All edges of the other gadget now can be mapped to \( G_2 \) in a signal preserving manner (\( \text{True} \) stays \( \text{True} \), \( \text{False} \) stays \( \text{False} \)). If there exists a clause \( C \) in \( A \), such that all three variables of \( C \) are positive (negative) in \( S \), we change one signal in the PERMUTE gadget from \( \text{True} \) to \( \text{False} \). Thus, we have found a placement for the whole graph \( G_1 \).

The following stronger result follows from the observation that we can grow \( \varepsilon \) a certain amount, preserving the characteristics of the subgraphs we constructed in the proof of Theorem [5].

**Theorem 6.** It is NP-hard to approximate the strong graph distance within a 1.10566 factor.

**Proof.** The theorem follows from the observation that we can grow \( \varepsilon \) a certain amount, preserving the characteristics of the gadgets. We give a detailed proof for the NAE-Clausé gadget and note that a similar argument holds for the other gadgets. See Figure [10] for an illustration of the arguments and the calculations below.

Let us fix \( \varepsilon = 1 \). As described in the proof of Theorem [5] we connect the vertices \( w_1 \) and \( w_2 \) by a path which leaves \( B_i(q_1) \), but stays inside \( B_i(v_1) \). (Introducing a spike which leaves \( B_i(q_1) \) and returns to \( B_i(q_1) \), see Figure [8].) We draw the spike such that its peak is arbitrarily close to the intersection of a straight line through the edge \( e_1 \) and the 1-circle around \( v \). When enlarging \( \varepsilon \), the point \( q_1 \) moves along \( e_1 \) toward \( \varepsilon \). We need to compute the smallest value \( \delta_{\text{min}} \), such that \( B_i(v) \) is completely contained in \( B_i(1 + \delta_{\text{min}})(q_1) \). For any value \( \delta < \delta_{\text{min}} \), there exists a drawing of the spikes, such that the characteristics of the NAE-Clausé gadget still hold, e.g., there is no placement of \( \varepsilon \) allowing an all-equal-labeling.

Note that \( \delta_{\text{min}} \) equals the distance from \( q_1 \) to \( \varepsilon \), when \( q_1 \) is at distance \( 1 + \delta_{\text{min}} \) to \( w_1 \). Let \( q' \) be the position of \( q_1 \) for \( \delta = 0 \) and let \( d \) be the distance between \( q' \) and \( w_1 \). Then we
Figure 9: For the Monotone-Planar-3-Sat instance $A$ with variables $V = \{x_1, x_2, \ldots, x_5\}$ and clauses $C = \{(x_1 \lor x_2 \lor x_3), (x_3 \lor x_4 \lor x_5), (\overline{x}_1 \lor \overline{x}_3 \lor \overline{x}_5)\}$ the Figure shows the construction of the clause $(x_1 \lor x_2 \lor x_3)$. 18
Figure 10: Illustration of the proof of Theorem 6.

Figure 11: A plane graph (left) is recursively decomposed into chordless cycles by splitting each cycle with a chord (right).

have \( \tan(30^\circ) = \frac{\delta_{\min} + d}{1} = \delta_{\min} + d \). Furthermore, we have \( d = \sqrt{(1 + \delta_{\min})^2 - 1} \) and therefore \( \delta_{\min} = \tan(30^\circ) - \sqrt{(1 + \delta_{\min})^2 - 1} \), which solves to \( \delta_{\min} = \frac{1}{4} - \frac{4}{\sqrt{3}} \approx 0.10566 \). The factor by which \( \varepsilon \) can be multiplied is greater than \( 1 + \delta_{\min} \) for all other gadgets. Thus, \( \delta_{\min} \) is the critical value for the whole construction and the theorem follows.

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. For each such combination, we iterate over all edges of \( G_1 \) to determine whether the vertex placements allow to map each edge to a path with Fréchet distance smaller than \( \varepsilon \). This can be done in constant time per edge using the previously computed reachability information. Thus, the runtime is \( O(m_1 \cdot m_2^n) \).

We can do better if we do not test all possible combinations of placements, but only combinations of placements of critical vertices. These vertices are the endpoints of paths which are incident to two cycles. We start by processing paths separating two adjacent faces and proceed by merging larger substructures until we find a placement for the whole graph, if such a placement exists.

First, we remove all tree-like substructures of \( G_1 \) and place these as described in the proof of Lemma 6. Next, we decompose the remainder of \( G_1 \) into chordless cycles, where a chord is a maximal path in \( G_1 \) incident to two faces (see Figure 11). We place the parts of \( G_1 \) from bottom up, deciding in each step if we can place two adjacent cycles and all the nested
substructures of the cycles simultaneously. To do so, we start with storing all combinations of placements of endpoints of a chord—which separates two faces (chordless cycles)—allowing us to place the two faces simultaneously. We prune all placements which are not part of any valid combination. In the following steps, for each placement $C_u$ of an endpoint $u$ of a chord and each valid combination $c$ of nested chords computed in the previous step, we run one graph exploration. For each placement $C_v$ of the other endpoint $v$ of the chord, which allow two place both cycles simultaneously, we store a new combination consisting of $C_u$, $c$ and $C_v$. We prune all placements of $u$ where we can not reach a valid placement of $v$ by using any of the previous computed combinations. Furthermore, we prune those placements of $v$ which are never reached by any graph exploration. If the list of placements gets empty for one vertex, we can conclude, that the graph distance is greater than $\varepsilon$. Conversely, if we find a valid combinations of placements of the endpoints of the chord in the last step, we can conclude that we can place the whole graph $G_1$ as we guarantee in each step that all substructures can be placed, too.

