# Sketching Merge Trees for Scientific Data Visualization

Mingzhe Li, Sourabh Palande, Lin Yan, Bei Wang

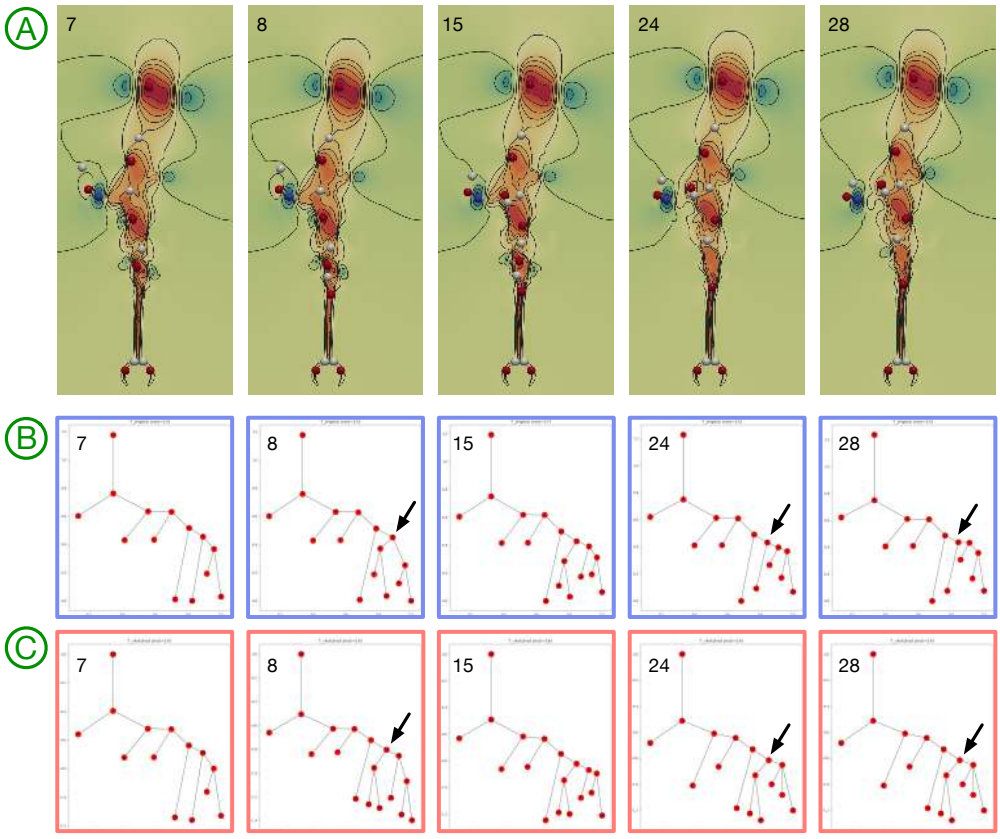


Fig. 1. We demonstrate our merge tree sketching framework with a set of 31 merge trees. Each merge tree is generated from the vertical velocity magnitude field coming from a time-varying flow simulation called the *Heated Cylinder*. By combining matrix sketching – iterative feature selection – with probabilistic matching, we show that the sketched trees (red boxes) approximate the input trees (blue boxes) reasonably well with only three basis trees. (A) A subset of scalar fields that give rise to the input merge trees labeled 7, 8, 15, 24, and 28. (B-C) Comparing each sketched tree with its corresponding input tree, highlighting noticeable structural differences among subtrees (whose roots are pointed by black arrows) before and after sketching.

**Abstract**— Merge trees are a type of topological descriptors that record the connectivity among the sublevel sets of scalar fields. They are among the most widely used topological tools in visualization. In this paper, we are interested in sketching a set of merge trees. That is, given a large set $\mathcal{T}$ of merge trees, we would like to find a much smaller 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 probabilistic matching with techniques from matrix sketching. We demonstrate the applications of our framework in sketching merge trees that arise from time-varying scientific simulations. Specifically, our framework obtains a much smaller representation of a large set of merge trees for downstream analysis and visualization. It is shown to be useful in identifying good representatives and outliers with respect to a chosen basis. Finally, our work shows a promising direction of utilizing randomized linear algebra within scientific visualization.

## 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 (e.g., see the surveys [33, 39]). Sketching, on the other hand, is a class of mathematical tools where a large dataset is replaced by a smaller one that preserves its properties of interests. In this paper, we are interested in sketching a set of topological descriptors – specifically merge trees – for scientific visualization.

We formulate our problem as follows: given a large set $\mathcal{T}$ of merge trees, we would like to find a much smaller basis set $\mathcal{S}$ such that each tree in $\mathcal{T}$ can be approximately reconstructed from a linear combination of merge trees in $\mathcal{S}$. The set $\mathcal{T}$ may arise from a time-varying field or an


- *Mingzhe Li, Lin Yan, and Bei Wang are with University of Utah. E-mails: mingzhel@cs.utah.edu, lin.yan@utah.edu, beiwang@sci.utah.edu.*
- *Sourabh Palande is with Michigan State University. E-mail: palandes@msu.edu.*

ensemble of scientific simulations, generated with varying parameters and/or different instruments. Our motivation is two-fold. We are interested in developing a merge tree sketching framework to:

- Obtain a compressed representation of a large set of merge trees – as a much smaller set of basis trees together with a coefficient matrix – for downstream analysis and visualization;
- Identify good representatives that capture topological variations in a set of merge trees as well as outliers.

A sketch of $\mathcal{T}$ with $\mathcal{S}$ gives rise to a significantly smaller representation that is a reasonable approximation of $\mathcal{T}$. In addition, elements in $\mathcal{S}$ will serve as good representatives of $\mathcal{T}$, while elements with large sketching errors will be considered as outliers.

We are inspired 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 Fig. 2 (gray box). Given a dataset of $N$ points with $d$ features, represented as a $d \times N$ matrix $A$ (with row-wise zero empirical 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 $b_1, b_2, \ldots, b_k$ 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 [10]. 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? Up until now, a large set of topological descriptors has 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 large set $\mathcal{T}$ of merge trees, can we find a much smaller basis set $\mathcal{S}$ as its "sketch"?

Our overall pipeline is illustrated in Fig. 2 and detailed in Sect. 4. In steps 1 and 2, given a set of $N$ merge trees $\mathcal{T} = \{T_1, T_2, \cdots, T_N\}$ as input, we represent each merge tree $T_i$ as a metric measure network and employ the Gromov-Wasserstein framework of Chowdhury and Needham [17] 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) or 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 probabilistic matching via Gromov-Wasserstein 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. Specifically, we show that understanding the sketchability properties of merge trees can be particularly useful for the study of time-varying fields and ensembles, where our framework can be used to *obtain compact representations* for downstream analysis and visualizaiton, and to *identify good representatives and outliers*.

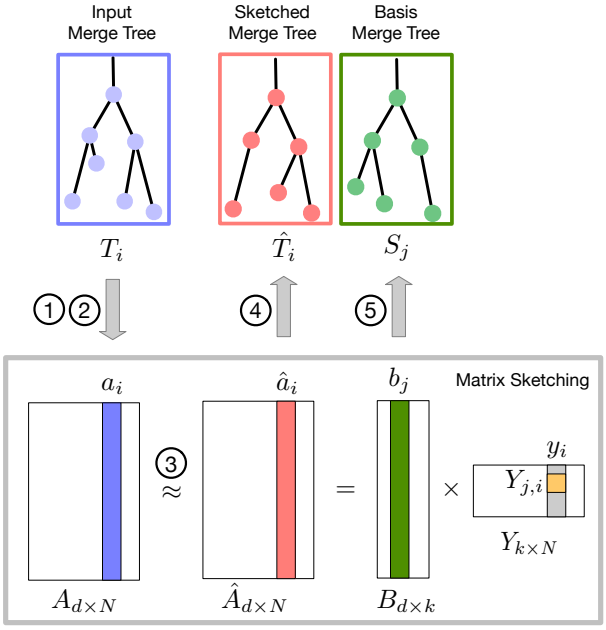


Fig. 2. The overall pipeline for sketching a set of merge trees.

## 2 Related Work

We review relevant work on merge trees, Gromov-Wasserstein distances, graph alignment, matrix sketching, and spanning trees.

**Merge trees.** Merge trees are a type of topological descriptors that record the connectivity among the sublevel sets of scalar fields (see e.g., [8,15]). They are rooted in Morse theory [48], which characterizes scalar field data by the topological changes in its sublevel sets at isolated critical points. In this paper, instead of a direct comparison between a pair of merge trees using existing metrics for merge trees or Reeb graphs (e.g., [8,28,54]), we treat merge trees as metric measure networks and utilize the Gromov-Wasserstein framework described in Sect. 3 to obtain their alignment and vector representations.

