Computing the Fréchet distance of trees and graphs of bounded treewidth*

Maike Buchin[1], Amer Krivošija[2], and Alexander Neuhaus[3]

1 **Ruhr-Universität Bochum, Germany**
`maike.buchin@rub.de`
2 **TU Dortmund, Germany**
`amer.krivosija@tu-dortmund.de`
3 **TU Dortmund, Germany**
`alexander2.neuhaus@tu-dortmund.de`

Abstract

We give algorithms to compute the Fréchet distance of embedded trees and graphs with bounded treewidth. Our algorithms run in $\mathcal{O}(n^2)$ time for trees with fixed root and of bounded degree, and $\mathcal{O}(n^2\sqrt{n \log n})$ time for trees of arbitrary degree. For graphs of bounded treewidth we show one can compute the Fréchet distance in FPT time.

1 Introduction

The Fréchet distance, a distance measure for curves introduced by Fréchet in [8], is a popular measure for comparing polygonal curves. It is defined via homeomorphisms between the parameter spaces of the curves. Intuitively imagine a man walking his dog. The Fréchet distance between the paths of man and dog is the shortest length of a leash connecting them.

The Fréchet distance is well studied. Alt and Godau [4] gave a polynomial time algorithm to compute the Fréchet distance between two curves, sparking research in many applications, such as character recognition [12] and navigation on road maps [13]. One can also define the Fréchet distance between other objects like surfaces [3] or polygons [5]. Here we study the Fréchet distance of straight-line embedded graphs, i.e., every vertex of the graph is assigned to a point in the metric space and every edge between two vertices is a straight line segment between the respective points. First we observe that two graphs are homeomorphic in the topological sense, if they are isomorphic and do not contain degree 2 vertices [5]. That is, a graph homeomorphism induces a graph isomorphism on graphs without degree 2 vertices, and vice versa. Hence we assume that the graphs do not contain degree 2 vertices and define the Fréchet distance between embedded graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_1)$ as:

$$\delta_F(G_1, G_2) = \min_{\pi\colon G_1 \mapsto G_2} \max_{v \in V_1} \|v - \pi(v)\|$$

where $\pi\colon G_1 \to G_2$ is an isomorphism and $\|v - \pi(v)\|$ is the distance between the points corresponding to v and $\pi(v)$. If the graphs do have degree 2 vertices we need to contract every path consisting of degree 2 vertices (except the endpoints) as follows. We view these paths as embedded curves and exchange them with marked edges between their endpoints. We still have to take these curves into account when computing the Fréchet distance of the original graphs. To do so we use the embedded curves of marked edges when computing the Fréchet distance of the graphs.

* This work is based on the BSc thesis and student research project by the third author A. Neuhaus. A. Krivošija was supported by the German Science Foundation (DFG) Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A2. A more detailed version of this work is available at `https://arxiv.org/abs/2001.10502`.

36th European Workshop on Computational Geometry, Würzburg, Germany, March 16–18, 2020.
This is an extended abstract of a presentation given at EuroCG'20. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Related work: To the best of our knowledge, the Fréchet distance of graphs (in this setting) has only been considered by Buchin et al. [5]. They studied the hardness of the Fréchet distance between surfaces and also discuss the Fréchet distance of graphs. In particular, they sketch how to decide in time $\mathcal{O}(n^2 \log n)$ whether there is an isomorphism between two embedded trees respecting a given distance δ. An isomorphism *respects* δ if the distance between every vertex and its image is at most δ. This decider can then be used by a binary search over all possible distances to compute the Fréchet distance. We expand upon this idea and show that one can compute the Fréchet distance between two trees even faster when computing the Fréchet distance between every subtree in a bottom up fashion.

