


# Sketching Merge Trees

Mingzhe Li 

School of Computing, University of Utah  
Salt Lake City, UT, USA  
mingzhel@cs.utah.edu

Sourabh Palande 

School of Computing, University of Utah  
Salt Lake City, UT, USA  
sourabh@sci.utah.edu

Bei Wang<sup>1</sup> 

School of Computing, University of Utah  
Salt Lake City, UT, USA  
beiwang@sci.utah.edu

---

## Abstract

Merge trees are a type of topological descriptors that record the connectivity among the sublevel sets of scalar fields. In this paper, we are interested in sketching a set of merge trees. That is, given a set  $\mathcal{T}$  of merge trees, we would like to find a basis set  $\mathcal{S}$  such that each tree in  $\mathcal{T}$  can be approximately reconstructed from a linear combination of merge trees in  $\mathcal{S}$ . A set of high-dimensional vectors can be sketched via matrix sketching techniques such as principal component analysis and column subset selection. However, up until now, topological descriptors such as merge trees have not been known to be sketchable. We develop a framework for sketching a set of merge trees that combines the Gromov-Wasserstein framework of Chowdhury and Needham with techniques from matrix sketching. We demonstrate the applications of our framework in sketching merge trees that arise from data ensembles in scientific simulations.

**2012 ACM Subject Classification** Theory of computation: Randomness, geometry and discrete structures: Computational geometry

**Keywords and phrases** Merge trees, sketching, topological data analysis, applications, experiments, implementations

**Funding** DOE DE-SC0021015

**Acknowledgements** We thank Jeff Phillips for discussions involving column subset selection and Benwei Shi for his implementation on length squared sampling. We also thank Ofer Neiman for sharing the code on low stretch spanning tree. We thank Lin Yan for sharing ensemble datasets and generating scalar field visualization shown in Figure 2 and Figure 6.

## 1 Introduction

Topological descriptors such as merge trees, contour trees, Reeb graphs, and Morse–Smale complexes serve to describe and identify characteristics associated with scalar fields, with many applications in the analysis and visualization of scientific data [44, 51]. In this paper, we are interested in sketching a set of topological descriptors.

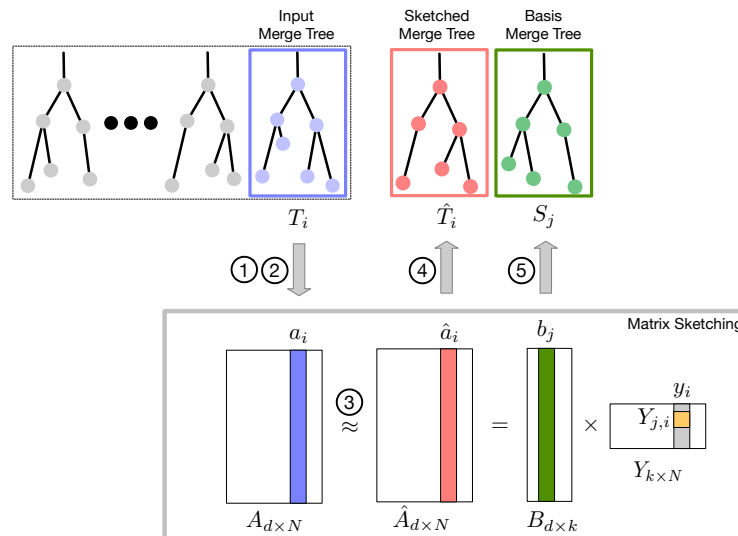
We are motivated by the idea of matrix sketching. A set of high-dimensional vectors is *sketchable* via matrix sketching techniques such as principle component analysis (PCA), and column subset selection (CSS), as illustrated in Figure 1 (gray box). Given a dataset of  $N$  points with  $d$  features, represented as a  $d \times N$  matrix  $A$  (with row-wise zero empirical

---

<sup>1</sup> Corresponding author

mean), together with a parameter  $k$ , PCA aims to find a  $k$ -dimensional subspace  $\mathbb{H}$  of  $\mathbb{R}^d$  that minimizes the average squared distance between the points and their corresponding projections onto  $\mathbb{H}$ . Equivalently, for every input point  $a_i$  (a column vector of  $A$ ), PCA finds a  $k$ -dimensional embedding  $y_i$  (a column vector of  $Y$ ) along the subspace  $\mathbb{H}$  to minimize  $\|A - \hat{A}\|_F^2 = \|A - BY\|_F^2$ .  $B$  is a  $d \times k$  matrix whose columns  $b_1, b_2, \dots, b_k$  form an orthonormal basis for  $\mathbb{H}$ .  $Y$  is a  $k \times N$  coefficient matrix, whose column  $y_i$  encodes the coefficients for approximating  $a_i$  using the basis from  $B$ . That is,  $a_i \approx \hat{a}_i = \sum_{j=1}^k b_j Y_{j,i}$ . Another technique we discuss is CSS, whose goal is to find a small subset of the columns in  $A$  to form  $B$  such that the projection error of  $A$  to the span of the chosen columns is minimized, that is, to minimize  $\|A - \hat{A}\|_F^2 = \|A - BY\|_F^2$ , where we restrict  $B$  to come from columns of  $A$ . Such a restriction is important for data summarization, feature selection, and interpretable dimensionality reduction. Thus, with either PCA or CSS, given a set of high-dimensional vectors, we could find a set of basis vectors such that each input vector can be approximately reconstructed from a linear combination of the basis vectors.

Now, what if we replace a set of high-dimensional vectors by a set of objects that encode topological information of data, specifically topological descriptors? Until now, topological descriptors have not been known to be sketchable. In this paper, we focus on merge trees, which are a type of topological descriptors that record the connectivity among the sublevel sets of scalar fields. We address the following question: given a set  $\mathcal{T}$  of merge trees, can we find a basis set  $\mathcal{S}$  such that each tree in  $\mathcal{T}$  can be approximately reconstructed from a linear combination of merge trees in  $\mathcal{S}$ ?



■ **Figure 1** The overall pipeline for sketching a set of merge trees.

Our overall pipeline is illustrated in Figure 1 and detailed in Section 4. In steps 1 and 2, given a set of  $N$  merge trees  $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$  as input, we represent each merge tree  $T_i$  as a metric measure network and employ the Gromov-Wasserstein (GW) framework of Chowdhury and Needham [21] to map it to a column vector  $a_i$  in the data matrix  $A$ . In step 3, we apply matrix sketching techniques, in particular, column subset selection (CSS) and non-negative matrix factorization (NMF), to obtain an approximated matrix  $\hat{A}$ , where  $A \approx \hat{A} = B \times Y$ . In step 4, we convert each column in  $\hat{A}$  into a merge tree (referred to as a sketched merge tree) using spanning trees, in particular, minimum spanning trees (MST) and low-stretch spanning trees (LSST). Finally, in step 5, we return a set of basis merge trees  $\mathcal{S}$



by applying LSST or MST to each column  $b_j$  in  $B$ . Each entry  $Y_{j,i}$  in the coefficient matrix  $Y$  defines the coefficient for basis tree  $S_j$  in approximating  $T_i$ . Thus, intuitively, with the above pipeline, given a set of merge trees, we could find a set of basis trees such that each input tree can be approximately reconstructed from a linear combination of the basis trees.

Our contribution is two-fold. First, we combine the notion of GW distances with matrix sketching techniques to give a class of algorithms for sketching a set of merge trees. Second, we provide experimental results that demonstrate the utility of our framework in sketching merge trees that arise from scientific simulations. Understanding the sketchability properties of merge trees can be particularly useful for the analysis and visualization of scientific ensembles, where our framework can be used to find good ensemble representatives and to identify outliers.

## 2 Related Work

In this section, we review relevant work on merge trees, Gromov-Wasserstein distances, graph alignment and averaging, matrix sketching, and spanning trees.

**Merge Trees.** Merge trees (also known as barrier trees [36] or join trees [18]) are a type of topological descriptors that record the connectivity among the sublevel sets of scalar fields. They are rooted in Morse theory [60], which characterizes scalar field data by the topological changes in its sublevel sets at isolated critical points. Other types of topological descriptors for scalar fields include contour trees [18], Reeb graphs [67], and Morse–Smale complexes [30, 29]. In this paper, instead of a direct comparison between a pair of merge trees using existing metrics for merge tree or Reeb graphs (e.g., [9, 11, 8, 10, 70, 20, 61, 24, 62, 12]), we treat merge trees as metric measure networks and utilize the Gromov-Wasserstein framework described in Section 3 to obtain their alignment and vector representations.

**Gromov-Wasserstein Distances.** Gromov introduced Gromov-Hausdorff (GH) distances [39] while presenting a systematic treatment of metric invariants for Riemannian manifolds. GH distances can be employed as a tool for shape matching and comparison (e.g., [16, 53, 54, 57, 58]): shapes are treated as metric spaces, and two shapes are considered equal if they are isometric. Memoli [55] modified the formulation of GH distances by introducing a relaxed notion of proximity between objects, thus generalizing GH distances to the notion of Gromov-Wasserstein (GW) distances for practical considerations. Since then, GW distances have had a number of variants based on optimal transport [72, 73] and measure-preserving mappings [56]. Apart from theoretical explorations [55, 71], GW distances have been utilized in the study of graphs and networks [45, 75, 76], machine learning [34, 17], and word embeddings [6]. Recently, Memoli *et al.* [59] considered the problem of approximating (sketching) metric spaces using GW distance. Their goal was to approximate a (single) metric measure space modeling the underlying data by a smaller metric measure space. The work presented in this paper focuses on approximating a set of merge trees (modeled as a set of metric measure networks) with a smaller set of merge trees.

**Aligning and Averaging Graphs.** Graph alignment or graph matching is a key ingredient in performing comparisons and statistical analysis on the space of graphs (e.g., [33, 40]). It is often needed to establish node correspondences between graphs of different sizes. The works most relevant here are the ones that utilize Riemannian geometry in studying statistics on graphs. Jain and Obermayer [47] imposed a metric structure on the space of graphs for performing alignment and estimating the mean of a distribution on attributed graphs. Guo *et al.* [43, 42] utilized the metric structure in [47] to quantify graph differences and to

compute optimal deformations between graphs. They further established a framework for computing mean, covariance, and PCA of graphs. For merge trees specifically, Gasparovic *et al.* [37] studied the metric 1-center of a set of labeled merge trees, which found applications in visualization [77]. However, these works focused on aligning graph nodes or tree nodes over permutations, thus creating “hard matchings” [21]. On the other hand, the approaches based on the GW distances [64, 21] are rooted in measure theory and employ probabilistic or “soft matchings” of nodes. Information in a graph can be captured by a symmetric positive semidefinite matrix that encodes distances or similarities between pairs of nodes. Dryden *et al.* [28] described a way to perform statistical analysis and to compute the mean of such matrices. Agueh *et al.* [4] considered barycenters of several probability measures, whereas Cuturi *et al.* [23] and Benamou *et al.* [13] developed efficient algorithms to compute such barycenters. Peyre *et al.* [64] combined these ideas with the notion of GW distances [55] to develop GW averaging of distance/similarity matrices. Chowdhury and Needham [21] built upon the work in [64] and provided a GW framework to compute a Fréchet mean among these matrices using measure couplings. In this paper, we utilize the GW framework proposed in [21] for “soft matching” among merge trees.

**Matrix Sketching.** Many matrix sketching techniques build upon numerical linear algebra and vector sketching. For simplicity, we formulate the problem as follows: Given a  $d \times N$  matrix  $A$ , we would like to approximate  $A$  using fewer columns, as a  $d \times k$  matrix  $B$  such that  $A$  and  $B$  are considered to be *close* with respect to some problem of interest. Basic approaches for matrix sketching include truncated singular value decomposition (SVD), column or row sampling [26, 27], random projection [68], and frequent directions [38, 50]; see [65, 74] for surveys.

The column (or row) sampling approach carefully chooses a subset of the columns (or rows) of  $A$  proportional to their *importance*, where the importance is determined by the squared norm (*e.g.*, [26]) or the (approximated) leverage scores (*e.g.*, [27]). The random projection approach takes advantage of the Johnson-Lindenstrauss (JL) Lemma [48] to create an  $N \times k$  linear projection matrix  $S$  (*e.g.*, [68]), where  $B = AS$ . The frequent directions approach [50, 38] focuses on replicating properties of the SVD. The algorithm processes each column of  $A$  at a time while maintaining the best rank- $k$  approximation as the sketch.

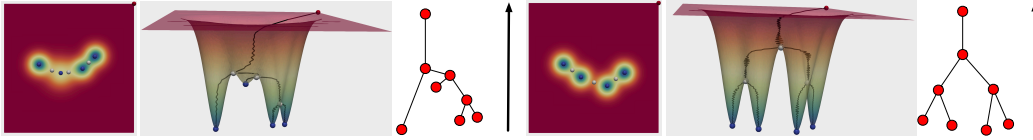
**Spanning Trees of Weighted Graphs.** Given an undirected, weighted graph  $G$ , a spanning tree is a subgraph of  $G$  that is a tree that connects all the vertices of  $G$  with a minimum possible number of edges. We consider two types of spanning trees: the *minimal spanning tree* (MST) and the *low stretch spanning tree* (LSST) [1, 2, 3]. Whereas the MST tries to minimize the sum of edge weights in the tree, LSST tries to minimize the stretch (relative distortion) of pairwise distances between the nodes of  $G$ . LSSTs were initially studied in the context of road networks [5]. They also play an important role in fast solvers for symmetric diagonally dominant (SDD) linear systems [31, 49].