**Gromov-Wasserstein (GW) distances.** Gromov introduced Gromov-Hausdorff (GH) distances [30] 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.*, [13,41, 42,45,46]), where shapes are treated as metric spaces, and two shapes are considered equal if they are isometric. Memoli [43] 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 [56,57] and measure-preserving mappings [44]. Apart from theoretical explorations [43,55], GW distances have been utilized in the study of graphs and networks [34,59,60], machine learning [14, 26], and word embeddings [6]. Recently, Memoli *et al.* [47] 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 instead focuses on approximating a large set of merge trees – modeled as a set of metric measure networks – with a much 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.*, [25,31]). It is often needed to establish node correspondences between graphs of different sizes. The approaches that are most relevant here are the ones based on the GW distances [17,50], which employ *probabilistic matching* ("soft matching") of nodes. Infor-

mation in a graph can be captured by a symmetric positive semidefinite matrix that encodes distances or similarities between pairs of nodes. Dryden *et al.* [23] 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.* [19] and Benamou *et al.* [9] developed efficient algorithms to compute such barycenters. Peyre *et al.* [50] combined these ideas with the notion of GW distances [43] to develop GW averaging of distance/similarity matrices. Chowdhury and Needham [17] built upon the work in [50] and provided a GW framework to compute a Frechét mean among these matrices using measure couplings. In this paper, we utilize the GW framework [17] for probabilistic 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 SVD, column or row sampling [21,22], random projection [53], and frequent directions [29,38]; see [51,58] for surveys.

The column sampling approach carefully chooses a subset of the columns of $A$ proportional to their *importance*, where the importance is determined by the squared norm (*e.g.*, [21]) or the (approximated) leverage scores (*e.g.*, [22]). The random projection approach takes advantage of the Johnson-Lindenstrauss (JL) Lemma [36] to create an $N \times k$ linear projection matrix $S$ (*e.g.*, [53]), where $B = AS$. The frequent directions approach [29,38] 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) [1–3]. 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 [24,37].

## 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. Our framework utilizes the probabilistic matching from the Gromov-Wasserstein (GW) framework of Chowdhury and Needham [17], with a few ingredients from Peyre *et al.* [50].

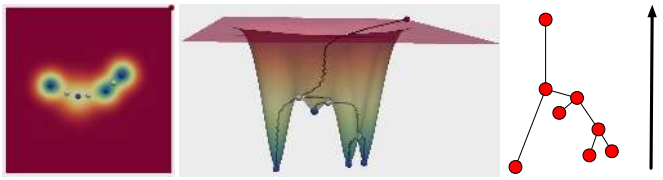

Fig. 3. An examples of a merge tree from a height field. From left to right: 2D scalar fields visualization, merge trees embedded in the graphs of the scalar fields, and abstract visualization of merge trees as rooted trees equipped with height functions.

**Merge trees.** Let $f : \mathbb{M} \to \mathbb{R}$ 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 Sect. 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. Fig. 3 displays a scalar field with its corresponding merge tree embedded in the graph of the scalar field. Abstractly, a merge tree $T$ is a rooted tree equipped with a scalar function defined on its node set, $f : V \to \mathbb{R}$.

**Gromov-Wasserstein distance for measure networks.** The GW distance was proposed by Memoli [42, 43] for metric measure spaces. Peyre *et al.* [50] 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 $\mathbf{1}_n = (1, 1, \ldots, 1)^T \in \mathbb{R}^n$.

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 $\{1, 2, \ldots, n\}$. $V_1 = \{x_i\}_{i \in [n_1]}$ 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 [50]

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

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* [50] is defined as

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

In this paper, we consider the quadratic loss function $L(a, b) = \frac{1}{2}|a - b|^2$. The *Gromov-Wasserstein distance* [17, 43, 50] $d_{GW}$ between $G_1$ and $G_2$ is defined as

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

It follows from the work of Sturm [55] that such minimizers always exist and are referred to as *optimal couplings*.

**Alignment and blowup.** Given a pair of graphs $G_1 = (V_1, W_1, p_1)$ and $G_2 = (V_2, W_2, p_2)$ with $n_1$ and $n_2$ nodes respectively, a coupling $C \in \mathcal{C}(p_1, p_2)$ can be used to *align* their nodes. In order to do this, we will need to increase the size of $G_1$ and $G_2$ appropriately into their respective *blowup* graphs $G_1'$ and $G_2'$, such that $G_1'$ and $G_2'$ contain the same $n$ number of nodes (where $n_1, n_2 \leq n$). Roughly speaking, let $x$ be a node in $G_1$, and let $n_x$ be the number of nodes in $G_2$ that have a nonzero coupling probability with $x$. The blowup graph $G_1' = (V_1', W_1', p_1')$ is created by making $n_x$ copies of node $x$ for each node in $G_1$, generating a new node set $V_1'$. The probability distribution $p_1'$ and the weight matrix $W_1'$ are updated from $p_1$ and $W_1$ accordingly. 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 be an $n \times n$ permutation matrix, 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 $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$, a *Fréchet mean* [17] $\overline{G}$ 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,

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

Chowdhury and Needham [17] 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 $k^{th}$ 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 [17]. If $\overline{G} = (\overline{V},\overline{W},\overline{p})$ is the Fréchet mean, then we have

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

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

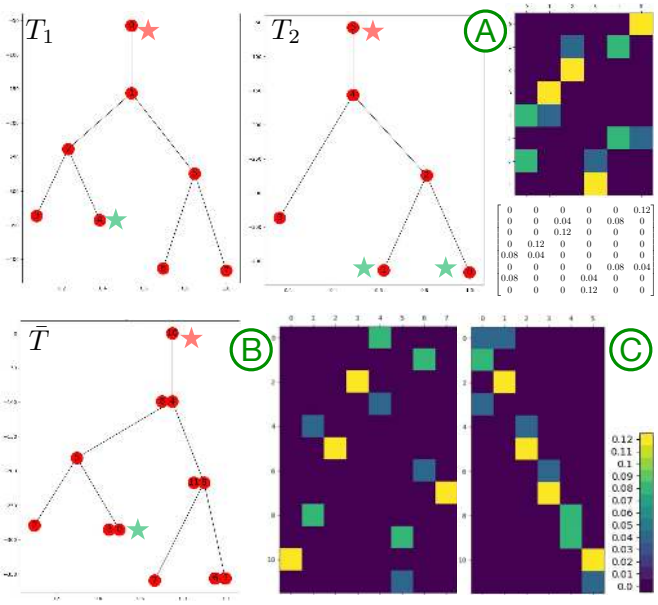


Fig. 4. An optimal coupling between two simple merge trees $T_1$ and $T_2$. The coupling matrix is visualized in (A): yellows means high and dark blue means low probability. Couplings between the Fréchet mean $\overline{T}$ with $T_1$ and $T_2$ are shown in (B) and (C), respectively.

**A simple example.** We give a simple example involving a pair of merge trees in Fig. 4. $T_1$ and $T_2$ contain 8 and 6 nodes, respectively (nodes are labeled starting with a 0 index). The optimal coupling $C$ obtained by the gradient descent algorithm is visualized in Fig. 4(A). $C$ is an $8\times 6$ matrix, and it shows that node 0 in $T_1$ is matched to node 5 in $T_2$ with the highest probability (0.12, red stars). Node 4 in $T_1$ is coupled with both node 0 (with a probability 0.08) and node 1 (with a probability 0.04) in $T_2$ (green stars).

Now, we compute the Fréchet mean $\overline{T}$ of $T_1$ and $T_2$, which has 12 nodes. We align both $T_1$ and $T_2$ to $\overline{T}$ via their blowup graphs. This gives rise to a coupling matrix between $\overline{T}$ and $T_1$ (of size $12\times 8$) in Fig. 4(B), and a coupling matrix between $\overline{T}$ and $T_2$ (of size $12\times 6$) in Fig. 4(C), respectively. As shown in Fig. 4, root node 10 of $\overline{T}$ is matched with root node 0 of $T_1$ and root node 5 of $T_2$ (red stars). Node 0 of $\overline{T}$ is matched probabilistically with node 4 in $T_1$ and nodes 0 and 1 in $T_2$ (green stars). 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 $\mathcal{S}$ with $k \ll N$ merge trees 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 [17] with techniques from matrix sketching to achieve this goal. We detail our pipeline to compute $\mathcal{S}$, as illustrated in Fig. 2.