As one can compute the Fréchet distance between two trees it seems natural to look at *tree-like* graphs. We say a graph is tree-like if it has bounded treewidth, see Section 3. The definition of the Fréchet distance given above requires the graphs to be isomorphic. Only recently Lokshtanov et al. [11] gave a canonization algorithm showing that graph isomorphism for graphs of bounded treewidth can be decided in fixed-parameter tractable-time. Their algorithm however only decides whether two graphs are isomorphic but does not provide any isomorphism as a witness. Grohe et al. [9] gave an algorithm computing all isomorphisms between two input graphs. We alter this algorithm to compute the Fréchet distance of two embedded graphs with bounded treewidth.

Results: In Section 2 we give an algorithm computing the Fréchet distance between two embedded trees. We show that this can be done in $\mathcal{O}(n^2\sqrt{d \log d})$ time, where n is the maximum number of vertices and d is the maximum degree of the trees. In Section 3 we show how one can compute the Fréchet distance of two graphs with n vertices and treewidth at most k in $2^{\mathcal{O}(k \log^c k)} n^{\mathcal{O}(1)} \cdot \log n$ time.

2 Algorithm for trees

Computing the Fréchet distance of two graphs requires the graphs to be isomorphic. It is well known that the isomorphism problem for two trees can be solved in polynomial time [1]. If not stated otherwise the trees have a designated root. If they do not, we use the fact that a tree isomorphism needs to match the centers of the trees.

As stated above we require homeomorphisms between the graphs to compute their Fréchet distance. Hence we contract all paths of degree 2 vertices of the two input trees as follows. We start by finding all paths of maximum length containing only degree 2 vertices in each graph with a DFS. For each path found, we connect the endpoints with an edge and store the whole path for the distance calculation as an embedded curve. The last step is to delete all degree 2 vertices and their adjacent edges. For a graph $G = (V, E)$ this can be done in time $\mathcal{O}(|V| + |E|)$ as we basically perform a DFS. Next we show that we can compute all Fréchet distances between edges in time $\mathcal{O}(n^2 \log n)$.

▶ **Lemma 2.1.** *Let $G = (V, E)$ and $G' = (V', E')$ be two trees with $|V| = |V'| = n$. One can contract all paths of degree 2 vertices and store the Fréchet distances between each pair of edges of the contracted graphs in an array of size $\mathcal{O}(n^2)$. This procedure takes $\mathcal{O}(n^2 \log n)$ time, or $\mathcal{O}(n^2)$ time if at least one graph has only paths of degree 2 vertices of constant length.*

Proof. Given two trees $G = (V, E)$ and $G' = (V', E')$ with $|V| = |V'| = n$. Notice that the paths of degree 2 vertices are pairwise disjoint, except possibly for their endpoints. Thus these paths define a partition of V and V' respectively. Also we can bound the number of such paths in each graph by $\frac{n}{2}$. Let $\ell_1, \ldots, \ell_{\frac{n}{2}}$ and $\ell'_1, \ldots, \ell'_{\frac{n}{2}}$ be the lengths of such paths

within G and G' respectively. If there are less than $\frac{n}{2}$ paths the corresponding lengths are 0. It holds that $\sum_{i=1}^{n/2} \ell_i \leq n$ and $\sum_{i=1}^{n/2} \ell'_i \leq n$.

We compute the distance between each pair of edges in the contracted graphs. The easy case is when both edges are non-contracted and we can compute the distance in $\mathcal{O}(1)$ time. If one edge is contracted and the other is not (or is contracted but has constant length), this takes time linear in the length of the contracted edge, which sums up to $\sum_{i=1}^{n/2}(c \cdot \ell'_i) \leq cn$ for a single edge, and hence quadratic time overall.

To compute the Fréchet distances between each pair of contracted edges (of non-constant length), i.e. paths of degree 2 vertices, in G and G' we use the algorithm of Alt and Godau [4] to compute the Fréchet distance of two paths in time $T(\ell_i, \ell'_j) = \mathcal{O}(\ell_i \ell'_j \log(\ell_i \ell'_j))$. Hence in total we need time $\sum_{i=1}^{n/2} \sum_{j=1}^{n/2} T(\ell_i, \ell'_j) \leq cn^2 \log n$. Note that this computation time is only necessary if both graphs have long degree 2 paths.