A LSST does not stretch distances of  $G$  too much, in a sense that the distance between any two nodes in the tree is at most a small constant times the distance between the same nodes in  $G$ . Alon *et al.* [5] studied LSSTs in the context of the  $k$ -server problem on a road network and provided the first theoretical bound on the average stretch of LSSTs. Elkin *et al.* [31] used LSSTs to improve the running time for solving symmetric diagonally dominant (SDD) linear systems [69]. Koutis *et al.* [49] further improved upon the running time of LSST computation to provide a faster SDD solver. In a series of papers, Abraham and coauthors [1, 2, 3] proposed incrementally faster algorithms to compute LSSTs, while at the same time improving the bound on the average stretch. Most of these works focused on

bounding the average stretch. The problem of finding a spanning tree that minimizes the max stretch, referred to as the Minimum Max-Stretch spanning Tree (MMST), is known to be NP-hard. However, Emek and Peleg [32] presented an approximation algorithm for MMST which is a variation of the LSST.

### 3 Technical Background

We begin by reviewing the notion of a merge tree that arises from a scalar field. We then introduce the technical background needed to map a merge tree to a column vector in the data matrix. We primarily utilize the Gromov-Wasserstein (GW) framework of Chowdhury and Needham [21], which builds upon theoretical results on GW distances [54, 55], with a few ingredients from Peyre *et al.* [64].



**Figure 2** Two examples of merge trees from scalar (height) fields. For each example, from left to right: 2D scalar fields visualization, merge trees embedded in the graphs of the scalar fields, and abstract visualization of merge trees as rooted trees equipped with height functions.

**Merge Trees.** Let  $f : \mathbb{M} \rightarrow \mathbb{R}$  be a scalar field defined on the domain of interest  $\mathbb{M}$ , where  $\mathbb{M}$  can be a manifold or a subset of  $\mathbb{R}^d$ . For our experiments in Section 5,  $\mathbb{M} \subset \mathbb{R}^2$ . Merge trees capture the connectivity among the *sublevel sets* of  $f$ , *i.e.*,  $\mathbb{M}_a = f^{-1}(-\infty, a]$ . Formally, two points  $x, y \in \mathbb{M}$  are *equivalent*, denoted by  $x \sim y$ , if  $f(x) = f(y) = a$ , and  $x$  and  $y$  belong to the same connected component of a sublevel set  $\mathbb{M}_a$ . The *merge tree*,  $T(\mathbb{M}, f) = \mathbb{M}/\sim$ , is the quotient space obtained by gluing together points in  $\mathbb{M}$  that are equivalent under the relation  $\sim$ . To describe a merge tree procedurally, as we sweep the function value  $a$  from  $-\infty$  to  $\infty$ , we create a new branch originating at a leaf node for each local minimum of  $f$ . As  $a$  increases, such a branch is extended as its corresponding component in  $\mathbb{M}_a$  grows until it merges with another branch at a saddle point. If  $\mathbb{M}$  is connected, all branches eventually merge into a single component at the global maximum of  $f$ , which corresponds to the root of the tree. For a given merge tree, leaves, internal nodes, and root node represent the minima, merging saddles, and global maximum of  $f$ , respectively. Figure 2 displays a set of two scalar fields with their corresponding merge trees embedded in the graphs of the scalar fields. In a nutshell, a merge tree  $T$  is a rooted tree equipped with a scalar function defined on its node set,  $f : V \rightarrow \mathbb{R}$ .

**Gromov-Wasserstein Distance for Measure Networks.** The Gromov-Wasserstein (GW) distance was proposed by Memoli [54, 55] for metric measure spaces. Peyre *et al.* [64] introduced the notion of a *measure network* and defined the GW distance between such networks. The key idea is to find a *probabilistic matching* between a pair of networks by searching over the convex set of couplings of the probability measures defined on the networks.

A finite, weighted graph  $G$  can be represented as a measure network using a triple  $(V, W, p)$ , where  $V$  is the set of  $n$  nodes,  $W$  is a weighted adjacency matrix, and  $p$  is a probability measure supported on the nodes of  $G$ . For our experiments,  $p$  is taken to be uniform, that is,  $p = \frac{1}{n} \mathbf{1}_n$ , where  $\mathbf{1}_n = (1, 1, \dots, 1)^T \in \mathbb{R}^n$ .

Let  $G_1(V_1, W_1, p_1)$  and  $G_2(V_2, W_2, p_2)$  be a pair of graphs with  $n_1$  and  $n_2$  nodes, respectively. Let  $[n]$  denote the set  $\{1, 2, \dots, n\}$ .  $V_1 = \{x_i\}_{i \in [n_1]}$  and  $V_2 = \{y_j\}_{j \in [n_2]}$ . A *coupling*

between probability measures  $p_1$  and  $p_2$  is a joint probability measure on  $V_1 \times V_2$  whose marginals agree with  $p_1$  and  $p_2$ . That is, a coupling is represented as an  $n_1 \times n_2$  non-negative matrix  $C$  such that  $C\mathbf{1}_{n_2} = p_1$  and  $C^T\mathbf{1}_{n_1} = p_2$ . Given matrix  $C$ , its *binarization* is an  $n_1 \times n_2$  binary matrix, denoted as  $\mathbf{1}_{C>0}$ : this matrix has 1 where  $C > 0$ , and 0 elsewhere.

The *distortion* of a coupling  $C$  with an arbitrary loss function  $L$  is defined as [64]

$$\mathcal{E}(C) = \sum_{i,k \in [n_1], j,l \in [n_2]} L(W_1(i,k), W_2(j,l)) C_{i,j} C_{k,l}. \quad (1)$$

Let  $\mathcal{C} = \mathcal{C}(p_1, p_2)$  denote the collection of all couplings between  $p_1$  and  $p_2$ . The *Gromov-Wasserstein discrepancy* [64] is defined as

$$\mathcal{D}(C) = \min_{C \in \mathcal{C}} \mathcal{E}(C). \quad (2)$$

In this paper, we consider the quadratic loss function  $L(a, b) = \frac{1}{2}|a - b|^2$ . The *Gromov-Wasserstein distance* [21, 55, 64]  $d_{GW}$  between  $G_1$  and  $G_2$  is defined as

$$d_{GW}(G_1, G_2) = \frac{1}{2} \min_{C \in \mathcal{C}} \sum_{i,k \in [n_1], j,l \in [n_2]} |W_1(i,k) - W_2(j,l)|^2 C_{i,j} C_{k,l}. \quad (3)$$

It follows from the work of Sturm [71] that such minimizers always exist and are referred to as *optimal couplings*. Memoli [55] showed that  $(d_{GW})^{1/2}$  defines a distance on the “space of metric measure spaces quotient by measure-preserving isometries” [64].

**Alignment and Blowup.** Given a pair of graphs  $G_1$  and  $G_2$ , a coupling  $C \in \mathcal{C}(p_1, p_2)$  can be used to *align* their nodes. For each node  $x_i \in V_1$ , let  $u_i = |\{y_j \in V_2 : C(i, j) > 0\}|$  denote the number of nodes in  $V_2$  that have a nonzero coupling probability with  $x_i$ . We define a new node set  $V'_1 = \bigcup_{x_i \in V_1} \{(x_i, l) : 1 \leq l \leq u_i\}$ , which, roughly speaking, contains  $u_i$  copies of each  $x_i \in V_1$ . For  $x_i \in V_1$ , let  $\{y_{i,l}\}_{l \in [u_i]}$  denote the nodes in  $V_2$  such that  $C(x_i, y_{i,l}) > 0$ . We define a new distribution on  $V'_1$  as  $p'_1((x_i, l)) = C(i, y_{i,l})$ . Finally, for  $x, x' \in V_1$ ,  $l \in [u_x]$ , and  $l' \in [u_{x'}]$ , we define a new weight matrix  $W'$  such that  $W'_1((x, l), (x', l')) = W_1(x, x')$ . The new graph  $G'_1 = (V'_1, W'_1, p'_1)$  is the *blowup* of  $G_1$ .  $G'_1$  can be possibly enlarged *w.r.t.*  $G_1$ . Similarly, we can construct the blowup  $G'_2 = (V'_2, W'_2, p'_2)$  of  $G_2$ .

An optimal coupling  $C$  expands naturally to a coupling  $C'$  between  $p'_1$  and  $p'_2$ . After taking appropriate blowups,  $C'$  can be binarized to an  $n \times n$  permutation matrix (where  $n_1, n_2 \leq n$ ), and used to align the nodes of the two blown-up graphs. The GW distance is given by a formulation equivalent to Equation 3 based on an optimal coupling,

$$d_{GW}(G_1, G_2) = \frac{1}{2} \sum_{i,j} |W'_1(i, j) - W'_2(i, j)|^2 p'_1(i) p'_1(j). \quad (4)$$

**Fréchet Mean.** Given a collection of graphs  $\mathcal{G} = \{G_1, G_2, \dots, G_N\}$ , a *Fréchet mean* [21]  $\bar{G}$  of  $\mathcal{G}$  is a minimizer of the functional  $F(H, \mathcal{G}) = \frac{1}{N} \sum_{i=1}^N d_{GW}(G_i, H)$  over the space  $\mathcal{N}$  of measure networks,

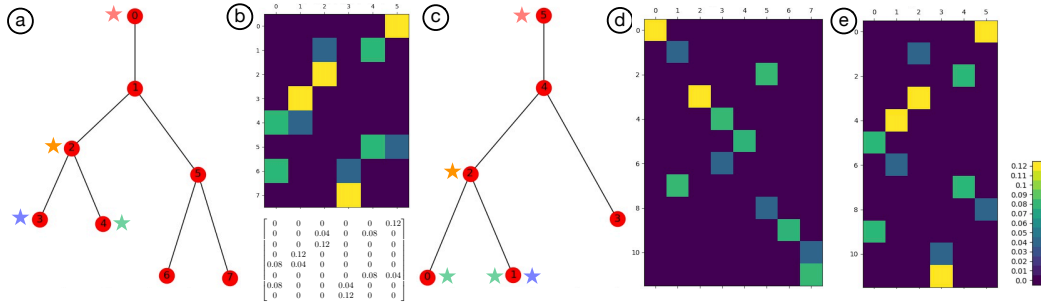
$$\bar{G} = \min_{H \in \mathcal{N}} \frac{1}{N} \sum_{i=1}^N d_{GW}(G_i, H). \quad (5)$$

Chowdhury and Needham [21] defined the directional derivative and the gradient of the functional  $F(H, \mathcal{G})$  at  $H$  and provided a gradient descent algorithm to compute the Fréchet mean. Their iterative optimization begins with an initial guess  $H_0$  of the Fréchet mean. At

the  $k^{\text{th}}$  iteration, there is a two-step process: each  $G_i$  is first blown-up and aligned to the current Fréchet mean,  $H_k$ ; then  $H_k$  is updated using the gradient of the functional  $F(H_k, \mathcal{G})$  at  $H_k$ . Such a two-step process is repeated until convergence where the gradient vanishes. For the complete algorithmic and implementational details, see [21]. If  $\bar{G} = (\bar{V}, \bar{W}, \bar{p})$  is the Fréchet mean, then we have

$$\bar{W}(i, j) = \frac{1}{N} \sum_{k=1}^N W'_k(i, j),$$

where  $W'_k$  is the weight matrix obtained by blowing-up and aligning  $G_k \in \mathcal{G}$  to  $\bar{G}$ . That is, when all the graphs in  $\mathcal{G}$  are blown-up and aligned to  $\bar{G}$ , the weight matrix of  $\bar{G}$  is given by a simple element-wise average of the weight matrices of the graphs.



**Figure 3** An optimal coupling between two simple merge trees (a) and (c). The coupling matrix is visualized in (b): yellows means high and dark blue means low probability. Couplings between the Fréchet mean  $\bar{T}$  with  $T_1$  and  $T_2$  are shown in (d) and (e), respectively.

**A Simple Example.** We give a simple example involving a pair of merge trees in Figure 3.  $T_1$  in (a) and  $T_2$  in (c) contain 8 and 6 nodes, respectively (nodes are labeled starting with a 0 index). The optimal coupling  $C$  obtained by the gradient descent algorithm of [21] is visualized in (b).  $C$  is an  $8 \times 6$  matrix, and it shows that node  $x_0$  in  $T_1$  is matched to node  $y_5$  in  $T_2$  with the highest probability (red stars). Similarly,  $x_2$  is matched with  $y_2$  (orange stars), and  $x_3$  is matched with  $y_1$  (blue stars), respectively. Node  $x_4$  in  $T_1$  is coupled with both  $y_0$  and  $y_1$  in  $T_2$  (green stars), with  $y_0$  having a higher probability (0.08) than  $y_1$  (0.04).