**Step 1: Representing merge trees as measure networks.** The first step is to represent merge trees as metric measure networks as described in Sect. 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.*, $p = \frac{1}{|V|}\mathbf{1}_{|V|}$. Recall that each node $x$ in a merge tree is associated with a scalar value $f(x)$. For a pair of nodes $x,x' \in V$, if they are adjacent, we define $W(x,x') = |f(x) - f(x')|$, *i.e.*, their absolute difference in function value; otherwise, $W(x,x')$ is the shortest path distance between them in $T$. By construction, a shortest path between two nodes goes through their lowest common ancestor in $T$. We define $W$ in such a way to encode information in $f$, which is inherent to a merge tree.

**Step 2: Merge tree vectorization via 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 $\overline{T} = (\overline{V},\overline{W},\overline{p})$. Let $n = |\overline{V}|$. 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 experiment, 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 $\overline{T}$ and $T_i$ is an $n\times n_i$ matrix with at least $n$ nonzero entries. If the number of nonzero entries in each row is greater than $n$, we keep only the largest value. That is, if a node of $\overline{T}$ 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 $\overline{T}$ with $T'$. The above procedure ensures that each blown-up tree $T'$ has exactly $n$ nodes, and the binarized coupling matrix $C'$ between $\overline{T}$ 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 Fig. 2 (step 2)[1]. Each $a$ is a vector representation of the input tree $T$ with respect to the Fréchet mean $\overline{T}$.

**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 Fig. 2 (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 Appendix C 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.

[1] In practice, $d = \overline{(n+1)n/2}$ as we store only the upper triangular matrix.

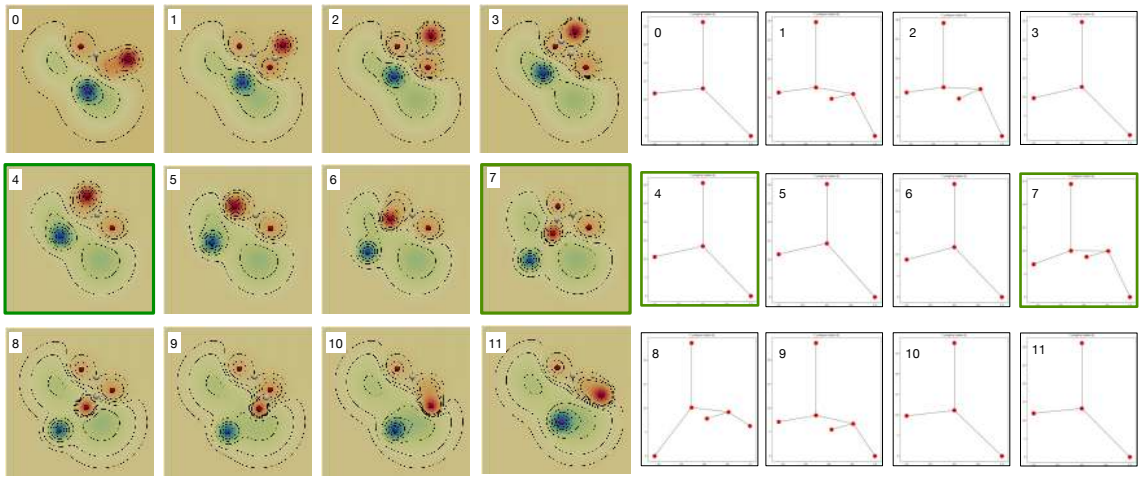


Fig. 5. Visualizing a time-varying mixture of Gaussian functions (left) together with (right) their corresponding merge trees.

In the first variant of CSS, referred to as *Length Squared Sampling* (*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, making the chosen basis as orthogonal as possible.

In the second variant of CSS, referred to as the *Iterative Feature Selection* (*IFS*), we use the algorithm proposed by Ordozgoiti *et al.* [49]. Instead of selecting columns sequentially as in *LSS*, *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.

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 [18, 27]. 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 $i^{th}$ 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 Appendix C 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 Fig. 2 (step 5).

**Error analysis.** For each of our experiments, we compute the *global sketch error* $\varepsilon = \|A-\hat{A}\|_F^2$, as well as *column-wise sketch error* $\varepsilon_i = \|a_i - \hat{a}_i\|_2^2$, where $\varepsilon = \sum_{i=1}^N \varepsilon_i$. By construction, $\varepsilon_i \leq \varepsilon$. 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 *column-wise GW loss*. The *global GW loss* is defined to be $\tau = \sum_{i=1}^N \tau_i$. For theoretical considerations, see discussions in Appendix B.

**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 $\mathcal{T} = \{T_0, \ldots, T_{11}\}$ of merge trees from 12 consecutive time steps, referred to as the *Rotating Gaussian* dataset. In Fig. 5, 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 $f$.

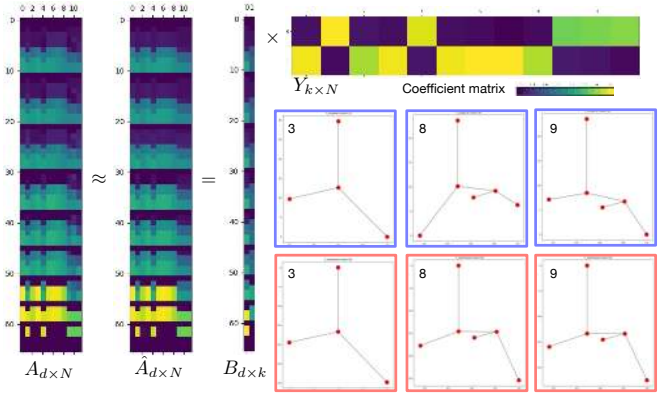


Fig. 6. *Rotating Gaussian* dataset: Examples of input merge trees (blue boxes) with their sketched versions (red boxes). Visualizing data matrices associated with the sketching, while highlighting the coefficient matrix.

Since the dataset is quite simple, a couple of basis trees are sufficient to obtain very good sketching results. Using $k = 2$, *IFS* select $\mathcal{S} = \{T_4, T_7\}$. In Fig. 5, we highlight the two basis trees selected with *IFS* and their corresponding scalar fields, respectively, with green boxes. The topological structures of $T_1$ and $T_6$ are noticeably distinct among the input trees. They clearly capture the structural variations and serve as good representatives of the set $\mathcal{T}$.

We also show a few input trees $T_3, T_8, T_9$ (blue boxes) and their sketched versions (red boxes) in Fig. 6. The input and the sketched tree for $T_3$ are almost indistinguishable. However, there are some structural differences between the input and sketched trees for $T_8$ and $T_9$ due to randomized approximations. We also visualize the data matrix $A$, $\hat{A}$, $B$, and highlight the coefficient matrix $Y$ in Fig. 6. The Frechét mean tree $\overline{T}$ contains 11 nodes. The coefficient matrix, shows that each input tree (column) is well represented (with high coefficient) by one of the two basis trees. In particular, columns in the coefficience matrix with high (yellow or light green) coefficients (*w.r.t.* the given basis) may be grouped together, forming two clusters $\{T_0, T_2, T_3, T_5, T_6, T_7.T_8\}$ and

$\{T_1, T_4, T_9, T_{10}, T_{11}\}$ whose elements look structurally similar.

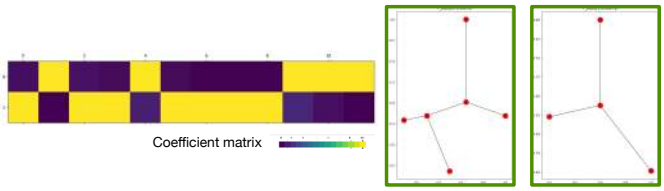


Fig. 7. *Rotating Gaussian* data set: coefficient matrices together with basis trees returned by NMF.

On the other hand, using NMF, when $k = 2$, we display the coefficient matrix together with basis trees (obtained via MST) in Fig. 7. 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 *IFS*. In addition, columns in the coefficient matrix with high coefficients (*w.r.t.* the same basis) may be grouped together that show the same two clusters as before.

## 5 Experimental Results

We demonstrate the applications of our sketching framework with merge trees that arise from three time-varying datasets from scientific simulations. The key takeaway is that, using matrix sketching and probabilistic matching between the merge trees, a large set $\mathcal{T}$ of merge trees is replaced by a much smaller basis set $\mathcal{S}$ such that trees in $\mathcal{T}$ are well approximated by trees in $\mathcal{S}$. Such a compressed representation can then be used for downstream analysis and visualization. In addition, our framework makes each large dataset simple to understand, where elements in $\mathcal{S}$ serve as good representatives that capture structural variations among the time instances, and elements with large sketching errors are considered as outliers (*w.r.t.* a chosen basis).