We store the distances in an array of size $O(|E| \cdot |E'|) = \mathcal{O}(n^2)$ ◀

The further steps are executed on the contracted graphs.

We give a brief overview of the algorithm, based on the ideas in [5]. The algorithm computes the Fréchet distance in a bottom up way, comparing two nodes of same height in every step. Let T and T' be the two input trees and $t \in T, t' \in T'$ two vertices of the same height and equal degree. Let $t_1, \ldots, t_i$ and $t'_1, \ldots, t'_i$ be the children of t and t'. Assume we already have computed the Fréchet distance between the subtrees rooted at the children. To compute the Fréchet distance between the trees rooted at t and t' we follow the ideas in [5] using ε-matchings. To find those matchings we create a new bipartite graph B. For every subtree of t and t' there is a vertex in B. We combine every vertex corresponding to a subtree of t with every vertex corresponding to a subtree of t' with a weighted edge. The weight of the edge is the Fréchet distance between the subtrees, if there are unmarked edges between the roots of the subtrees and their parents. If at least one of the edges is marked we assign the maximum of the Fréchet distance between the edges and the Fréchet distance of the trees. If the subtrees are not isomorphic we assign ∞ as the edge weight. Note if one of the edges was marked we use the corresponding path to compute the distance.

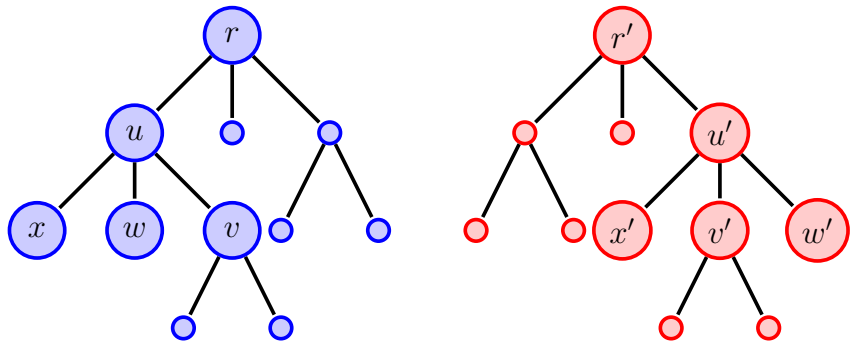
Now we find a *bottleneck matching* for this graph, which equals the ε-matching used in [5]. A bottleneck matching is a perfect matching such that the maximum weight of its edges is minimal for all perfect matchings possible. The Fréchet distance of the trees rooted at t and t' is either the maximum weight of the found bottleneck matching or the distance of t and t'. We store the correct value in a two-dimensional array. If we could not find a bottleneck matching the trees are not isomorphic and thus have no Fréchet distance. In this case we store ∞. If the subtrees are empty we store 0.

The algorithm iterates over both trees doing the above computation for each pair of nodes of same height. Afterwards the algorithm returns the value stored for the roots of the trees.

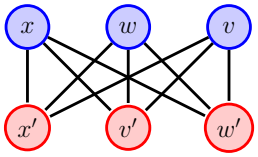
Figure 1 illustrates the computation of a bottleneck matching. Given the two trees in (a) as input. Consider the computation of the Fréchet distance of the trees rooted at u and u'. The graph B is shown in (b) and the distances are given in the table in (c). In (d) we see the computed bottleneck matching with maximum Fréchet distance of 1.

It remains to compute such a bottleneck matching. To do this store all edge weights in a sorted array and search for the smallest edge weight for which we can find a perfect matching. The matching corresponding to this weight is the desired bottleneck matching.