Now, we align both  $T_1$  and  $T_2$  to their Fréchet mean (denoted as  $\bar{T}$ ) via blown-ups.  $\bar{T}$  has 12 nodes. This gives rise to a coupling matrix between  $\bar{T}$  and  $T_1$  (of size  $12 \times 8$ ), and a coupling matrix between  $\bar{T}$  and  $T_2$  (of size  $12 \times 6$ ), respectively. As shown in Figure 3, node  $z_0$  of  $\bar{T}$  is matched with node  $x_0$  of  $T_1$  in (d) and node  $y_5$  of  $T_2$  in (e), respectively; and  $z_5$  is matched with  $x_4$  in (d) as well as  $y_0$  in (e). Now both trees  $T_1$  and  $T_2$  are blown-up to be  $T'_1$  and  $T'_2$ , each with 12 nodes, and can be vectorized into column vectors of the same size.

## 4 Methods

Given a set  $\mathcal{T}$  of  $N$  merge trees as input, our goal is to find a basis set  $\mathcal{S}$  such that each tree in  $\mathcal{T}$  can be approximately reconstructed from a linear combination of merge trees in  $\mathcal{S}$ . We propose to combine the GW framework [21] with techniques from matrix sketching to achieve this goal. We detail our pipeline to compute  $\mathcal{S}$ , as illustrated in Figure 1.

**Step 1: Representing Merge Trees as Metric Measure Networks.** The first step is to represent merge trees as metric measure networks as described in Section 3. Each merge tree

$T \in \mathcal{T}$  can be represented using a triple  $(V, W, p)$ , where  $V$  is the node set,  $W$  is a matrix of pairwise distances between its nodes, and  $p$  is a probability distribution on  $V$ .

In this paper, we define  $p$  as a uniform distribution, *i.e.*,  $p = \frac{1}{|V|} \mathbf{1}_{|V|}$ . To define  $W$ , recall that each node  $x \in V$  is associated with a scalar value  $f(x)$ . For a pair of adjacent nodes  $x, x' \in V$ ,  $W(x, x') = |f(x) - f(x')|$ , *i.e.*, the weight  $W(x, x')$  is the absolute difference in function value between them. We define  $W$  in such a way to encode information in  $f$ , which is inherent to a merge tree. For a pair of nonadjacent nodes  $x, x' \in V$ ,  $W(x, x')$  is the shortest path distance between them in  $T$ ; where a shortest path between  $x$  and  $x'$  by construction goes through their lowest common ancestor in  $T$ .

**Step 2: Merge Tree Vectorization via Blowup and Alignment to the Fréchet Mean.** The second step is to convert each merge tree into a column vector of the same size via blowup and alignment to the Fréchet mean. Having represented each merge tree as a metric measure network, we can use the GW framework to compute a Fréchet mean of  $\mathcal{T}$ , denoted as  $\bar{T} = (\bar{V}, \bar{W}, \bar{p})$ . Let  $n = |\bar{V}|$ . In theory,  $n$  may become as large as  $\prod_{i=1}^N |V_i|$ . In practice,  $n$  is chosen to be much smaller; in our experiments, we choose  $n$  to be a small constant factor (2 or 3) times the size of the largest input tree. The optimal coupling  $C$  between  $\bar{T}$  and  $T = T_i$  is an  $n \times n_i$  matrix with at least  $n$  nonzero entries. If the number of nonzero entries is greater than  $n$ , we keep only the largest value in each row. That is, if a node of  $\bar{T}$  has a nonzero probability of coupling with more than one node of  $T$ , we consider the mapping with only the highest probability, so that each coupling matrix  $C$  has exactly  $n$  nonzero entries. We then blow up each  $T$  to obtain  $T' = (V', W', p')$ , and align  $\bar{T}$  with  $T'$ . The above procedure ensures that each blown-up tree  $T'$  has exactly  $n$  nodes, and the binarized coupling matrix  $C'$  between  $\bar{T}$  and  $T'$  induces a node matching between them.

We can now vectorize (*i.e.*, flatten) each  $W'$  (an  $n \times n$  matrix) to form a column vector  $a \in \mathbb{R}^d$  of matrix  $A$  (where  $d = n^2$ ), as illustrated in Figure 1 (step 2)<sup>2</sup>. Each  $a$  is a vector representation of the input tree  $T$  with respect to the Fréchet mean  $\bar{T}$ .

**Step 3: Merge Tree Sketching via Matrix Sketching.** The third step is to sketch merge trees by applying matrix sketching to the data matrix  $A$ , as illustrated in Figure 1 (step 3). By construction,  $A$  is a  $d \times N$  matrix whose column vectors  $a_i$  are vector representations of  $T_i$ . We apply matrix sketching techniques to approximate  $A$  by  $\hat{A} = B \times Y$ . In our experiments, we use two linear sketching techniques, namely, column subset selection (CSS) and non-negative matrix factorization (NMF). See Section 6 for implementation details.

Using CSS, the basis set is formed by sampling  $k$  columns of  $A$ . Let  $B$  denote the matrix formed by  $k$  columns of  $A$  and let  $\Pi = BB^+$  denote the projection onto the  $k$ -dimensional space spanned by the columns of  $B$ . The goal of CSS is to find  $B$  such that  $\|A - \Pi A\|_F$  is minimized. We experiment with two variants of CSS.

In the first variant of CSS, referred to as *Length Squared Sampling (CSS-LSS)*, we sample (without replacement) columns of  $A$  with probabilities  $q_i$  proportional to the square of their Euclidean norms, *i.e.*,  $q_i = \|a_i\|_2^2 / \|A\|_F^2$ . We modify the algorithm slightly such that before selecting a new column, we factor out the effects from columns that are already chosen.

In the second variant of CSS, referred to as the *Iterative Feature Selection (CSS-IFS)*, we use the algorithm proposed by Ordozgoiti *et al.* [63]. Instead of selecting columns sequentially as in *CSS-LSS*, *CSS-IFS* starts with a random subset of  $k$  columns. Then each selected column is either kept or replaced with another column, based on the residual after the other selected columns are factored out simultaneously.

---

<sup>2</sup> In practice,  $d = (n + 1)n/2$  as we store only the upper triangular matrix.

In the case of NMF, the goal is to compute non-negative matrices  $B$  and  $Y$  such that  $\|A - \hat{A}\|_F = \|A - BY\|_F$  is minimized. We use the implementation provided in the decomposition module of the scikit-learn package [22, 35]. The algorithm initializes matrices  $B$  and  $X = Y^T$  and minimizes the residual  $Q = A - BX^T + b_j x_j^T$  alternately with respect to column vectors  $b_j$  and  $x_j$  of  $B$  and  $X$ , respectively, subject to the constraints  $b_j \geq 0$  and  $x_j \geq 0$ .

**Step 4: Reconstructing Sketched Merge Trees.** For the fourth step, we convert each column in  $\hat{A}$  as a sketched merge tree. Let  $\hat{A} = BY$ , where matrices  $B$  and  $Y$  are obtained using CSS or NMF. Let  $\hat{a} = \hat{a}_i$  denote the  $i^{\text{th}}$  column of  $\hat{A}$ . We reshape  $\hat{a}$  as an  $n \times n$  weight matrix  $\hat{W}'$ . We then obtain a tree structure  $\hat{T}'$  from  $\hat{W}'$  by computing its MST or LSST.

A practical consideration is the *simplification* of a sketched tree  $\hat{T}'$  coming from NMF.  $\hat{T}'$  without simplification is an approximation of the blow-up tree  $T'$ . It contains many more nodes compared to the original tree  $T$ . Some of these are internal nodes with exactly one parent node and one child node. In some cases, the distance between two nodes is almost zero. We further simplify  $\hat{T}'$  to obtain a final sketched tree  $\hat{T}$  by removing internal nodes and nodes that are too close to each other; see Section 6 for details.

**Step 5: Returning Basis Trees.** Finally, we return a set of basis merge trees  $\mathcal{S}$  using information encoded in the matrix  $B$ . Using CSS, each column  $b_j$  of  $B$  corresponds directly to a column in  $A$ ; therefore, the set  $\mathcal{S}$  is trivially formed by the corresponding merge trees from  $\mathcal{T}$ . Using NMF, we obtain each basis tree by applying MST or LSST to columns  $b_j$  of  $B$  with appropriate simplification, as illustrated in Figure 1 (step 5).

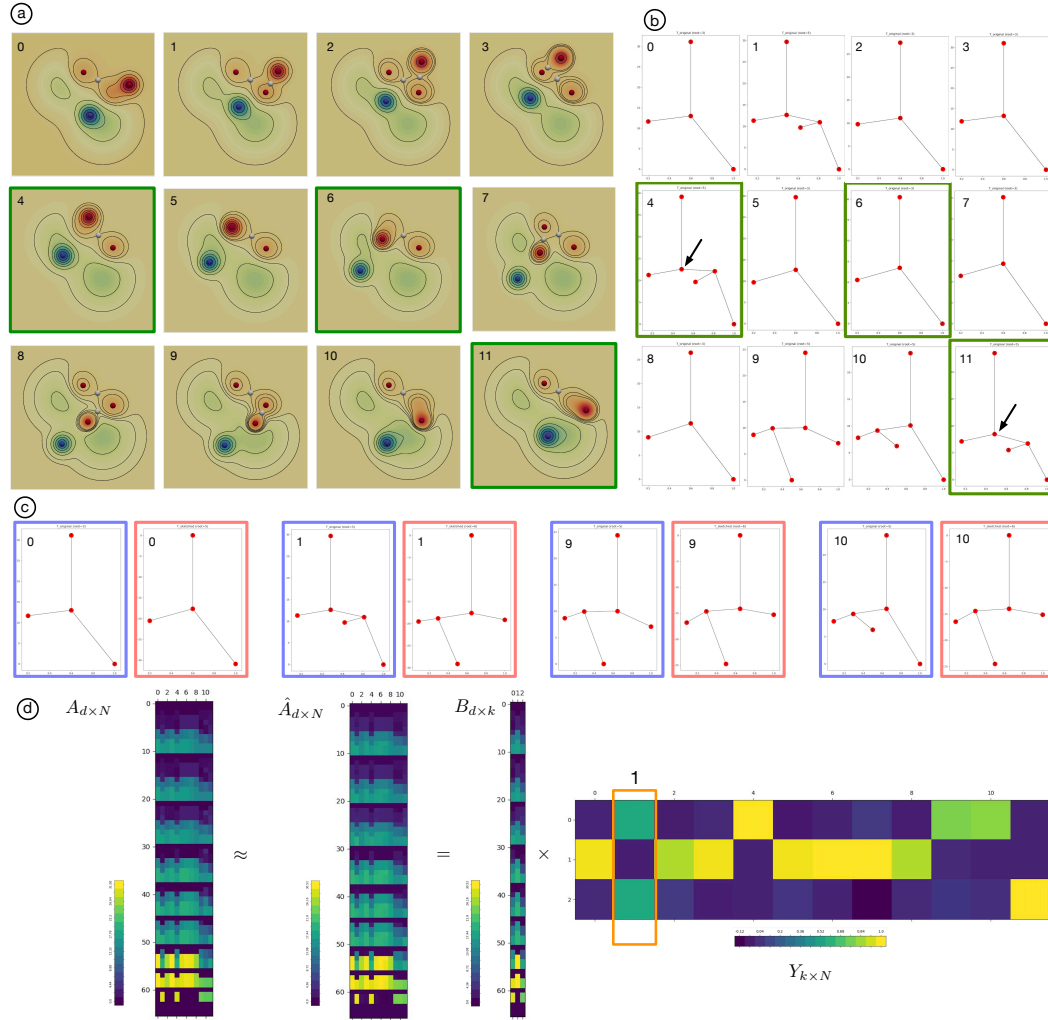
**Error Analysis.** For each of our experiments, we compute the *global sketch error*  $\epsilon = \|A - \hat{A}\|_F^2$ , as well as *column-wise sketch error*  $\epsilon_i = \|a_i - \hat{a}_i\|_2^2$ , where  $\epsilon = \sum_{i=1}^N \epsilon_i$ . By construction,  $\epsilon_i \leq \epsilon$ . For merge trees, we measure the GW distance between each tree  $T_i$  and its sketched version  $\hat{T}_i$ , that is  $\tau_i = d_{GW}(T_i, \hat{T}_i)$ , referred to as the *element-wise GW loss*. The *global GW loss* is defined to be  $\tau = \sum_{i=1}^N \tau_i$ . For theoretical considerations, see discussions in Section 7.

**A Simple Synthetic Example.** We give a simple synthetic example to illustrate our pipeline. A time-varying scalar field  $f$  is a mixture of 2D Gaussians that translate and rotate on the plane. We obtain a set of merge trees (indexed from 0) from 12 consecutive time steps, referred to as the *Rotating Gaussian*. In Figure 4(a) and (b), we show the scalar fields and the corresponding merge trees, respectively. Each merge tree is computed from  $-f$ ; thus, its leaves correspond to the local maxima (red), internal nodes are saddles (white), and the root node is the global minimum (blue) of  $f$  (see Figure 4(a)).