**Parameters.** To choose the appropriate $k$ number of basis trees for each dataset, we use the "elbow method" to determine $k$, similar to cluster analysis. We plot the global GW loss and global sketch error as a function of $k$, and pick the elbow of the curve as the $k$ to use. As shown in Fig. 8, $k$ is chosen to be 3, 15 and 30 for the *Heated Cylinder*, *Corner Flow*, and *Red Sea*[2] datasets respectively. In subsequent sections, element-wise GW losses and sketch errors also reaffirm these choices.

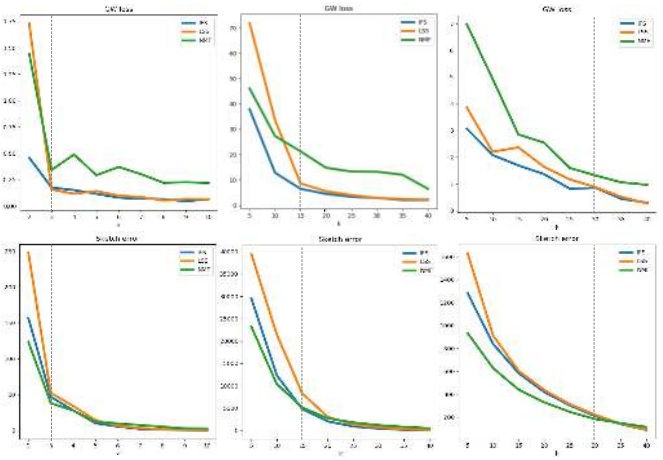

Fig. 8. Global GW losses and global sketch errors for varying $k$, the number of basis trees. From left to right, *Heated Cylinder*, *Corner Flow*, and *Red Sea* datasets.

For our experiments shown in Fig. 8, *IFS* and *LSS* give sketched trees with lower GW losses than NMF, in particular, for datasets with smaller merge trees or smaller amount of topological changes, such as *Heated Cylinder* and *Corner Flow* datasets. While NMF performs better for *Red Sea* dataset with lower sketch errors when individual input trees do not capture the complex topological changes across time instances. Furthermore, *IFS* (blue curve) performs slightly better than *LSS* (orange curve), based on error analysis (see Fig. 8 and Appendix D for details). In term of merge tree reconstruction from distance matrices, MST generally gives more visually appealing sketched trees and basis trees in practice than LSST, thus we discuss MSTs throughout this section and include some results on LSST in Appendix A.

### 5.1 Heated Cylinder Dataset

Two of our datasets come from numerical simulations available online[3]. 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 [32,52]. The dataset shows a time-varying turbulent plume containing numerous small vortices. We convert each time instance of the flow (a vector field) into a scalar field using the magnitude of its vertical (y) velocity component. We generate a set of merge trees from these scalar fields based on 31 time steps – they correspond to steps 600-630 from the original 2000 time steps. This set captures the evolution of small vortices over time.

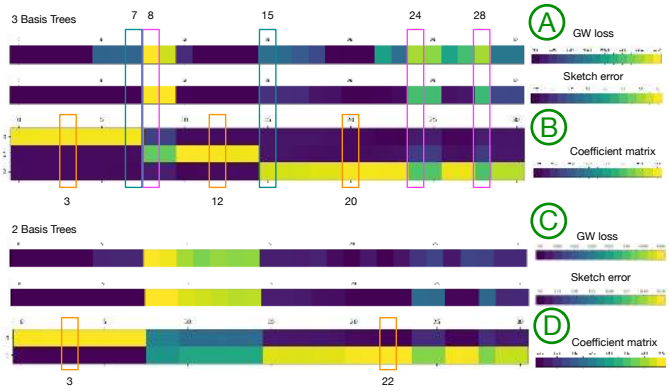


Fig. 9. Sketching the *Heated Cylinder* dataset with three (A-B) and two (C-D) basis trees. (A, C) column-wise sketch error and GW loss, (B, D) coefficient matrix. Orange boxes highlight basis trees. Magenta and teal boxes highlight trees with large and small sketch error/GW loss, respectively. Configuration: *IFS* with MST.

Given 31 merge trees $\mathcal{T} = \{T_0, \ldots, T_{30}\}$ from the *Heated Cylinder* dataset, we apply both CSS (specifically, *IFS* and *LSS*) and NMF to obtain a set of basis trees $\mathcal{S}$ and reconstruct the sketched trees. We first demonstrate that with only three basis trees, we could obtain visually appealing sketched trees with small errors. We then describe how the basis trees capture structural variations among the time-varying input.

**Sketched trees with *IFS*.** We first illustrate our sketching results using *IFS*. Based on our error analysis using the "elbow method", three basis trees appear to be the appropriate choice that strikes a balance between data summarization and structural preservation. The coefficient matrix, column-wise sketch error and GW loss (Fig. 9) are used to guide our investigation into the quality of individual sketched trees. Trees with small GW losses or sketch errors are considered well sketched, whereas those with large errors are considered outliers. We give examples of a couple of well-sketched tree – $T_7$ and $T_{15}$ (teal boxes) – with several outliers – $T_8$, $T_{24}$, and $T_{28}$ (magenta boxes) – *w.r.t.* the chosen basis.

As illustrated in Fig. 1, we compare a subset of input trees (B, blue boxes) against their sketched versions (C, red boxes). Even though we only use three basis trees, a large number of input trees – such as $T_3$, $T_{15}$ – and their sketched versions are indistinguishable with small errors. Even though $T_8$, $T_{24}$, and $T_{28}$ are considered outliers relative to other input trees, their sketched versions do not deviate significantly from the original trees. We highlight subtrees with noticeable structural

[2] Admittedly, *Red Sea* dataset is the hardest to sketch, even 30 (basis trees) may not be the optimal.

[3] https://cgl.ethz.ch/research/visualization/data.php

differences before and after sketching in Fig. 1(C), whose roots are pointed by black arrows.

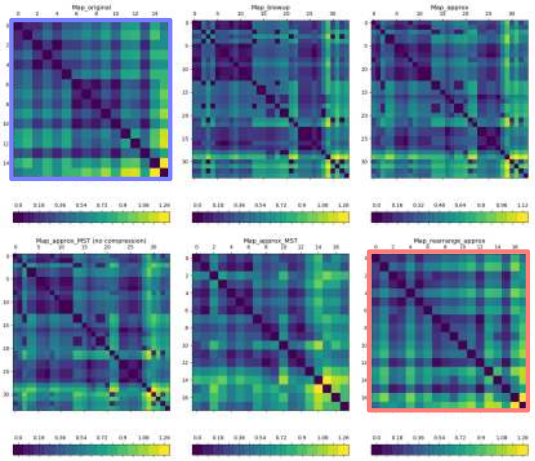

Fig. 10. *Heated Cylinder*: weight matrices associated with $T_{24}$ during the sketching process. Configuration: *IFS* with MST.

In Fig. 10, we further investigate the weight matrices from different stages of the sketching pipeline for tree $T = T_{24}$. From left to right, we show the weight matrix $W$ of the input tree, its blow-up matrix $W'$ (which is linearized to a column vector $a$), the approximated column vector $\hat{a}$ after sketching (reshaped into a square matrix), the weight matrix $\hat{W}'$ of the MST derived from the reshaped $\hat{a}$, the weight matrix of the MST after simplification, and root alignment $\hat{W}$ *w.r.t.* $T$. We observe minor changes between $W$ (blue box) and $\hat{W}$ (red box), which explain the structural differences before and after sketching in Fig. 1.

**Basis trees as representatives.** As shown in Fig. 11(A), *IFS* produces three basis trees, $\mathcal{S} = \{T_3, T_{12}, T_{20}\}$, which capture noticeable structural variations among the input merge trees. Specifically, moving from $T_3$ to $T_{12}$, and $T_{12}$ to $T_{20}$, a saddle-minima pair appears in the merge trees respectively (highlighted by orange circles). These changes in the basis trees reflect the appearances of critical points in the domain of the time-varying fields, see Fig. 11(B). In Fig. 11(C), we highlight (with orange balls) the appearances of these critical points in the domain.

Furthermore, the coefficient matrix in Fig. 9(B) contains a number of yellow or light green blocks, indicating that consecutive input trees share similar coefficients *w.r.t.* the chosen basis and thus grouped together into clusters. Again, such a blocked structure indicates that the chosen basis trees appear to be good representatives of the clusters. In comparison, using just two basis trees does not capture the structural variations as well, where we see a slight degradation in the blocked structure thus sketching quality in Fig. 9(C-D).