The running time of this algorithm is as follows. We use the algorithm of Alt et al. [2] to compute the perfect matchings. This means we can bound the running time of one bottleneck matching computation by $\mathcal{O}\left(d^2\sqrt{d \log d}\right)$ time, with d being the degree of the



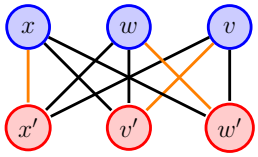
(a)



(b)

	v	w	x
v'	1	2	2
w'	2	1	2
x'	2	2	1

(c)



(d)

Figure 1 Two trees (a), the bipartite graph B (b), the weights (c) and bottleneck matching (d)

trees. A careful analysis of the number of bottleneck computations yields that the algorithm has running time $\mathcal{O}\left(n^2\sqrt{d\log d}\right)$. Thus we get the following result:

▶ **Theorem 2.2.** *The Fréchet distance of two embedded trees can be computed in time* $\mathcal{O}\left(n^2\sqrt{n\log n}\right)$. *If the trees have bounded degree the computation only takes* $\mathcal{O}(n^2)$ *time.*

Note that if we need more than constant time to compute the distance between two points the running time of this algorithm increases. Next we generalize this result to graphs that are not trees but are tree-like in their structure.

3 Algorithm for graphs with bounded treewidth

Here we consider a larger class of graphs. It is known that many hard problems for graphs can be solved easily on trees, such as 3-coloring problem and finding a vertex cover. Both these problems can be solved in linear time on a tree. One can ask if these problems are easy for tree-like graphs. A measurement for such a likeliness is the *treewidth*. To define the treewidth we first introduce *tree decompositions*. Intuitively a tree decomposition of a graph G represents the vertices and edges of G as subgraphs inside a tree. Formally:

▶ **Definition 3.1.** Let $G = (V, E)$ be a graph. Let T be a tree and β a function mapping all $t \in T$, the nodes, to subsets of V. We call (β, T) a tree decomposition of G if:

1. for each vertex $v \in V$ it holds that all subsets $\beta(t) \in T$, called bags, containing v induce a nonempty connected subtree of T, and
2. for each edge $(u, v) \in E$ there is one $t \in T$, such that the bag $\beta(t)$ contains u and v.

The width of such a tree decomposition is the size of its largest bag -1. The treewidth of G is the minimal width among all its tree decompositions.

Tree decompositions play an important role in the context of parameterized algorithms. An algorithm is parameterized, if its running time does not only depend on the input size n

but also on a parameter k. A problem is fixed parameter tractable, if there is a algorithm with running time $f(k) \cdot \text{poly}(n)$ for it. Many graph problems have algorithms with treewidth as parameter, see [6, 7]. Typically these algorithms use dynamic programming over a tree decomposition of the graph. Our algorithm is based on the algorithm of Grohe et al. [9]. We extended their algorithm such that we can compute all isomorphisms between two embedded graphs that respect a distance limit. Using this algorithm we conduct a binary search on every possible distance between nodes of G_1 and G_2. The Fréchet distance of the two graphs is the smallest distance found that is respected by an isomorphism.

We describe our algorithm. Let $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ be two embedded graphs and δ a distance limit. We start by preprocessing the graphs the same way as in the case of trees, see Section 2. Next we compute an initial tree decomposition using the techniques of Leimer [10]. The resulting decomposition is known to be isomorphism invariant. We carefully refine these tree decompositions in a bottom up way using the techniques of [9]. In every step we maintain the invariant that the tree decompositions are isomorphism invariant, the size of the resulting bags is not too high, and we get crucial information about the structure of both graphs. We now use the resulting tree decomposition to compute the isomorphisms respecting δ in a bottom up way. Suppose we are looking at two nodes $t_1 \in T_1$ and $t_2 \in T_2$, and have already computed all isomorphisms respecting δ between the subtrees of t_1 and t_2 rooted at their respective children. To compute the isomorphisms between the graphs induced by the trees rooted at t_1 and t_2 Grohe et al. [9] used an abstraction called coset-hypergraph isomorphism. An instance of this abstraction consists of the two graphs, the isomorphisms computed between the subtrees one can extend to isomorphisms between the graphs, and colorings of the graphs to indicate which node sets can be mapped to each other.