Since the dataset is quite simple, a few basis trees are sufficient to get good sketching results. With two basis trees ( $k = 2$ ), both CSS algorithms select  $\mathcal{S} = \{T_1, T_6\}$ . The topological structures of  $T_1$  and  $T_6$  are recognizably distinct among the input trees. For  $k = 3$ , *CSS-LSS* returns  $\mathcal{S} = \{T_1, T_6, T_{10}\}$ , while *CSS-IFS* gives  $\mathcal{S} = \{T_4, T_6, T_{11}\}$ . In Figure 4(a) and (b), we highlight the three basis trees selected with *CSS-IFS* and their corresponding scalar fields, respectively, with green boxes. These basis trees clearly capture rather unique structural representatives in the set. It is important to note that although  $T_4$  and  $T_{11}$  share a similar tree topology, they are of different heights ( $T_4$  has a height of 30, whereas  $T_{11}$  has a height of 24), and the branching nodes (black arrows) are positioned differently *w.r.t.* the root.

We also show a few input trees  $\{T_0, T_1, T_9, T_{10}\}$  (blue boxes) and their sketched versions (red boxes) in Figure 4(c). The input and the sketched trees are almost indistinguishable for  $T_0, T_1$ , and  $T_9$  (ignoring differences in tree layouts), but there are minor differences for  $T_{10}$ . We also visualize the data matrix  $A$ ,  $\hat{A}$ ,  $B$ , and highlight the coefficient matrix  $Y$  in Figure 4(d). The Fréchet mean tree  $\bar{T}$  contains 11 nodes. The matrix  $Y$  in Figure 4(d),



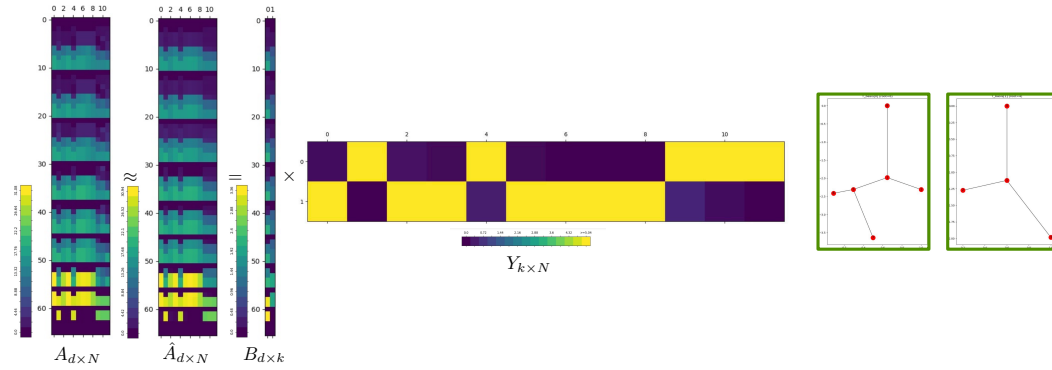


**Figure 4** *Rotating Gaussian: CSS-IFS with MST.* (a) Visualizing a time-varying mixture of Gaussian functions together with (b) their corresponding merge trees. (c) Examples of input merge trees (blue boxes) with their sketched versions (red boxes). (d) Visualizing data matrices associated with the sketching, while highlighting the coefficient matrix  $Y$ .

for instance, shows that  $T_1$  (orange box) is a linear combination of basis trees  $T_4$ ,  $T_6$ , and  $T_{11}$ , with coefficient 0.56,  $-0.01$ , and 0.57, respectively. In addition, columns in  $Y$  with high (yellow or light green) coefficients (*w.r.t.* the given basis) may be grouped together, forming clusters such as  $\{T_0, T_2, T_3, T_5, T_6, T_7, T_8\}$ , whose elements look structurally similar.

On the other hand, using NMF, when  $k = 2$ , we display the data matrices together with basis trees (obtained via MST) in Figure 5. The most interesting aspect of using NMF is that the basis trees (green boxes) are not elements of  $\mathcal{T}$ ; however, they very much resemble the basis trees obtained by CSS algorithms. In addition, columns in  $Y$  with high coefficients (*w.r.t.* the same basis tree) may be grouped together that show, for instance, structural similarities among the input trees  $T_1, T_4, T_9, T_{10}$ , and  $T_{11}$ .

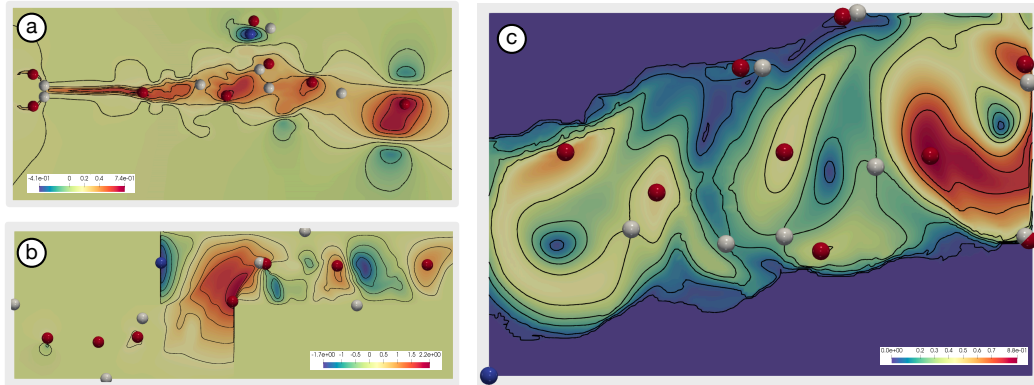




■ **Figure 5** *Rotating Gaussian*: data matrices together with basis trees returned by NMF.

## 5 Experimental Results

We demonstrate the applications of our merge tree sketching framework with three ensemble datasets from scientific simulations. The key takeaway is that for each ensemble dataset, our framework obtains basis merge trees that capture structural transitions and structural diversity among the ensemble members. In general, CSS gives better sketched trees than NMF; *Iterative Feature Selection (CSS-IFS)* performs slightly better than *Length Squared Sampling (CSS-LSS)*, based on error analysis. MST gives more visually appealing trees in practice than LSST. All source codes and data will be made publicly available.



■ **Figure 6** Visualizing a scalar field  $f$  that arises from a vorticity field of (a) *Heated Cylinder* dataset, (b) *Corner Flow* dataset, and a velocity magnitude field of (c) *Red Sea* dataset, respectively. Critical points (after de-noising) are shown in red (maxima) and white (saddles). Global minima are in blue. Land is colored blue in (c). All merge trees are computed from  $-f$ .

**Heated Cylinder Dataset.** Two of our datasets come from numerical simulations available online<sup>3</sup>. The first dataset, referred to as the *Heated Cylinder with Boussinesq Approximation (Heated Cylinder in short)*, comes from the simulation of a 2D flow generated by a heated cylinder using the Boussinesq approximation [41, 66]. The dataset shows a time-varying turbulent plume containing numerous small vortices. We convert each time instance of the flow into a scalar field using its vorticity. We generate a set of merge trees from these scalar

<sup>3</sup> <https://cgl.ethz.ch/research/visualization/data.php>

fields based on 31 time steps (they correspond to steps 600-630 from the original 1310 time steps). This set captures the evolution of small vortices over time. An instance of the scalar field after de-noising is illustrated in Figure 6(a).

**Corner Flow Dataset.** The second dataset, referred to as the *Cylinder Flow Around Corners* (*Corner Flow* in short), arises from the simulation of a viscous 2D flow around two cylinders [7, 66]. The channel into which the fluid is injected is bounded by solid walls. A vortex street is initially formed at the lower left corner, which then evolves around the two corners of the bounding walls. We generate a set of merge trees from the vorticity fields of the first 100 time instances, which describe the formation of a one-sided vortex street on the upper right corner; see Figure 6(b) for an example.

**Red Sea Dataset.** The third dataset, referred to as the *Red Sea eddy simulation* (*Red Sea* in short) dataset, originates from the IEEE Scientific Visualization Contest 2020<sup>4</sup>. The dataset was generated using a high-resolution MITgcm (Massachusetts Institute of Technology general circulation model), together with remote sensing satellite observations. It is used to study the circulation dynamics and eddy activities of the Red Sea (see [46, 78, 79]). For our analysis, we use merge trees that arise from velocity magnitude fields of an ensemble (named *001.tgz*) with 60 time steps. Figure 6(c) shows a latter time step that captures the formation of various eddies, which are circular movements of water important for transporting energy and biogeochemical particles in the ocean.

## 5.1 Heated Cylinder Dataset

Given merge trees  $\mathcal{T} = \{T_0, \dots, T_{30}\}$  from the *Heated Cylinder* dataset, we apply both CSS and NMF to obtain a set of basis trees  $\mathcal{S}$  and reconstruct the sketched trees. We compare all sketching techniques by setting  $k = 5$ , all of which give fairly good sketching results.

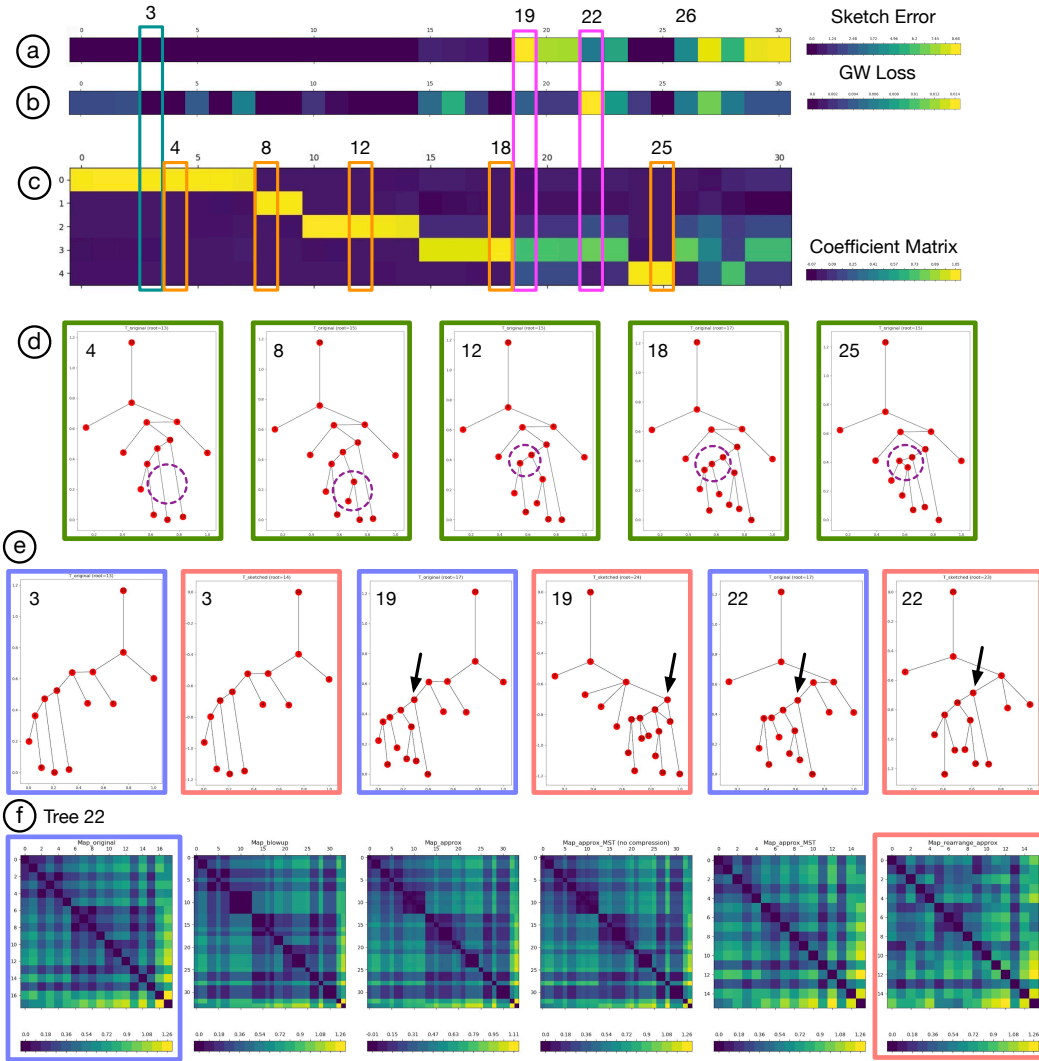
**Coefficient Matrices and Basis Trees from CSS and NMF.** As shown in Figure 7, *CSS-IFS* produces five basis trees:  $T_4$ ,  $T_8$ ,  $T_{12}$ ,  $T_{18}$ , and  $T_{25}$ . Figure 7(d) visualizes these basis trees while highlighting their structural differences (purple circles). The coefficient matrix  $Y$  in Figure 7(c) contains a number of yellow or light green blocks, indicating that consecutive input trees are grouped together into clusters, and represented by one or two shared basis. The columns of  $Y$  corresponding to the basis trees are highlighted with orange boxes in Figure 7(c). The basis trees appear to be good representatives of the clusters.

Further investigation of the input trees reveals that these basis trees capture structural transitions in the underlying scalar fields, even though such changes are not easy to detect directly from their corresponding scalar fields (see Figure 9). In Figure 8, we see noticeable structural transitions (purple circles) among trees from adjacent time steps:  $T_7 \rightarrow T_8$ ,  $T_{14} \rightarrow T_{15}$ ,  $T_{23} \rightarrow T_{24}$ , etc. The input trees with similar structures can be clustered based on such transitions, where the basis trees are selected as representatives of each cluster.