**Theorem 7.** 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 $G_1$.

**Proof.** Each graph exploration takes $O(m_2)$ time and in each node we have to run $O(m_3k)$ explorations, where $k$ is the number of valid combinations of endpoint placements from previously investigated chords. As the tree has a depth of $\log(F)$, we have

$$\frac{F}{2}m_2^2 + \frac{F}{4}m_2^2m_2^4 + \cdots + m_2^2 \left(m_2^{2\log(F)-2}\right)^2 = F \sum_{i=1}^{\log(F)} \frac{1}{2^i}m_2^{2i+1-2}$$

as the total running time of the graph explorations. We can upper bound this term as follows:

$$F \sum_{i=1}^{\log(F)} \frac{1}{2^i}m_2^{2i+1-2} \leq F \sum_{i=1}^{\log(F)} m_2^{2i+1-2} \leq F \sum_{i=1}^{2\log(F)+1-2} m_2^i$$

$$= F \sum_{i=1}^{2F-2} m_2^i = F \frac{m_2^{2F-1} - m_2}{m_2 - 1} \in O\left(Fm_2^{2F-1}\right),$$

where the first equality uses

$$\sum_{i=1}^{n} a^i = \frac{a^n - 1}{a - 1},$$

for $a \in \mathbb{R}$. In the first step, we have to store $O(m_2^2)$ combinations for each two faces we want to map simultaneously. Let $k$ be the number of combinations in the previous step. Then we have to store up to $k^2m_2^2$ combinations in the next step. This results in storing $O(m_2^{2F-1})$ combinations in the root node of the decomposition. \hfill $\Box$

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

### 4.3 Approximation for Plane Graphs

If the embedding of $G_1$ has some nice characteristics, namely if the embedded edges are not too short and if the angle between two embedded edges is not too small, Algorithm 1 yields a good approximation for deciding the strong graph distance for plane graphs. Again, the decision is based on the existence of valid $\varepsilon$-placements computed in steps 1 to 3. Therefore, the runtime is the same as stated in Theorem 4.
**Theorem 8.** Let $G_1 := (V_1, E_1)$ and $G_2 := (V_2, E_2)$ be plane graphs. Assume that each edge of $G_1$ has length greater than $2\varepsilon$. Let $\alpha_v$ 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 $\varepsilon$-placement for each vertex of $G_1$, then $\delta_G(G_1, G_2) \leq \frac{1}{\sin(\alpha)} \varepsilon$.

**Proof.** Let $\alpha$ be the smallest angle between two edges incident to a vertex $v$ with degree at least three and let $C_1, C_2, \ldots, C_j$ be the valid placements of $v$ for a given distance value $\varepsilon$. Furthermore, let $V_{C_i}$ be the set of vertices of $C_i$. It can be easily shown that for a larger distance value of $\varepsilon_1 \geq \frac{1}{\sin(\alpha)} \varepsilon$, there exists vertices $v_1, v_2, \ldots, v_k$, embedded inside $B_{\varepsilon_1}$, such that the subgraph $C = (V', E')$, where $V' = \bigcup_{i=1}^j V_{C_i} \cup \{v_1\} \cup \{v_2\} \cup \cdots \cup \{v_k\}$ and $E' = \{uw\} \in E_2 | u \in V', w \in V'$ is connected (see Figure 12b). Note that this property is not true if we allow edges with length smaller than $2\varepsilon$: Figure 12b shows an example with $\alpha = 45^\circ$ where the two placements do not merge inside $B_{\varepsilon_1}$. However, with the condition of a minimal edge length of $2\varepsilon$, there is only one valid $\frac{1}{\sin(\alpha)} \varepsilon$-placement $C$ for each vertex with degree at least three. Furthermore, every valid $\varepsilon$ placement is a valid $\frac{1}{\sin(\alpha)} \varepsilon$-placement. Now, for a path $P$ of $G_1$ starting at a vertex $v$, where $\deg(v) \geq 3$ and ending at a vertex $w$, where $\deg(w) \neq 2$, with vertices of degree two in the interior of $P$, the argumentation is analogous to the proof of Lemma 6. For two paths which start and/or end at a common vertex $v$, $v$ is mapped to the same placement as there is only one valid $\frac{1}{\sin(\alpha)} \varepsilon$-placement of $v$. This ensures that each edge of $G_1$ is mapped correctly.

Figure 12: Illustration of the proof of Theorem 8. Left: Three valid $\varepsilon$-placements form one connected component inside a slightly larger ball around $v$. Right: In case of short edges, the placements remain unconnected also inside a larger surrounding of $v$. 

21
5 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] 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.

[2] Mahmuda Ahmed, Brittany Terese Fasy, and Carola Wenk. Local persistent homology based distance between maps. In 22nd ACM SIGSPATIAL GIS, pages 43–52, 2014.

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

[4] Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoer, and Carola Wenk. Map Construction Algorithms. Springer, 2015.

[5] 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.

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

[7] 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.

[8] Ayser Armiti and Michael Gertz. Geometric graph matching and similarity: A probabilistic approach. ACM International Conference Proceeding Series, 06 2014.

[9] 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.

[10] Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly sub-quadratic algorithms unless SETH fails. In IEEE 55th Annual Symposium on Foundations of Computer Science, pages 661–670, 2014.

[11] Kevin Buchin, Maike Buchin, Christian Knauer, Günter Rote, and Carola Wenk. How difficult is it to walk the dog? In Proc. 23rd European Workshop on Computational Geometry (EWCG), pages 170–173, 2007.

[12] 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.
[13] Jonathan J. Davies, Alastair R. Beresford, and Andy Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4):4754, 2006.

[14] 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.

[15] 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. URL: [http://dl.acm.org/citation.cfm?id=313651.313830](http://dl.acm.org/citation.cfm?id=313651.313830).

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