We expand upon this approach by using the colorings to indicate which sets are isomorphic to each other, and respect a given distance limit. Especially the colorings can indicate if two edges correspond to curves that can be mapped to each other. Using these colorings we compute all isomorphisms between the graphs using the algorithm of [9]. The algorithm uses dynamic programming over the tree decompositions in a bottom up way. In every step it constructs all isomorphisms between the subgraphs induced by vertices that were already covered by the tree decompositions. During this the algorithm can find isomorphisms that do not respect the given distance limit. In this case we simply mark such an isomorphism, indicating that it does not witness the distance limit. After this computation we simply check if there is at least one isomorphism that is not marked. In this case we know that the input graphs are isomorphic under the given distance limit. If the dimension of the space where the graphs are embedded is constant, the running time of the algorithm by Grohe et al. [9] does not change. This means one can decide whether two embedded graphs with treewidth at most k are isomorphic under a given distance limit in time $2^{\mathcal{O}(k \log^c k)} n^{\mathcal{O}(1)}$, where n is the number of vertices of one graph, and c is a positive constant.

Furthermore we know that the Fréchet distance between the two graphs must be a distance between two vertices, with one belonging to the first and one to the second graph. Hence we store all n^2 distances in a sorted array and perform a binary search using the above algorithm to find the smallest distance under which the graphs are isomorphic. This yields:

▶ **Theorem 3.2.** *The Fréchet distance between two graphs with n nodes and treewidth at most k can be computed in time $2^{\mathcal{O}(k \log^c k)} n^{\mathcal{O}(1)} \cdot \log n$.*

4 Conclusion

We have shown how to compute the Fréchet distance of two rooted trees and of two graphs with bounded treewidth. It would be interesting to investigate the case of trees with high vertex degrees. For those trees the algorithm presented in Section 2 only gives a slight improvement over the algorithm sketched in [5]. The more general case remains interesting since a polynomial time algorithm for computing the Fréchet distance of two graphs could be used to decide whether the graphs are isomorphic. We do not expect a polynomial time algorithm for general graphs. But perhaps there are other parameters in which the Fréchet distance of two graphs is fixed parameter tractable and can be computed more efficiently.

References

1 A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing, Boston, MA, USA, 1st edition, 1974.

2 H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$. *Inf. Process. Lett.*, 37(4):237–240, Feb. 1991.

3 H. Alt and M. Buchin. Can we compute the similarity between surfaces? *Discrete & Computational Geometry*, 43(1):78, Mar 2009.

4 H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 03 1995.

5 K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: Some simple hard cases. In M. de Berg and U. Meyer, editors, *Algorithms – ESA 2010*, pages 63–74. Springer Berlin Heidelberg, 2010.

6 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 2012.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, Berlin, Heidelberg, 2006.

8 M. M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1):1–72, Dec 1906.

9 M. Grohe, D. Neuen, P. Schweitzer, and D. Wiebking. An improved isomorphism test for bounded-tree-width graphs. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP,*, pages 67:1–67:14, 2018.

10 H.-G. Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, 113(1):99–123, 1993.

11 D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM Journal on Computing*, 46(1):161–189, 2017.

12 S. Sen, J. Chakraborty, S. Chatterjee, R. Mitra, R. Sarkar, and K. Roy. Online handwritten bangla character recognition using Fréchet distance and distance based features. In S. Sundaram and G. Harit, editors, *Document Analysis and Recognition*, pages 65–73, Singapore, 2019. Springer Singapore.

13 K. P. Sharma, R. C. Pooniaa, and S. Sunda. Map matching algorithm: curve simplification for Fréchet distance computing and precise navigation on road network using RTKLIB. *Cluster Computing*, 22(6):13351–13359, Nov 2019.