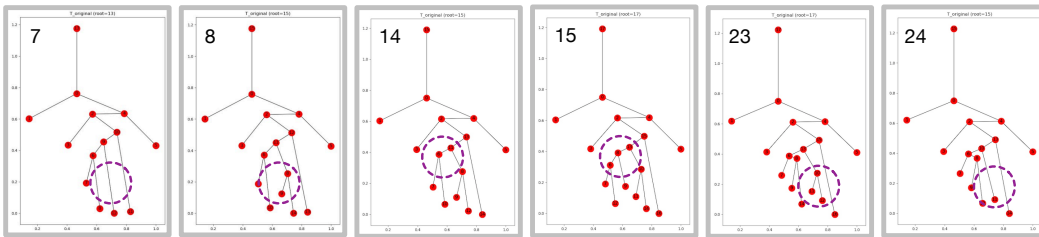
On the other hand, *CSS-LSS* gives five basis trees,  $\mathcal{S} = \{T_2, T_8, T_{19}, T_{25}, T_{30}\}$ , which are quite similar to the results of *CSS-IFS*, as shown in Figure 10. Using NMF, we show the five basis trees together with coefficient matrix  $Y$  in Figure 11. It is interesting to notice the star-like basis trees, although it is unclear if there is an intuitive explanation.

**Sketched Trees.** As illustrated in Figure 7(a) and (b), the column-wise sketch error and the element-wise GW loss can be used to guide our investigation into individual trees. Using

<sup>4</sup> <https://kaust-vislab.github.io/SciVis2020/>

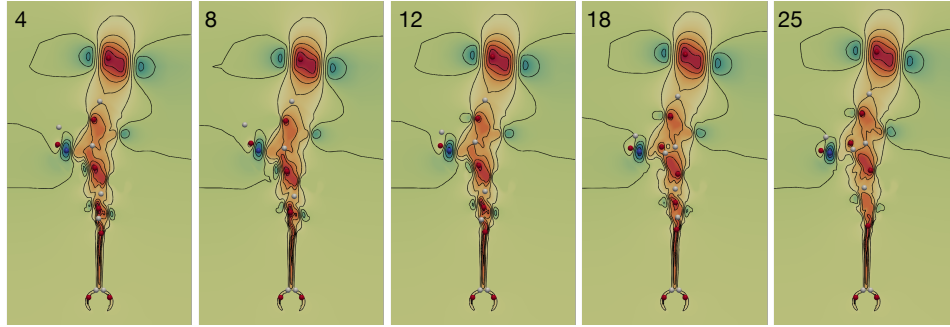


**Figure 7** *Heated Cylinder*, *CSS-IFS* with *MST*: (a) column-wise sketch error, (b) element-wise *GW* loss, (c) coefficient matrix  $Y$ , (d) basis trees, (e) input trees (blue boxes) with their sketched versions (red boxes), (f) weight matrices associated with  $T_{22}$  during the sketching process.

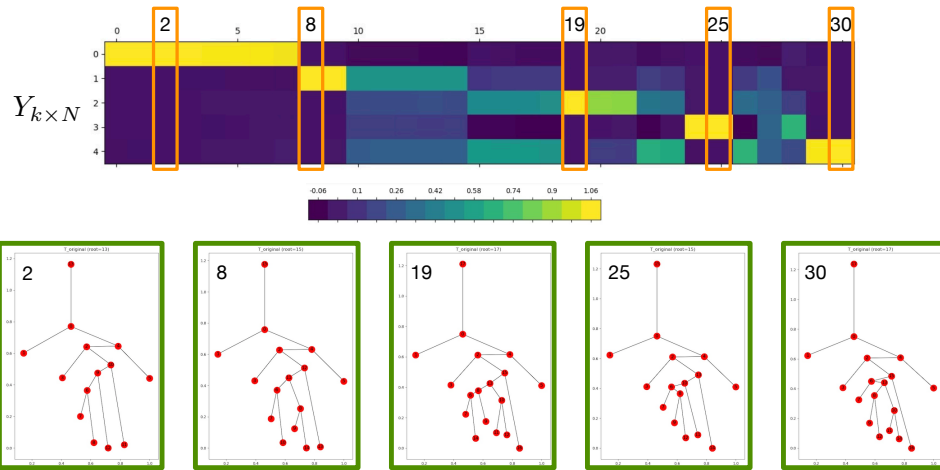


**Figure 8** *Heated Cylinder*: *CSS-IFS* with *MST*. Instances of structural transitions among consecutive merge trees.

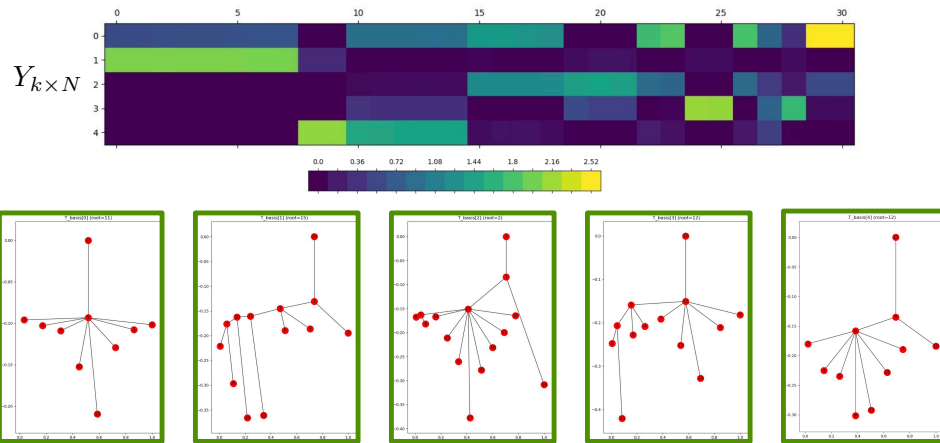
*CSS-IFS*, we give an example of a well-sketched tree ( $T_3$ ) with a couple of outliers ( $T_{19}$  and  $T_{22}$ ) based on the error analysis. In Figure 7(e), we compare each input tree (blue box)



■ **Figure 9** *Heated Cylinder*: scalar fields that correspond to the basis trees selected by *CSS-IFS*.



■ **Figure 10** *Heated Cylinder*: *CSS-LSS* with MST.



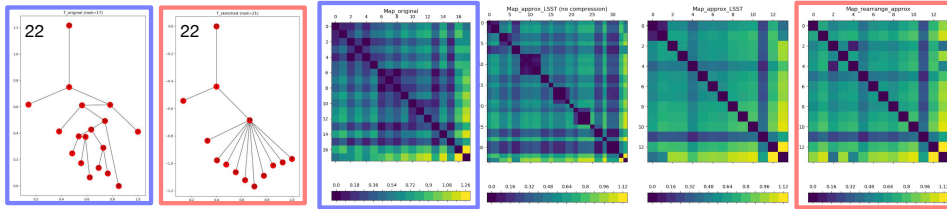
■ **Figure 11** *Heated Cylinder*: NMF with MST.

against its sketched version (red box).  $T_3$  and its sketched version  $\hat{T}_3$  are indistinguishable. Even though  $T_{19}$  and  $T_{22}$  are considered outliers relative to other input trees, they both share similar subtrees (whose roots are pointed by black arrows) with their sketched versions.

In Figure 7(f), we further investigate the weight matrices from different stages of the

sketching pipeline for  $T_{22}$ . From left to right, we show the weight matrix  $W_{22}$  of the input tree, its blow-up matrix  $W'_{22}$  (which is linearized to  $a_{22}$ ), the approximated column vector  $\hat{a}_{22}$  after sketching (reshaped into a square matrix), the weight matrix  $\hat{W}'_{22}$  of the MST derived from the reshaped  $\hat{a}_{22}$ , the weight matrix of the MST after simplification, and root alignment  $\hat{W}_{22}$  w.r.t.  $T_{22}$ . We observe minor changes between  $W_{22}$  (blue box) and  $\hat{W}_{22}$  (red box), which explains the structural variation between  $T_{22}$  and  $\hat{T}_{22}$ .

**Reconstructing Merge Trees with LSST.** For comparative purposes, instead of MST, we demonstrate the tree reconstruction results using LSST for *CSS-IFS*, as shown in Figure 12. The reconstructed tree using LSST for  $T_{22}$  (blue box) is visually less appealing compared to the reconstruction using MST. The star-like features are most likely a consequence of the petal decomposition algorithm of LSST. However, the weight matrix view shows that the LSST preserves distances fairly well (red box vs. blue box).



■ **Figure 12** *Heated Cylinder: CSS-IFS with LSST.*

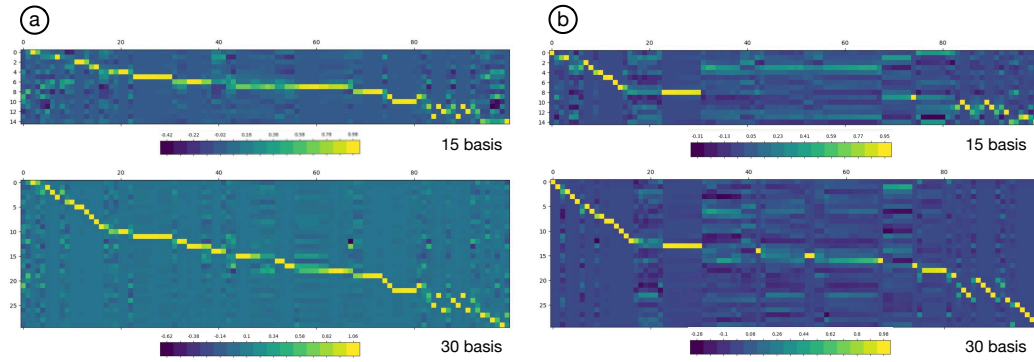
**Error Analysis.** Finally, we discuss our results quantitatively. The global sketch error for *CSS-IFS*, *CSS-LSS*, and NMF is 70.17, 93.22, and 53.91, respectively. Using MST, the global GW loss for *CSS-IFS*, *CSS-LSS*, and NMF is 0.15, 0.25, and 0.33, respectively. Using LSST, the global GW loss for *CSS-IFS*, *CSS-LSS*, and NMF is 0.39, 0.35, and 0.47, respectively. Therefore, for this particular dataset, NMF gives the best matrix approximation but less visually appealing merge trees, while *CSS-IFS* performs best among the three sketching techniques in terms of preserving tree topology measured by the GW distance.

## 5.2 Corner Flow Dataset

The *Corner Flow* dataset contains 100 merge trees. We start with an error analysis with the coefficient matrices, and then describe the sketching results in detail.

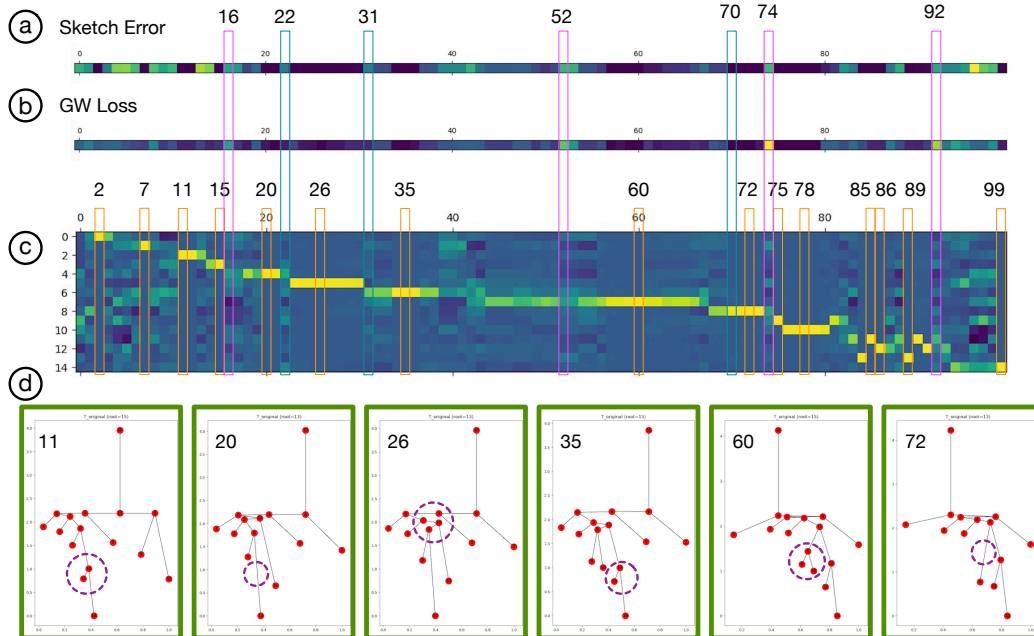
**Error Analysis with Coefficient Matrices.** First, we compare the coefficient matrices generated using both CSS algorithms, for  $k = 15$  and  $k = 30$ , respectively. In Figure 13(a), the matrix  $Y$  for *CSS-IFS* contains a larger number of yellow rows (indicating large coefficients) in comparison with Figure 13(b), which means that consecutive input trees can be reconstructed with a small number of shared basis. This result implies that *CSS-IFS* produces basis (columns) that are better cluster representatives than *CSS-LSS*. In terms of errors, *CSS-IFS* has a smaller global GW loss: 14.41 ( $k = 15$ ) and 10.90 ( $k = 30$ ) for *CSS-IFS*, and 23.35 ( $k = 15$ ) and 16.21 ( $k = 30$ ) for *CSS-LSS*, respectively. *CSS-IFS* also has a smaller (6886.91) global sketch error than *CSS-LSS* (11641.56), for  $k = 15$ . The column-wise sketch error is visualized in Figure 14(a), and the element-wise GW loss is shown in Figure 14(b).