**Sketching with *LSS* and *NMF*.** Additionally, we include the sketching results using *LSS* and NMF as alternative strategies, again with three basis trees. *LSS* gives basis trees $T_2, T_{10}$ and $T_{27}$ in Fig. 12 (Top), which are similar to the ones obtained by *IFS* (Fig. 11). Using NMF, we show the three basis trees together with a coefficient matrix in Fig. 12 (Bottom). Although these basis trees are generated by non-negative matrix factorization, that is, they do not correspond to any input trees; nevertheless they nicely pick up the structural variations in data and are shown to resemble the basis trees chosen by column selections. This shows that even through these matrix sketching techniques employ different (randomized) algorithms, they all give rise to reasonable choices of basis trees, which lead to good sketching results.

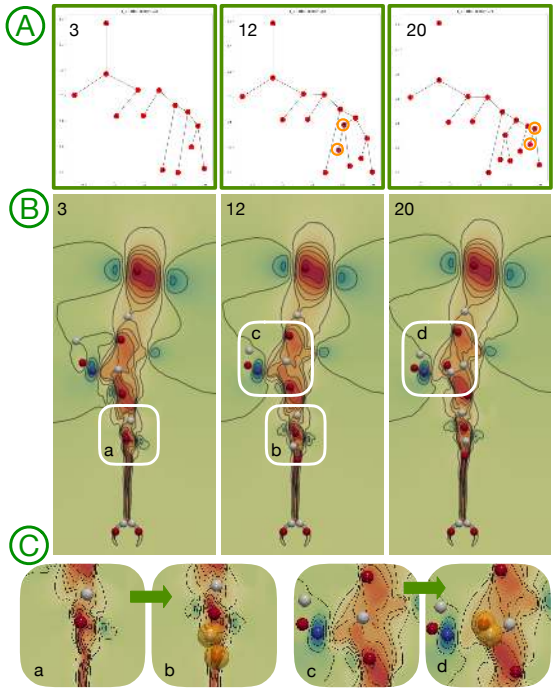


Fig. 11. Sketching the *Heated Cylinder* dataset with 3 basis trees: (A) basis trees where orange circles highlight topological changes *w.r.t.* nearby basis trees, (B) scalar fields that give rise to these basis trees, areas with critical points appearances/disappearances are shown with zoomed views in (C). Configuration: *IFS* with MST.

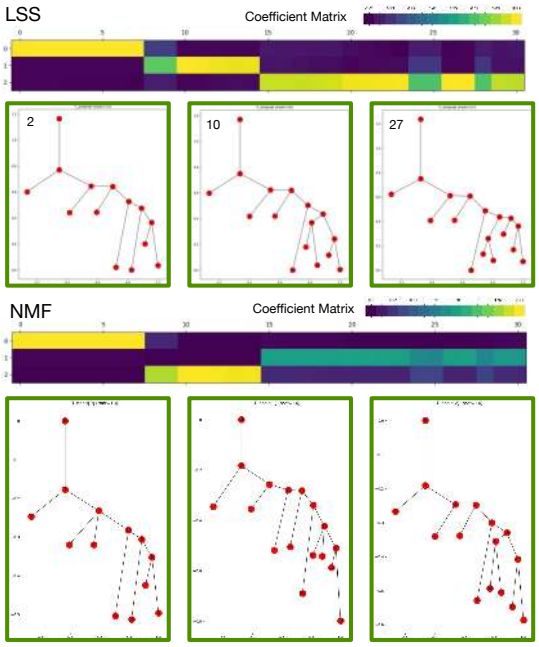


Fig. 12. Coefficient matrices and basis trees used to sketch the *Heated Cylinder* dataset with 3 basis trees with *LSS* (top) and NMF (bottom).

## 5.2 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, 52]. 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 magnitude of the velocity fields of 100 time instances, which correspond to steps 801-900 from the original 1500 time steps. This dataset describes the formation of a one-sided vortex street on the upper right corner.

Given a set of 100 merge trees from the *Corner Flow* dataset, we first demonstrate that a set of 15 basis trees chosen with *IFS* gives visually appealing sketched trees with small error, based on the coefficient

matrices and error analysis.

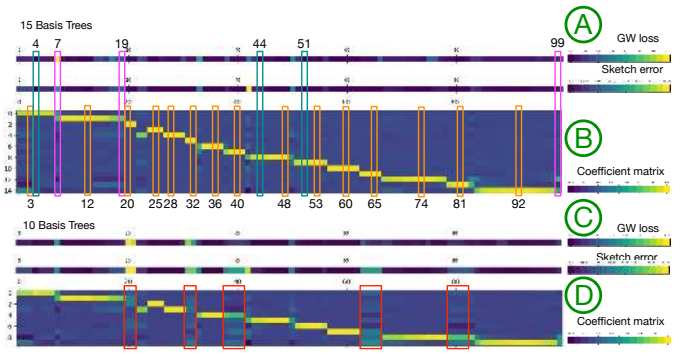


Fig. 13. Sketching the *Corner Flow* dataset with 15 (A, B) and 10 (C, D) basis trees. (A, C) column-wise sketch error and GW loss, (B, D) coefficient matrix. Orange boxes highlight basis trees. Magenta and teal boxes highlight trees with large and small sketch error/GW loss, respectively. Red boxes in (D) indicate trees that are better sketched with 15 basis trees. Configuration: *IFS* with MST.

**Coefficient matrices.** Using the "elbow method" in the error analysis, we set $k = 15$. We first compare the coefficient matrices generated using *IFS*, for $k = 10, 15$, respectively. Comparing Fig. 13(A) and (C), we see in general improved column-wise GW loss and sketch error using 15 instead of 10 basis trees. Furthermore, the coefficient matrix with 15 basis trees (B) contains better block structure than the one with 10 basis trees (D). Particularly, using additional basis trees improves upon the sketching results in regions enclosed by red boxes in (D).

**Basis trees as representatives.** We thus report the sketching results with 15 basis trees under *IFS*. The basis trees are selected with labels 3, 12, 21, 25, 28, 32, 36, 40, 48, 53, 60, 65, 74, 81, 92. Similar to the *Heated Cylinder*, we observe noticeable structural changes among pairs of adjacent basis trees, which lead to a partition of the input trees into clusters with similar structures; see the block structure in Fig. 13(B). Thus the basis trees serve as good cluster representatives, as they are roughly selected one per block. We highlight the structural changes (with black arrows pointing at the roots of subtrees) among a subset of adjacent basis trees in Fig. 15 (green boxes).

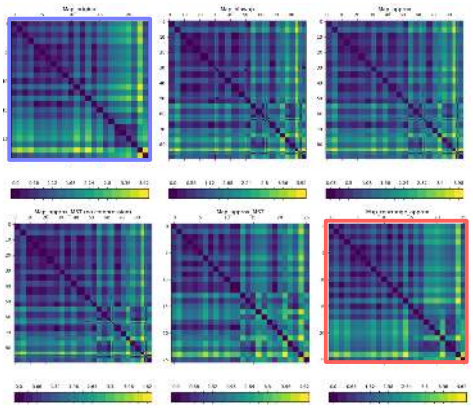

Fig. 14. Sketching the *Corner Flow* dataset with 15 basis trees. Weight matrices associated with $T_{99}$ during the sketching process. Configuration: *IFS* with MST.

**Sketched trees.** Finally, we investigate individual sketched trees in Fig. 15. We utilize the column-wise errors to select well-sketched trees (trees 4, 44, and 51) and outliers (trees 7, 19, and 99). Trees with lower GW loss and sketch error are structurally similar to the chosen basis trees, and thus have a good approximation of their topology. For instance, the sketched tree 4 (red box) is almost indistinguishable *w.r.t.* to the original (blue box); the only difference is that the node pointed by the black arrow has a slightly higher function value.

On the other hand, we observe that each outlier tree (e.g., $T_7, T_{99}$) is less visually appealing and has a higher sketch error. For instance, $T_{99}$ is shown to be a linear combination of two basis trees ($T_{74}$ and $T_{92}$), see also Fig. 13(B). Its weight matrices before, during, and after sketching are shown in Fig. 14, their differences before (blue box) and after (red box) sketching explain the observed structural discrepancies.

### 5.3 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[4]. The dataset is used to study the circulation dynamics and eddy activities of the Red Sea (see [35, 61, 62]). For our analysis, we use merge trees that arise from velocity magnitude fields of an ensemble (named *001.tgz*) with 60 times steps. Latter time steps capture the formation of various eddies, which are circular movements of water important for transporting energy and biogeochemical particles in the ocean.

The *Red Sea* dataset comes with 60 merge trees. The input does not exhibit natural clustering structures because many adjacent time instances give rise to trees with a large number of structural changes. In this case, NMF performs better than *IFS* and *LSS* in providing visually appealing sketched trees since individual input trees do not capture these complex topological changes.