**Basis Trees.** We report the sketching results with 15 basis under *CSS-IFS*. The basis trees are  $T_2, T_7, T_{11}, T_{15}, T_{20}, T_{26}, T_{35}, T_{60}, T_{72}, T_{75}, T_{78}, T_{85}, T_{86}, T_{89},$  and  $T_{99}$ ; see Figure 14(c-d). Similar to the *Heated Cylinder* dataset, we observe noticeable structural transitions among some pairs of adjacent input trees, which partition the set into clusters with similar



■ **Figure 13** *Corner Flow* dataset: *CSS-IFS* (a) and *CSS-LSS* (b) with MST. Visualizing weight matrices  $Y$  with 15 and 30 basis trees, respectively.

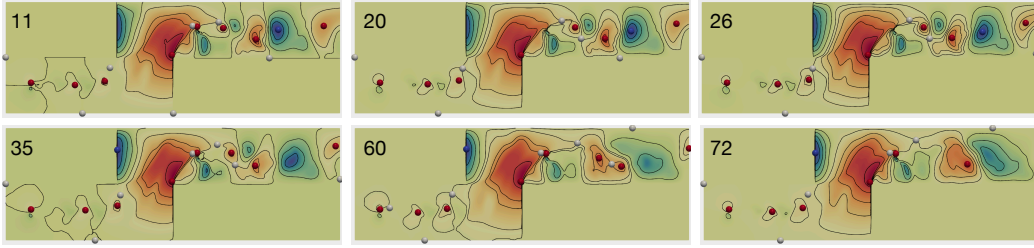
structures (blocks of yellow). As illustrated in Figure 14(c), the basis trees (orange boxes) serve as good cluster representatives, as they are mostly selected among yellow blocks. In particular, they capture structural transitions in the set, which are highlighted in purple circles in Figure 14(d). Such structural transitions may not be obvious in their scalar fields counterparts; see Figure 15.



■ **Figure 14** *Corner Flow*: *CSS-IFS* with MST. Visualizing column-wise sketch error (a), element-wise GW loss (b), weight matrix  $Y$  with weights associated with basis trees, and (d) a subset of basis trees. Purple circles highlight structural changes between basis trees.

**Outliers among Sketched Merge Trees.** Finally, we investigate individual sketched trees. We utilize the column-wise sketch error in Figure 14(a) with element-wise GW loss in Figure 14(b) to select outliers among the sketched trees, such as  $T_{16}$ ,  $T_{52}$ ,  $T_{74}$ , and  $T_{92}$  (magenta boxes). Figure 14(c) shows that such outlier trees are linear combinations of basis trees with fairly uniform weights. In Figure 16(b), we observe that these outlier trees are less visually

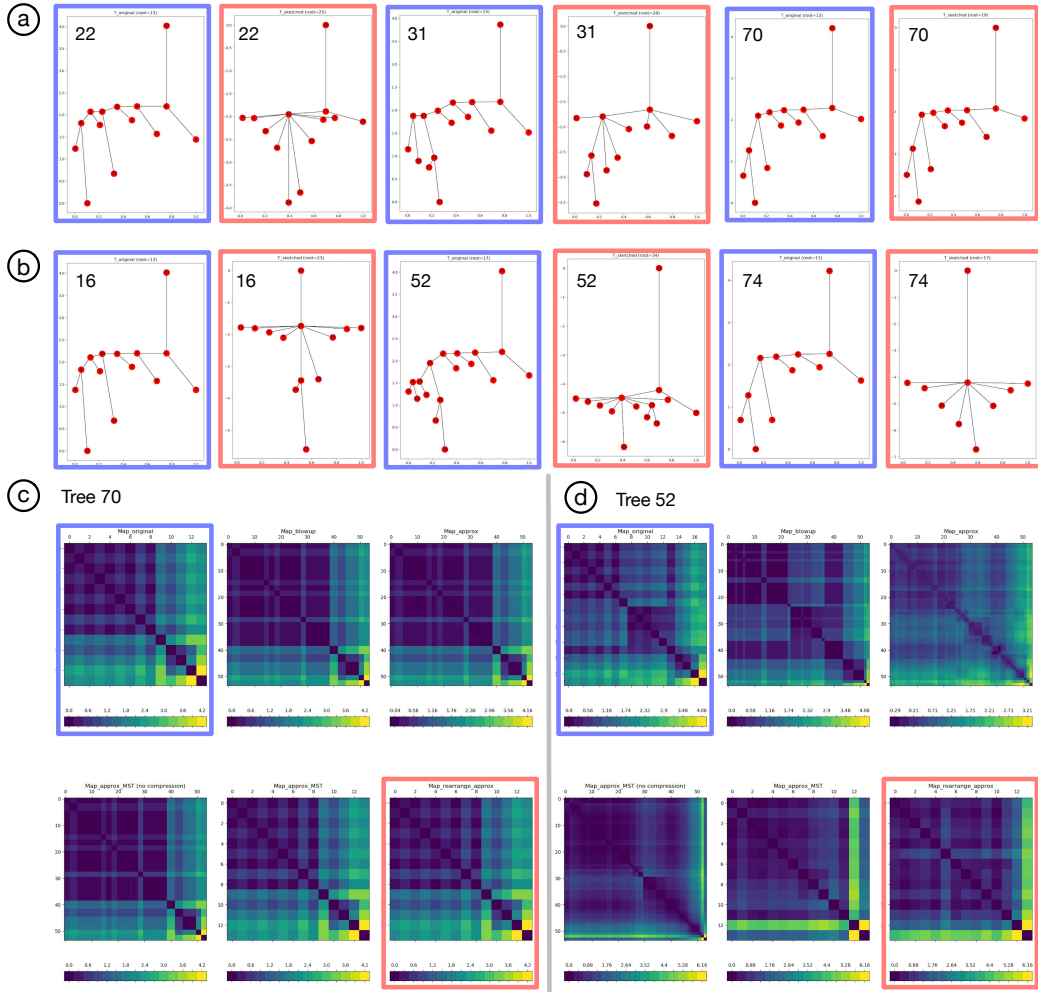




■ **Figure 15** *Corner Flow: CSS-IFS with MST.* Scalar fields that correspond to the basis.

appealing, where their sketched versions (red boxes) contain a number of star-like features. For instance, Figure 16(d) shows noticeable amount of changes in comparing the merge tree weight matrices for  $T_{52}$  (blue box) and  $\hat{T}_{52}$  (red box).

On the other hand, in Figure 16(a), trees with lower element-wise GW loss are structurally similar to basis trees, and thus have a good approximation of their topology. For instance, the sketched tree  $\hat{T}_{70}$  (red box) is indistinguishable *w.r.t.* to the original input  $T_{70}$ .



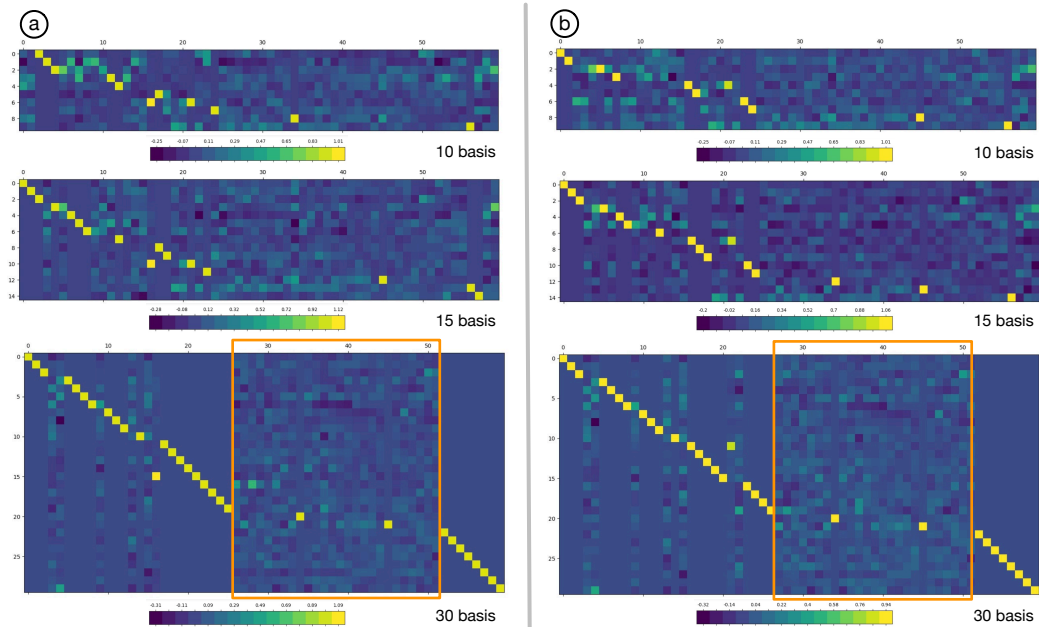
■ **Figure 16** *Corner Flow: CSS-IFS with MST.* Individual sketched trees with high and low errors.

### 5.3 Red Sea Dataset

The *Red Sea* dataset comes with 60 merge trees. The input set does not exhibit natural clustering structures because many adjacent time steps give rise to trees with noticeable structural changes. Nevertheless, we report our sketching results and discuss how such results improve with an increasing number of basis.

**Coefficient Matrices.** Using both *CSS-IFS* and *CSS-LSS*, we compare the coefficient matrices  $Y$  for  $k = 10, 15$  and  $30$ , respectively; see Figure 17. For  $k = 10$ ,  $\mathcal{S}$  contains  $T_2, T_3, T_4, T_{11}, T_{12}, T_{17}, T_{21}, T_{24}, T_{34}, T_{56}$ . For  $k = 15$ ,  $\mathcal{S}$  includes  $T_0, T_1, T_2, T_4, T_6, T_7, T_8, T_{12}, T_{17}, T_{18}, T_{21}, T_{23}, T_{45}, T_{56}$ , and  $T_{57}$ . Since *CSS-IFS* is not a deterministic algorithm, as we increase  $k$ , the basis set for  $k = 10$  does not necessarily form a subset of the basis set for  $k = 15$ . The input trees appear to have diverse structures, since the clustering among them is unclear. This phenomenon is evident by the lack of long yellow rows in the matrices  $Y$ . It is also interesting to notice that for both CSS algorithms, there exists a subset of consecutive columns that contain few selected basis (orange boxes for  $k = 30$ ).

In general, the global sketch error and GW loss improve as we increase the number of basis. With 15 basis, the global GW loss for *CSS-IFS*, *CSS-LSS*, and NMF is 1.23629, 1.52865, and 6.91503, respectively; the global sketch error is 1385.71, 1408.21, and 1115.27, respectively. Globally, therefore, NMF best approximates the matrix, whereas *CSS-IFS* best preserves the tree topology.

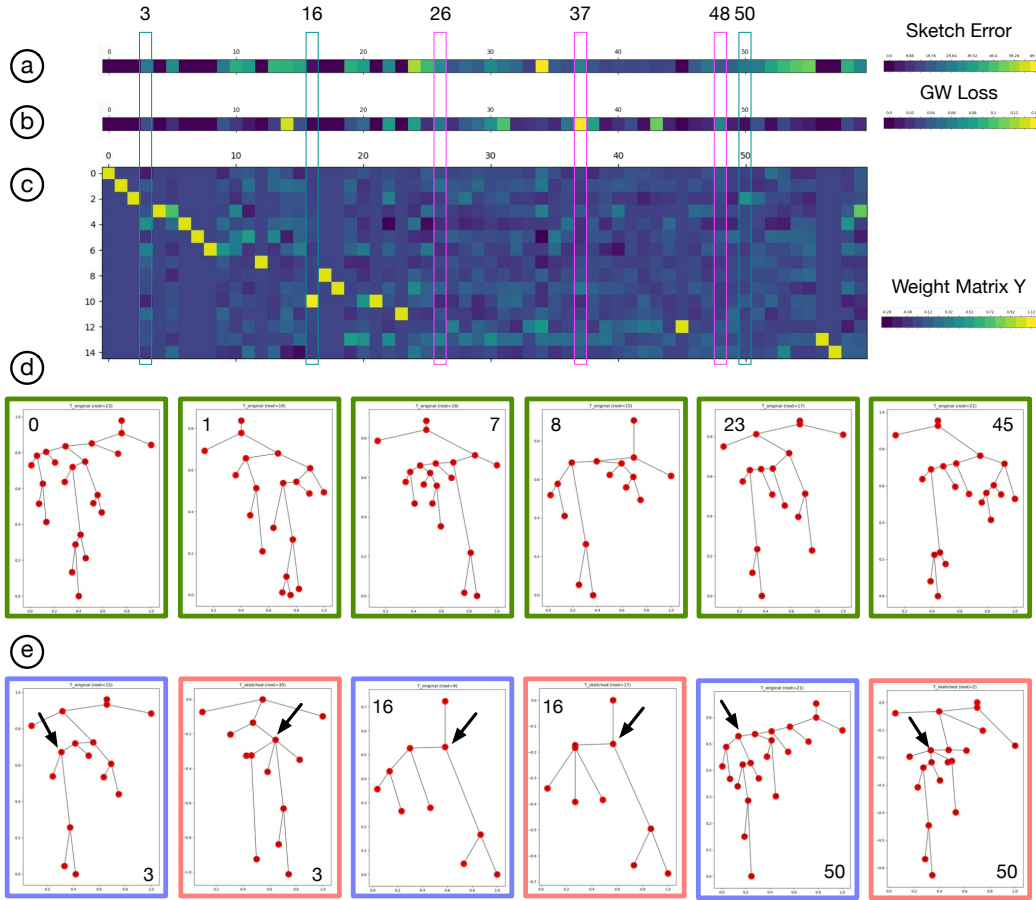


■ **Figure 17** *Red Sea* dataset: *CSS-IFS* (a), *CSS-LSS* (b). Visualizing weight matrix  $Y$  with 10, 15, and 30 basis trees, respectively.

**Basis Trees and Sketched Trees.** We now discuss basis trees obtained using *CSS-IFS* with 15 basis. In Figure 18(d), we show a subset of the basis trees,  $T_0, T_1, T_7, T_8$ , and  $T_9$  to demonstrate the diverse structures. We include column-wise sketch errors and element-wise GW losses in Figure 18(a) and Figure 18(b), respectively. Similar to the *Corner Flow* dataset, we could use these errors to study examples of well-sketched trees (green boxes) and outliers (magenta boxes). Well-sketched trees are those with low GW losses and sketch errors, like



$T_3$ ,  $T_{16}$ , and  $T_{50}$ . Given the diversity of the input trees, with just 15 basis, we still observe very similar subtree structures among these trees (whose roots are pointed by black arrows), see Figure 18(e).



**Figure 18** Red Sea dataset: *CSS-IFS*,  $k = 15$ . Visualizing column-wise sketch error (a), element-wise GW loss (b), weight matrix  $Y$  (c), a subset of basis trees (d), and examples of well-sketched trees (e). Basis trees are encoded by green boxes. Input trees and their sketched version are encoded by blue and red boxes, respectively.

## 6 Implementation Details

In this section, we provide some implementation details for various algorithms employed in our merge tree sketching framework.

**Matrix Sketching Algorithms.** We use two variants of column subset selection (*CSS*) algorithms, as well as non-negative matrix factorization (*NMF*) to sketch the data matrix  $A$ . Here, we provide pseudocode for these matrix sketching algorithms.

- Modified Length Squared Sampling (*CSS-LSS*)
  1.  $s \leftarrow 0$ ,  $B$  is an empty matrix,  $A' = A$ .
  2.  $s \leftarrow s + 1$ . Select column  $c$  from  $A'$  with the largest squared norm (or select  $c$  randomly proportional to the squared norm) and add it as a column to  $B$ . Remove  $c$  from  $A'$ .

3. For each remaining column  $c'$  in  $A'$  (i.e.,  $c' \neq c$ ), factor out the component along  $c$  as:
    - a.  $u \leftarrow c/\|c\|$
    - b.  $c' \leftarrow c' - \langle u, c' \rangle u$
  4. While  $s < k$ , go to step 2.
- Iterative Feature Selection (*CSS-IFS*)
1. Choose a subset of  $k$  column indices  $r = \{i_1, i_2, \dots, i_k\}$  uniformly at random.
  2. Construct subset  $B_r = [a_{i_1}, a_{i_2}, \dots, a_{i_k}]$  of  $A$  with columns indexed by  $r$ .
  3. Repeat for  $j = 1, 2, \dots, k$ :
    - a. Let  $X_{jl}$  denote matrix formed by replacing column  $a_{i_j}$  with column  $a_l$  in  $B_r$ , where  $l \in [n] \setminus r$ . Let  $X_{jl}^+$  denote its Moore-Penrose pseudoinverse.
    - b. Find  $w = \operatorname{argmin}_{l \in [n] \setminus r} \|A - X_{jl} X_{jl}^+ A\|_F$ .
    - c.  $B_r \leftarrow X_{jw}$ .
    - d.  $r \leftarrow (r \setminus \{i_j\}) \cup \{w\}$ .

- Non-Negative Matrix Factorization (NMF)
1. Given  $A$  and  $k$ , initialize  $B \in \mathbb{R}^{d \times k}$ ,  $Y = X^T \in \mathbb{R}^{k \times N}$  using the non-negative double singular value decomposition algorithm of Boutsidis and Gallopoulos [14].
  2. Normalize columns of  $B$  and  $X$  to unit  $L_2$  norm. Let  $E = A - BX^T$ .
  3. Repeat until convergence: for  $j = 1, 2, \dots, k$ ,
    - a.  $Q \leftarrow E + b_j x_j^T$ .
    - b.  $x_j \leftarrow [Q^T b_j]_+$ .
    - c.  $b_j \leftarrow [Q x_j]_+$ .
    - d.  $b_j \leftarrow b_j / \|b_j\|$ .
    - e.  $E \leftarrow Q - b_j x_j^T$ .

Here,  $[Q]_+$  means that all negative elements of the matrix  $Q$  are set to zero.

**LSST Algorithm.** We construct low stretch spanning trees (LSST) using the petal decomposition algorithm of Abraham and Neiman [3]. Given a graph  $G$ , its LSST is constructed by recursively partitioning the graph into a series of clusters called *petals*. Each petal  $P(x_0, t, r)$  is determined by three parameters: the center of the current cluster  $x_0$ , the target node of the petal  $t$ , and the radius of the petal  $r$ .

A cone  $C(x_0, x, r)$  is the set of all nodes  $v$  such that  $d(x_0, x) + d(x, v) - d(x_0, v) \leq r$ . A petal is defined as a union of cones of varying radii. Suppose  $x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_k = t$  is the sequence of nodes on the shortest path between nodes  $x_0$  and  $t$ . Let  $d_k$  denote the distance  $d(x_k, t)$ . Then the petal  $P(x_0, t, r)$  is defined as the union of cones  $C(x_0, x_k, (r - d_k)/2)$  for all  $x_k$  such that  $d_k \leq r$ .

Beginning with a vertex  $x_0$  specified by the user, the algorithm partitions the graph into a series of petals. When no more petals can be obtained, all the remaining nodes are put into a cluster called the stigma. A tree structure, rooted in the stigma, is constructed by connecting the petals and the stigma using some of the intercluster edges. All other edges between clusters are dropped. This process is applied recursively within each petal (and the stigma) to obtain a spanning tree structure.

**Merge Tree Simplification.** To reconstruct a sketched tree, we reshape the sketched column vector  $\hat{a}$  of  $\hat{A}$  into an  $n \times n$  matrix  $\hat{W}'$ , and obtain a tree structure  $\hat{T}'$  by computing its MST or LSST.  $\hat{T}'$  is an approximation of the blow-up tree  $T'$ . To get a tree approximation closer to the original input tree  $T$ , we further simplify  $\hat{T}'$  as described below.

The simplification process has two parameters. The first parameter  $\alpha$  is used to merge internal nodes that are too close ( $\leq \alpha$ ) to each other. Let  $R$  be the diameter of  $\hat{T}'$  and  $n$

the number of nodes in  $\hat{T}'$ .  $\alpha$  is set to be  $c_\alpha R/n^2$  for  $c_\alpha \in \{0.5, 1, 2\}$ . A similar parameter was used in simplifying LSST in [3]. The second parameter  $\beta = c_\beta R/n$  is used to merge leaf nodes that are too close ( $\leq \beta$ ) to the parent node, where  $c_\beta \in \{0.5, 1, 2\}$ . Let  $\hat{W}'$  be the weight matrix of  $\hat{T}'$ . The simplification process is as follows:

1. Remove from  $\hat{T}'$  all edges  $(u, v)$  where  $\hat{W}'(u, v) \leq \alpha$ .
2. Merge all leaf nodes  $u$  with their respective parent node  $v$  if  $\hat{W}'(u, v) \leq \beta$ .
3. Remove all the internal nodes.

The tree  $\hat{T}$  obtained after simplification is the final sketched tree.

**Merge Tree Layout.** To visualize both input merge trees and sketched merge trees, we experiment with a few strategies. To draw an input merge tree  $T$  equipped with a function defined on its nodes,  $f : V \rightarrow \mathbb{R}$ , we set each node  $u \in V$  to be at location  $(x_u, y_u)$ ; where  $y_u = f(u)$ , and  $x_u$  is chosen within a bounding box while avoiding edge intersections. The edge  $(u, v)$  is drawn proportional to its weight  $W(u, v) = |f(u) - f(v)| = |y_u - y_v|$ .

To draw a sketched tree as a merge tree, we perform the following steps:

1. Fix the root of the sketched tree at  $(0, 0)$ .
2. The y-coordinate of each child node is determined by the weight of the edge between the node and its parent.
3. The x-coordinate, which determines the left-to-right ordering of the child nodes, is computed using a heuristic strategy described below.
  - a. Sort the child nodes by their distance to the parent node in descending order.
  - b. Suppose the order of child nodes after sorting is  $c_1, c_2, \dots, c_t$ . If  $t$  is odd, we reorder the nodes from left to right as  $c_t, c_{t-2}, c_{t-4}, \dots, c_3, c_1, c_2, c_4, \dots, c_{t-3}, c_{t-1}$ . If  $t$  is even, we reorder the nodes as  $c_{t-1}, c_{t-3}, c_{t-5}, \dots, c_3, c_1, c_2, c_4, \dots, c_{t-2}, c_t$ .

The idea is to keep the child nodes that have a larger distance to the parent near the center to avoid edge crossings between sibling nodes and their subtrees.

Our layout strategy assumes that the trees are rooted. However,  $\hat{T}$ , which is our approximation of  $T$ , is not rooted. In our experiments, we use two different strategies to pick a root for  $\hat{T}$  and align  $T$  and  $\hat{T}$  for visual comparison.

Using the **balanced layout** strategy, we pick the node  $u$  of  $\hat{T}$  that minimizes the sum of distances to all other nodes. Set  $u$  to be the *balanced root* of  $\hat{T}$ . Similarly, we find the balanced root  $v$  of the input tree  $T$ .  $T$  and  $\hat{T}$  are drawn using the balanced roots.

Using the **root alignment** strategy, we compute the optimal coupling between  $T$  and  $\hat{T}$  using the GW framework. Then we find the node  $u$  from  $\hat{T}$  that has the highest coupling probability with the root node  $v$  in  $T$  and layout  $\hat{T}$  rooted at  $u$ .

**Programming Language and Included Packages.** Our framework is mainly implemented in Python. The code to compute LSST and MST from a given weight matrix is implemented in Java. The *CSS-IFS* algorithm for matrix sketching is implemented in MATLAB. For data processing and merge tree visualization, we use Python packages, including `numpy`, `matplotlib`, and `networkx`. In addition, the GW framework of Chowdhury and Needham [21] requires the Python Optimal Transport (POT) package.

## 7 Theoretical Considerations

We discuss some theoretical considerations in sketching merge trees. In the first two steps of our framework, we represent merge trees as metric measure networks and vectorize them via blow-up and alignment to a Fréchet Mean using the GW framework [21]. Each merge tree  $T = (V, W, p) \in \mathcal{T}$  is mapped to a column vector  $a$  in matrix  $A$ , where  $W$  captures the

shortest path distances using function value differences as weights. The computation of the Fréchet mean  $\bar{T}$  is an optimization process, but the blow-up of  $T$  and its alignment to  $\bar{T}$  does not change the underlying distances between the tree nodes, which are encoded in  $W$ . Therefore, reshaping the column vector  $a$  back to a pairwise distance matrix and computing its corresponding MST fully recovers the original input merge tree.

In the third step, we sketch the matrix  $A$  using either NMF or CSS. Both matrix sketching techniques (albeit with different constraints) aim to obtain an approximation  $\hat{A} = BY$  of  $A$  that minimizes the error  $\epsilon = \|A - \hat{A}\|_F$ . Let  $A_k$  denote the (unknown) best rank- $k$  approximation of  $A$ . In the case of CSS, the theoretical upper bound was given as a multiplicative error of the form  $\epsilon \leq \epsilon_k \cdot \|A - A_k\|_F$ , where  $\epsilon_k$  depends on the choice of  $k$  [25, 15], or it was given as an additive error  $\epsilon \leq \|A - A_k\|_F + \epsilon_{k,A}$ , where  $\epsilon_{k,A}$  depends on  $k$  and  $\|A\|_F$  [26, 52].  $\|A - A_k\|_F$  is often data dependent. In the case of NMF, a rigorous theoretical upper bound on  $\epsilon$  remains unknown.