**Coefficient matrices.** Using both NMF and *LSS*, we compare the coefficient matrices for $k = 15$ and 30, respectively; see Fig. 16 (Top). For *LSS*, the input trees appear to have very diverse structures without clear large clusters. This phenomenon is evident by the lack of block structure (e.g., long yellow rows) in the coefficient matrices. It is also interesting to notice that for LSS, there exists a subset of consecutive columns that contain few selected basis (e.g., red boxes for $k = 15, 30$). On the other hand, using NMF, we obtain a slightly better block structure in the coefficient matrices. In general, the global sketch error and GW loss improve as we increase the number of basis.

**Sketched trees.** In general, the *Red Sea* dataset exhibits complex topological changes across time, thus it is not an easy dataset to sketch. We investigate the sketched individual trees with NMF and $k = 30$ in Fig. 16 (Bottom). We visualize a number of sketched trees (trees labeled 1, 2, 3, 6, 36, 52) with varying errors together with their corresponding scalar fields. Given the diversity of the input trees, with 30 basis, we obtain a number of visually appealing sketched trees with minor structural differences (pointed by black arrows) *w.r.t.* to the original input trees. This show a great potential in using matrix factorization approaches to study and compress large collections of scientific datasets while preserving their underlying topology.

## 6 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 framework can be used to obtain compact representations for downstream analysis and visualization, and to identify good representatives and outliers. 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., [16]), which is left for future work.


## Acknowledgments

This work was partially funded by DOE DE-SC0021015. 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.


[4] https://kaust-vislab.github.io/SciVis2020/

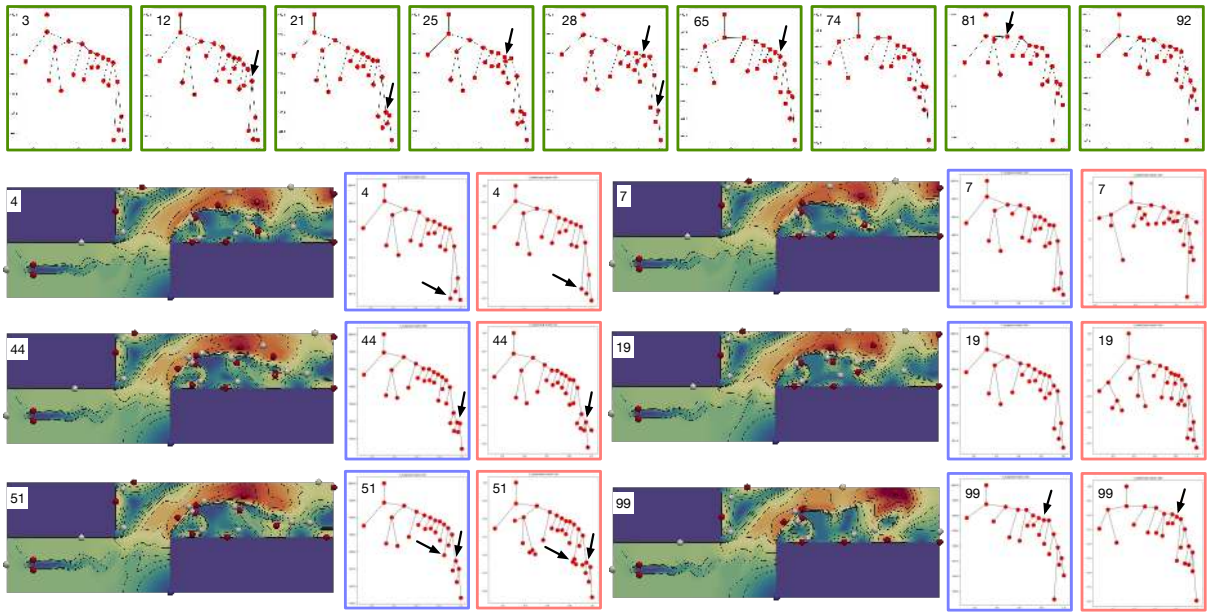


Fig. 15. Individual sketched trees for the *Corner Flow* dataset with 15 basis trees. Configuration: *IFS* and MST. Green boxes are basis trees. Blue boxes are input trees while red boxes are sketched trees.

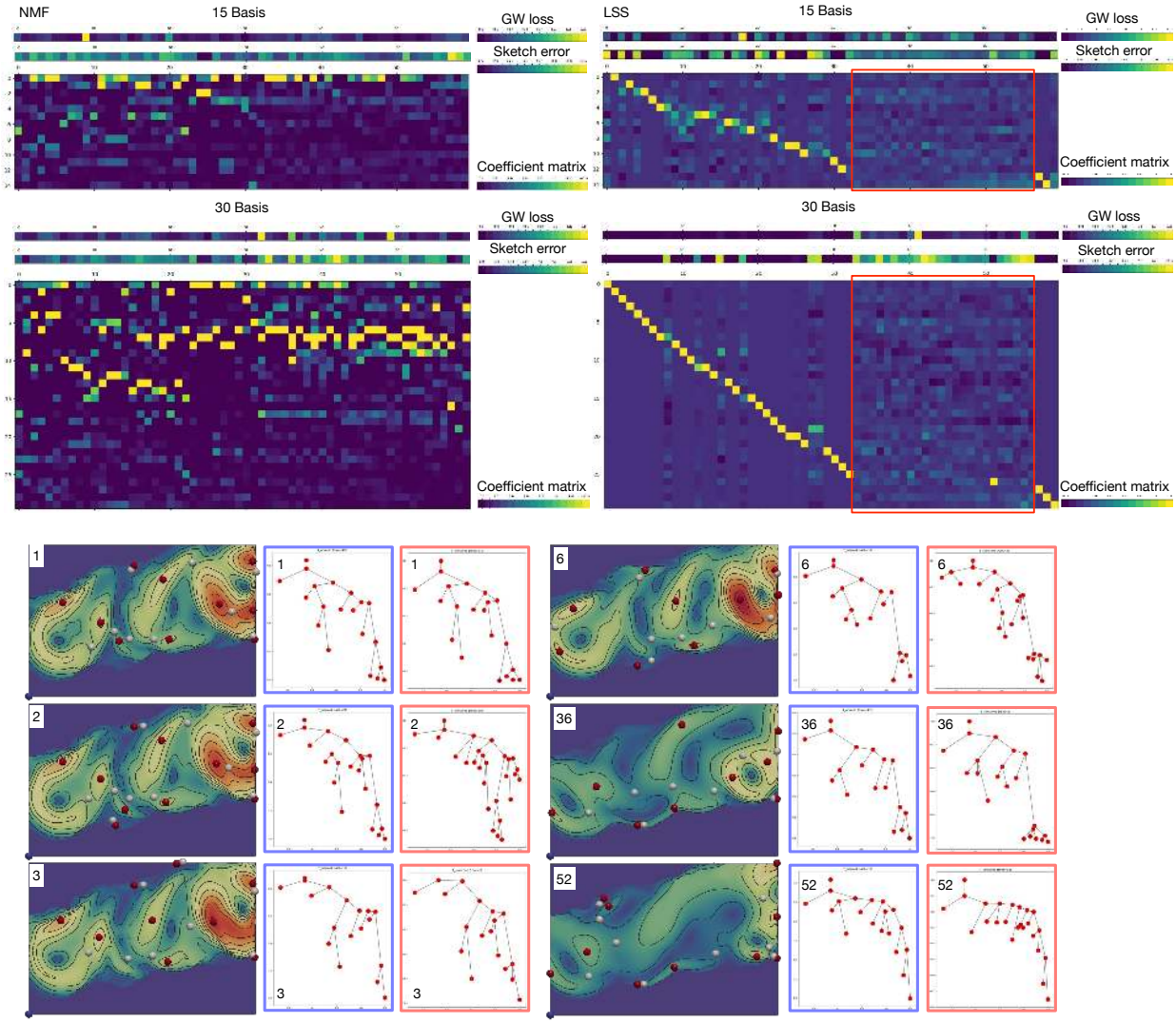


Fig. 16. Top: Coefficient matrices, column-wise sketch error and GW loss, for sketching the *Red Sea* dataset with 15 and 30 basis trees using NMF (left) and *LSS* (right), respectively. Bottom: Individual sketched trees for the *Red Sea* dataset together with their corresponding scalar fields. Configuration: 30 basis trees, NMF and MST. Blue boxes are input trees and red boxes are sketched trees.

## REFERENCES


[1] I. Abraham, Y. Bartal, and O. 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] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. *IEEE Symposium on Foundations of Computer Science*, pages 781–790, 2008.