Given an approximation  $\hat{A}$  of  $A$ , the next step is to reconstruct a sketched merge tree from each column vector  $\hat{a}$  of  $\hat{A}$ . We reshape  $\hat{a}$  into an  $n \times n$  matrix  $\hat{W}$  and construct a sketched tree  $\hat{T}$  by computing the MST or the LSST of  $\hat{W}$ . The distance matrix  $\hat{D}$  of the sketched tree  $\hat{T}$  thus approximates the distance matrix  $W'$  of the blow-up tree  $T'$ .

When a sketched merge tree is obtained via a LSST, there is a theoretical upper bound on the relative distortion of the distances [3], that is,  $\theta \leq O(\log n \log \log n)$  for  $\theta = \frac{1}{\binom{n}{2}} \sum_{x,x'} \left( \hat{D}(x, x') / \hat{W}(x, x') \right)$ . When a sketched merge tree is obtained via a MST, the theoretical bounds on  $\|\hat{W} - \hat{D}\|_F$  are unknown, although, in practice, MST typically provides better sketched trees in comparison with LSST, as demonstrated in Section 5. Finally, although the smoothing process does not alter the tree structure significantly, it does introduce some error in the final sketched tree, whose theoretical bound is not yet established.

Therefore, while we have obtained good experimental results in sketching merge trees, there is still a gap between theory and practice for individual sketched trees. Filling such a gap is left for future work.

## 8 Conclusion

In this paper, we present a framework to sketch merge trees. Given a set  $\mathcal{T}$  of merge trees of (possibly) different sizes, we compute a basis set of merge trees  $\mathcal{S}$  such that each tree in  $\mathcal{T}$  can be approximately reconstructed using  $\mathcal{S}$ . We demonstrate the utility of our framework in sketching merge trees that arise from scientific simulations. Our approach is flexible enough to be generalized to sketch other topological descriptors such as contour trees, Reeb graphs, and Morse–Smale graphs (e.g., [19]), which is left for future work.

---

### References

- 1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511, 2007.
- 2 Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *IEEE Symposium on Foundations of Computer Science*, pages 781–790, 2008.
- 3 Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. *ACM Symposium on Theory of Computing*, pages 395–406, 2012.
- 4 Martial Agueh and Guillaume Carlier. Barycenters in the Wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011.

- 5 Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.
- 6 David Alvarez-Melis and Tommi Jaakkola. Gromov-Wasserstein alignment of word embedding spaces. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1881–1890, 2018.
- 7 Irene Baeza Rojo and Tobias Günther. Vector field topology of time-dependent flows in a steady reference frame. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):280–290, 2020.
- 8 U. Bauer, B. Di Fabio, and C. Landi. An edit distance for Reeb graphs. *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, pages 27–34, 2016.
- 9 Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. Measuring distance between reeb graphs. *Proceedings of the 30th Annual Symposium on Computational Geometry*, page 464, 2014.
- 10 Ulrich Bauer, Claudia Landi, and Facundo Memoli. The Reeb graph edit distance is universal. *Foundations of Computational Mathematics*, 2017.
- 11 Ulrich Bauer, Elizabeth Munch, and Yusu Wang. Strong equivalence of the interleaving and functional distortion metrics for Reeb graphs. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 461–475, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- 12 Kenes Beketayev, Damir Yeliussizov, Dmitriy Morozov, Gunther Weber, and Bernd Hamann. Measuring the distance between merge trees. *Topological Methods in Data Analysis and Visualization III: Theory, Algorithms, and Applications, Mathematics and Visualization*, pages 151–166, 2014.
- 13 Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative Bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.
- 14 Christos Boutsidis and Efstratios Gallopoulos. SVD based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, 2008.
- 15 Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977, 2009.
- 16 Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Efficient computation of isometry-invariant distances between surfaces. *SIAM Journal on Scientific Computing*, 28(5):1812–1836, 2006.
- 17 Charlotte Bunne, David Alvarez-Melis, Andreas Krause, and Stefanie Jegelka. Learning generative models across incomparable spaces. *International Conference on Machine Learning*, pages 851–861, 2019.
- 18 Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.
- 19 Michael J. Catanzaro, Justin M. Curry, Brittany Terese Fasy, Janis Lazovskis, Greg Malen, Hans Riess, Bei Wang, and Matthew Zabka. Moduli spaces of Morse functions for persistence. *Journal of Applied and Computational Topology*, 4:353–385, 2020.
- 20 Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J. Guibas, and Steve Y. Oudot. Proximity of persistence modules and their diagrams. *Proceedings of the 25th Annual Symposium on Computational Geometry*, pages 237–246, 2009.
- 21 Samir Chowdhury and Tom Needham. Gromov-Wasserstein averaging in a Riemannian framework. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 842–843, 2020.
- 22 Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

- 23 Marco Cuturi and Arnaud Doucet. Fast computation of Wasserstein barycenters. *Proceedings of the 31st International Conference on Machine Learning, PMLR*, 32(2):685–693, 2014.
- 24 Vin de Silva, Elizabeth Munch, and Amit Patel. Categorized Reeb graphs. *Discrete & Computational Geometry*, pages 1–53, 2016.
- 25 Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303, 2006.
- 26 Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36:158–183, 2006.
- 27 Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, and David P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13:3441–3472, 2012.
- 28 Ian L. Dryden, Alexey Koloydenko, and Diwei Zhou. Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *Annals of Applied Statistics*, 3(3):1102–1123, 2009.
- 29 Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Morse-Smale complexes for piece-wise linear 3-manifolds. *Proceedings of the 19th Annual symposium on Computational Geometry*, pages 361–370, 2003.
- 30 Herbert Edelsbrunner, John Harer, and Afra J. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete & Computational Geometry*, 30:87–107, 2003.
- 31 Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 2005.
- 32 Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM Journal on Computing*, 38(5):1761–1781, 2009.
- 33 Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, 346–347:180–197, 2016.
- 34 Danielle Ezuz, Justin Solomon, Vladimir G Kim, and Mirela Ben-Chen. GWCNN: A metric alignment layer for deep shape analysis. *Computer Graphics Forum*, 36:49–57, 2017.
- 35 Cédric Févotte and Jérôme Idier. Algorithms for nonnegative matrix factorization with the  $\beta$ -divergence. *Neural computation*, 23(9):2421–2456, 2011.
- 36 Christoph Flamm, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Barrier trees of degenerate landscapes. *Zeitschrift für Physikalische Chemie*, 216(2), 2002.
- 37 Ellen Gasparovic, Elizabeth Munch, Steve Oudot, Katharine Turner, Bei Wang, and Yusu Wang. Intrinsic interleaving distance for merge trees. arXiv preprint arXiv:1908.00063, 2019.
- 38 Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal of Computing*, 45(5):1762–1792, 2016.
- 39 Mikhail Gromov. *Metric Structures for Riemannian and Non-Riemannian Spaces*, volume 152 of *Progress in mathematics*. Birkhäuser, Boston, USA, 1999.
- 40 Shawn Gu and Tijana Milenković. Data-driven network alignment. *PLoS ONE*, 15(7):e0234978, 2020.
- 41 Tobias Günther, Markus Gross, and Holger Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics*, 36(4):141:1–141:11, 2017.
- 42 Xiaoyang Guo and Anuj Srivastava. Representations, metrics and statistics for shape analysis of elastic graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.
- 43 Xiaoyang Guo, Anuj Srivastava, and Sudeep Sarkar. A quotient space formulation for statistical analysis of graphical data. arXiv preprint arXiv:1909.12907, 2019.



- 44 C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth. A survey of topology-based methods in visualization. *Computer Graphics Forum*, 35(3):643–667, 2016.
- 45 Reigo Hendrikson. Using Gromov-Wasserstein distance to explore sets of networks. Master’s thesis, University of Tartu, 2016.
- 46 Ibrahim Hoteit, Xiaodong Luo, Marc Bocquet, Armin Köhl, and Boujemaa Ait-El-Fquih. Data assimilation in oceanography: Current status and new directions. In Eric P. Chassignet, Ananda Pascual, Joaquin Tintoré, and Jacques Verron, editors, *New Frontiers in Operational Oceanography*. GODAE OceanView, 2018.
- 47 Brijnesh J Jain and Klaus Obermayer. Learning in Riemannian orbifolds. arXiv preprint arXiv:1204.4294, 2012.
- 48 William B Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.
- 49 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly- $m \log n$  time solver for SDD linear systems. *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science*, 2011.
- 50 Edo Liberty. Simple and deterministic matrix sketching. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–588, 2013.
- 51 Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1249–1268, 2017.
- 52 Michael W. Mahone and Petros Drineas. Structural properties underlying high-quality randomized numerical linear algebra algorithms. In M. Kane P. Buhmann, P. Drineas and M. van de Laan, editors, *Handbook of Big Data*, pages 137–154. Chapman and Hall, 2016.
- 53 Facundo Mémoli. *Estimation of distance functions and geodesics and its use for shape comparison and alignment: theoretical and computational results*. PhD thesis, University of Minnesota, 2005.
- 54 Facundo Mémoli. On the use of Gromov-Hausdorff distances for shape comparison. *Eurographics Symposium on Point-Based Graphics*, pages 81–90, 2007.
- 55 Facundo Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11(4):417–487, 2011.
- 56 Facundo Mémoli and Tom Needham. Gromov-Monge quasi-metrics and distance distributions. arXiv preprint arXiv:1810.09646, 2020.
- 57 Facundo Mémoli and Guillermo Sapiro. Comparing point clouds. *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 32–40, 2004.
- 58 Facundo Mémoli and Guillermo Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Foundations of Computational Mathematics*, 5:313–347, 2005.
- 59 Facundo Memoli, Anastasios Sidiropoulos, and Kritika Singhal. Sketching and clustering metric measure spaces. arXiv preprint arXiv:1801.00551, 2018.
- 60 J. Milnor. *Morse Theory*. Princeton University Press, New Jersey, 1963.
- 61 Dmitriy Morozov, Kenes Beketayev, and Gunther Weber. Interleaving distance between merge trees. *Proceedings of Topology-Based Methods in Visualization (TopoInVis)*, 2013.
- 62 Elizabeth Munch and Anastasios Stefanou. The  $\ell^\infty$ -cophenetic metric for phylogenetic trees as an interleaving distance. In *Research in Data Science*, Association for Women in Mathematics Series, pages 109–127. Springer International Publishing, 2019.
- 63 Bruno Ordozgoiti, Sandra Gómez Canaval, and Alberto Mozo. A fast iterative algorithm for improved unsupervised feature selection. *IEEE 16th International Conference on Data Mining*, pages 390–399, 2016.
- 64 Gabriel Peyré, Marco Cuturi, and Justin Solomon. Gromov-Wasserstein averaging of kernel and distance matrices. *Proceedings of the 33rd International Conference on Machine Learning, PMLR*, 48:2664–2672, 2016.

- 65 Jeff M. Phillips. Coresets and sketches. In *Handbook of Discrete and Computational Geometry*, chapter 48. CRC Press, 3rd edition, 2016.
- 66 S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), 2004. URL: <http://gfs.sf.net/>.
- 67 G. Reeb. Sur les points singuliers d’une forme de pfaff complètement intergrable ou d’une fonction numérique (on the singular points of a complete integral pfaff form or of a numerical function). *Comptes Rendus Acad.Science Paris*, 222:847–849, 1946.
- 68 Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. *Proceedings of 47th IEEE Symposium on Foundations of Computer Science*, pages 143–152, 2006.
- 69 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, 2004.
- 70 Raghavendra Sridharamurthy, Talha Bin Masood, Adhitya Kamakshidasan, and Vijay Nataraajan. Edit distance between merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 2018.
- 71 Karl-Theodor Sturm. The space of spaces: curvature bounds and gradient flows on the space of metric measure spaces. arXiv preprint arXiv:1208.0434, 2012.
- 72 Vayer Titouan, Nicolas Courty, Romain Tavenard, and Rémi Flamary. Optimal transport for structured data with application on graphs. *International Conference on Machine Learning*, pages 6275–6284, 2019.
- 73 Vayer Titouan, Rémi Flamary, Nicolas Courty, Romain Tavenard, and Laetitia Chapel. Sliced Gromov-Wasserstein. *Advances in Neural Information Processing Systems*, pages 14726–14736, 2019.
- 74 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.
- 75 Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable Gromov-Wasserstein learning for graph partitioning and matching. *Advances in Neural Information Processing Systems*, pages 3046–3056, 2019.
- 76 Hongteng Xu, Dixin Luo, Hongyuan Zha, and Lawrence Carin. Gromov-Wasserstein learning for graph matching and node embedding. *International Conference on Machine Learning*, pages 6932–6941, 2019.
- 77 Lin Yan, Yusu Wang, Elizabeth Munch, Ellen Gasparovic, and Bei Wang. A structural average of labeled merge trees for uncertainty visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):832–842, 2020.
- 78 Peng Zhan, George Krokos, Daquan Guo, and Ibrahim Hoteit. Three-dimensional signature of the Red Sea eddies and eddy-induced transport. *Geophysical Research Letters*, 46(4):2167–2177, 2019.
- 79 Peng Zhan, Aneesh C. Subramanian, Fengchao Yao, and Ibrahim Hoteit. Eddies in the red sea: A statistical and dynamical study. *Journal of Geophysical Research*, 119(6):3909–3925, 2014.