[3] I. Abraham and O. Neiman. Using petal-decompositions to build a low stretch spanning tree. *ACM Symposium on Theory of Computing*, pages 395–406, 2012.

[4] M. Agueh and G. Carlier. Barycenters in the Wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011.

[5] N. Alon, R. M. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

[6] D. Alvarez-Melis and T. Jaakkola. Gromov-Wasserstein alignment of word embedding spaces. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1881–1890, 2018.

[7] I. Baeza Rojo and T. 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] K. Beketayev, D. Yeliussizov, D. Morozov, G. Weber, and B. 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.

[9] J.-D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré. Iterative Bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

[10] A. Bhaskara, S. Lattanzi, S. Vassilvitskii, and M. Zadimoghaddam. Residual based sampling for online low rank approximation. *IEEE 60th Annual Symposium on Foundations of Computer Science*, 2019.

[11] C. Boutsidis and E. Gallopoulos. SVD based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, 2008.

[12] C. Boutsidis, M. W. Mahoney, and P. 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.

[13] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Efficient computation of isometry-invariant distances between surfaces. *SIAM Journal on Scientific Computing*, 28(5):1812–1836, 2006.

[14] C. Bunne, D. Alvarez-Melis, A. Krause, and S. Jegelka. Learning generative models across incomparable spaces. *International Conference on Machine Learning*, pages 851–861, 2019.

[15] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.

[16] M. J. Catanzaro, J. M. Curry, B. T. Fasy, J. Lazovskis, G. Malen, H. Riess, B. Wang, and M. Zabka. Moduli spaces of Morse functions for persistence. *Journal of Applied and Computational Topology*, 4:353–385, 2020.

[17] S. Chowdhury and T. 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.

[18] A. Cichocki and A.-H. 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.

[19] M. Cuturi and A. Doucet. Fast computation of Wasserstein barycenters. *Proceedings of the 31st International Conference on Machine Learning, PMLR*, 32(2):685–693, 2014.

[20] A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303, 2006.

[21] P. Drineas, R. Kannan, and M. 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.

[22] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13:3441–3472, 2012.

[23] I. L. Dryden, A. Koloydenko, and D. Zhou. Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *Annals of Applied Statistics*, 3(3):1102–1123, 2009.

[24] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 2005.

[25] F. Emmert-Streib, M. Dehmer, and Y. Shi. Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, 346–347:180–197, 2016.

[26] D. Ezuz, J. Solomon, V. G. Kim, and M. Ben-Chen. GWCNN: A metric alignment layer for deep shape analysis. *Computer Graphics Forum*, 36:49–57, 2017.

[27] C. Févotte and J. Idier. Algorithms for nonnegative matrix factorization with the $\beta$-divergence. *Neural computation*, 23(9):2421–2456, 2011.

[28] E. Gasparovic, E. Munch, S. Oudot, K. Turner, B. Wang, and Y. Wang. Intrinsic interleaving distance for merge trees. arXiv preprint arXiv:1908.00063, 2019.

[29] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal of Computing*, 45(5):1762–1792, 2016.

[30] M. Gromov. *Metric Structures for Riemannian and Non-Riemannian Spaces*, volume 152 of *Progress in mathematics*. Birkhäuser, Boston, USA, 1999.

[31] S. Gu and T. Milenković. Data-driven network alignment. *PLoS ONE*, 15(7):e0234978, 2020.

[32] T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics*, 36(4):141:1–141:11, 2017.

[33] 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.

[34] R. Hendrikson. Using Gromov-Wasserstein distance to explore sets of networks. Master's thesis, University of Tartu, 2016.

[35] I. Hoteit, X. Luo, M. Bocquet, A. Köhl, and B. Ait-El-Fquih. Data assimilation in oceanography: Current status and new directions. In E. P. Chassignet, A. Pascual, J. Tintoré, and J. Verron, editors, *New Frontiers in Operational Oceanography*. GODAE OceanView, 2018.

[36] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[37] I. Koutis, G. L. Miller, and R. Peng. A nearly-$m \log n$ time solver for SDD linear systems. *Procedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science*, 2011.

[38] E. Liberty. Simple and deterministic matrix sketching. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–588, 2013.

[39] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1249–1268, 2017.

[40] M. W. Mahone and P. Drineas. Structural properties underlying high-quality randomized numerical linear algebra algorithms. In M. K. P. Buhlmann, P. Drineas and M. van de Laan, editors, *Handbook of Big Data*, pages 137–154. Chapman and Hall, 2016.

[41] F. 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.

[42] F. Mémoli. On the use of Gromov-Hausdorff distances for shape comparison. *Eurographics Symposium on Point-Based Graphics*, pages 81–90, 2007.

[43] F. Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11(4):417–487, 2011.

[44] F. Mémoli and T. Needham. Gromov-Monge quasi-metrics and distance distributions. arXiv preprint arXiv:1810.09646, 2020.

[45] F. Mémoli and G. Sapiro. Comparing point clouds. *Proceedings of the Eurographics/ACM SIGGRAPH Symposiumon Geometry Processing*, pages 32–40, 2004.

[46] F. Mémoli and G. Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Foundations of Computational Mathematics*, 5:313–347, 2005.

[47] F. Memoli, A. Sidiropoulos, and K. Singhal. Sketching and clustering metric measure spaces. arXiv preprint arXiv:1801.00551, 2018.

[48] J. Milnor. *Morse Theory*. Princeton University Press, New Jersey, 1963.

[49] B. Ordozgoiti, S. G. Canaval, and A. Mozo. A fast iterative algorithm for improved unsupervised feature selection. *IEEE 16th International Conference on Data Mining*, pages 390–399, 2016.

[50] G. Peyré, M. Cuturi, and J. Solomon. Gromov-Wasserstein averaging of kernel and distance matrices. *Proceedings of the 33rd International Conference on Machine Learning, PMLR*, 48:2664–2672, 2016.

[51] J. M. Phillips. Coresets and sketches. In *Handbook of Discrete and Computational Geometry*, chapter 48. CRC Press, 3rd edition, 2016.

[52] S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), 2004.

[53] T. Sarlós. Improved approximation algorithms for large matrices via random projections. *Proceedings of 47th IEEE Symposium on Foundations of Computer Science*, pages 143–152, 2006.

[54] R. Sridharamurthy, T. B. Masood, A. Kamakshidasan, and V. Natarajan. Edit distance between merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[55] K.-T. Sturm. The space of spaces: curvature bounds and gradient flows on the space of metric measure spaces. arXiv preprint arXiv:1208.0434, 2012.

[56] V. Titouan, N. Courty, R. Tavenard, and R. Flamary. Optimal transport for structured data with application on graphs. *International Conference on Machine Learning*, pages 6275–6284, 2019.

[57] V. Titouan, R. Flamary, N. Courty, R. Tavenard, and L. Chapel. Sliced Gromov-Wasserstein. *Advances in Neural Information Processing Systems*, pages 14726–14736, 2019.

[58] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.

[59] H. Xu, D. Luo, and L. Carin. Scalable Gromov-Wasserstein learning for graph partitioning and matching. *Advances in Neural Information Processing Systems*, pages 3046–3056, 2019.

[60] H. Xu, D. Luo, H. Zha, and L. Carin. Gromov-Wasserstein learning for graph matching and node embedding. *International Conference on Machine Learning*, pages 6932–6941, 2019.

[61] P. Zhan, G. Krokos, D. Guo, and I. Hoteit. Three-dimensional signature of the Red Sea eddies and eddy-induced transport. *Geophysical Research Letters*, 46(4):2167–2177, 2019.

[62] P. Zhan, A. C. Subramanian, F. Yao, and I. Hoteit. Eddies in the red sea: A statistical and dynamical study. *Journal of Geophysical Research*, 119(6):3909–3925, 2014.

## A Experimental Results with LSST

For comparative purposes, we describe sketching results for *Heated Cylinder* dataset using low-stretch spanning trees (LSST). The reconstructed sketched trees and basis trees using LSST are visually less appealing compared to the reconstruction using MST. As shown in Fig. 17, the star-like features in the sketched trees are most likely a consequence of the petal decomposition algorithm of LSST [3].

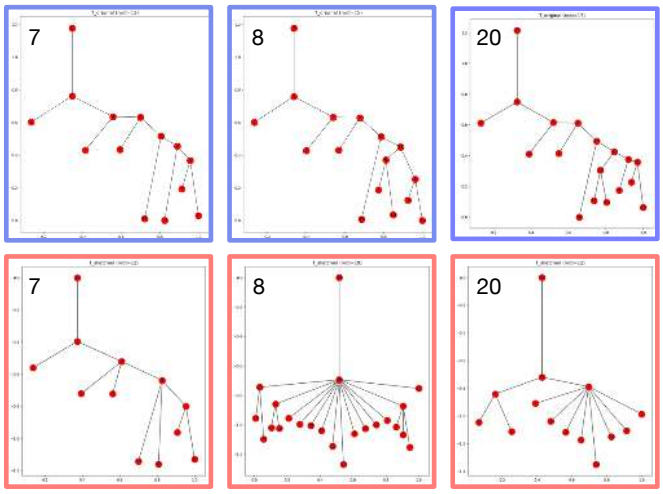


Fig. 17. Sketched trees from the *Heated Cylinder* dataset constructed with LSST based on *IFS*.

On the other hand, the weight matrices show that the LSST preserves some structures within the distance matrices, e.g., for $T_{20}$, before (blue box) and after (red box) sketching, see Fig. 18.

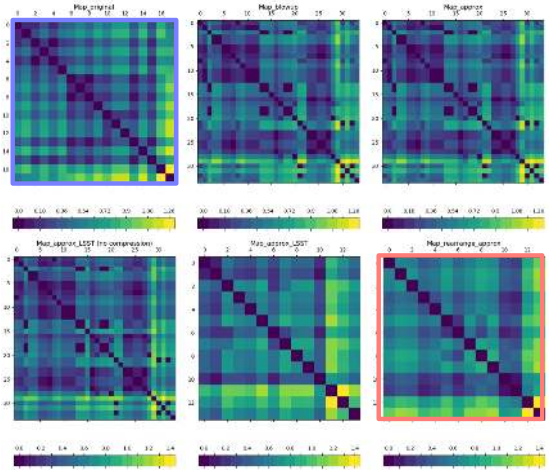

Fig. 18. Sketching the *Heated Cylinder* dataset with 3 basis trees. Weight matrices associated with $T_{20}$ during the sketching process. Configuration: *IFS* with LSST.

## B 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 [17]. 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 $\overline{T}$ is an optimization process, but the blow-up of $T$ and its alignment to $\overline{T}$ 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 $\varepsilon = \|A - \hat{A}\|_F$. Let $A_k$ denote the (unknown) best rank-$k$ approximation of $A$. In the case of CSS, the theoretical upper bound is given as a multiplicative error of the form $\varepsilon \leq \varepsilon_k \cdot \|A - A_k\|_F$, where $\varepsilon_k$ depends on the choice of $k$ [12, 20], or it is given as an additive error $\varepsilon \leq \|A - A_k\|_F + \varepsilon_{k,A}$, where $\varepsilon_{k,A}$ depends on $k$ and $\|A\|_F$ [21, 40]. $\|A - A_k\|_F$ is often data dependent. In the case of NMF, a rigorous theoretical upper bound on $\varepsilon$ 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 Sect. 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.

## C Implementation Details

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

**Initializing the coupling probability distribution.** In Sect. 4, we introduce the blowup procedure that transforms a merge tree $T$ to a larger tree $T'$. This procedure optimizes the probability of coupling between $T$ and $\overline{T}$, the Fréchet mean. Since the optimization process is finding a coupling matrix that is a local minimum of the loss function, similar input trees may give different coupling matrices due to the optimization process. This may affect the ordering of nodes in the blown-up trees, leading to completely different vectorization results and large sketch errors. Specifically for time-varying data, to ensure that adjacent trees are initialized with similar coupling probabilities *w.r.t.* $\overline{T}$, we use the coupling probability between $T_{i-1}$ and $\overline{T}$ to initialize the coupling probability between $T_i$ and $\overline{T}$, for $1 \leq i \leq N-1$. This strategy is based on the assumption that merge trees from adjacent time instances share similar structures.

**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 (*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 (*IFS*)

1. Choose a subset of $k$ column indices $r = \{i_1, i_2, \ldots, i_k\}$ uniformly at random.
2. Construct subset $B_r = [a_{i_1}, a_{i_2}, \ldots, a_{i_k}]$ of $A$ with columns indexed by $r$.
3. Repeat for $j = 1, 2, \ldots, k$:
   (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 $w = \text{argmin}_{l \in [n] \setminus r} \|A - X_{jl} X_{jl}^+ A\|_F$.
   (c) $B_r \leftarrow X_{jw}$.
   (d) $r \leftarrow (r \setminus \{i_j\}) \bigcup \{w\}$.

- 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 [11].
2. Normalize columns of $B$ and $X$ to unit $L_2$ norm. Let $E = A - BX^T$.
3. Repeat until convergence: for $j = 1, 2, \ldots, k$,
   (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 $x_0 \to x_1 \to \cdots \to x_k = t$ 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 blown-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, $f : V \to \mathbb{R}$, 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 is determined by the left-to-right ordering of the child nodes. We consider to order the child nodes that share the same parent node by using a heuristic strategy described below.
   (a) Sort the child nodes by their size of the subtrees of which the child node is the root in ascending order. This is trying to keep larger subtrees on the right so the overall shape of the tree is protected and straightforward to read.
   (b) If the sizes of multiple subtrees are the same, we apply the following strategy: we sort child nodes by their distances to the parent node in descending order. Suppose the order of child nodes after sorting is $c_1, c_2, \ldots, c_t$. If $t$ is odd, we reorder the nodes from left to right as $c_t, c_{t-2}, c_{t-4}, \ldots, c_3, c_1, c_2, c_4, \ldots, c_{t-3}, c_{t-1}$. If $t$ is even, we reorder the nodes as $c_{t-1}, c_{t-3}, c_{t-5}, \ldots, c_3, c_1, c_2, c_4, \ldots, c_{t-2}, c_t$.

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 obtain the root node of the sketched tree by keeping track of the root node during the entire sketching process. We can get the root node of $T'$ because it is either a duplicate node or the same node of the root node in $T$. Then we can get the root node in $\hat{T}'$, as the labels in the sketched blown-up tree are identical to $T'$. Lastly, by keeping track of the process of merge tree simplification, we can know the label of the root of $\hat{T}$.

**Other implementation details.** Our framework is mainly implemented in Python. The code to compute LSST and MST from a given weight matrix is implemented in Java. 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 [17] requires the Python Optimal Transport (POT) package.

## D Detailed Error Analysis

In Fig. 8, we see that all global sketch errors and most global GW losses decrease as the number of basis trees $k$ increases. The decrease of global sketch errors is not surprising as this is a direct consequence of matrix sketching when $k$ increases. For the *Heated Cylinder* dataset, we see that the global GW loss does not decrease drastically for $k \geq 3$. This is because that almost all input trees are well sketched at $k = 3$,

| Sketching Method | $k$ | GW loss MST | GW loss LSST | Sketch error |
|---|---|---|---|---|
| *NMF* | 2 | 1.4435 | 0.5233 | 123.1844 |
| | 3 | 0.3420 | 0.5138 | 38.1245 |
| | 4 | 0.4892 | 0.4992 | 27.7735 |
| | 5 | 0.2927 | 0.3906 | 12.1199 |
| *IFS* | 2 | 0.4581 | 0.7500 | 156.4803 |
| | 3 | 0.1756 | 0.5422 | 46.8917 |
| | 4 | 0.1514 | 0.5453 | 28.1174 |
| | 5 | 0.1159 | 0.4944 | 10.0892 |
| *LSS* | 2 | 1.7292 | 0.6311 | 247.4370 |
| | 3 | 0.1574 | 0.6258 | 52.3285 |
| | 4 | 0.1157 | 0.6743 | 34.1059 |
| | 5 | 0.1416 | 0.5837 | 14.4842 |

Table 1. GW losses and sketch errors of sketching the *Heated Cylinder* dataset with increasing $k$.

| Sketching Method | $k$ | GW loss MST | GW loss LSST | Sketch error |
|---|---|---|---|---|
| *NMF* | 5 | 46.0725 | 16.6348 | 23217.9002 |
| | 10 | 27.2971 | 13.3785 | 10359.9521 |
| | 15 | 21.2988 | 12.4028 | 5119.7659 |
| | 20 | 14.7962 | 12.2321 | 2724.0766 |
| | 25 | 13.3384 | 11.9885 | 1781.4510 |
| | 30 | 13.2442 | 12.0104 | 1189.0048 |
| *IFS* | 5 | 37.9071 | 16.9597 | 29547.6731 |
| | 10 | 12.8413 | 13.2241 | 12332.1347 |
| | 15 | 6.5156 | 12.7953 | 4780.1059 |
| | 20 | 4.5175 | 9.4656 | 2023.8101 |
| | 25 | 3.3680 | 7.5935 | 888.4257 |
| | 30 | 2.9452 | 5.9000 | 412.4219 |
| *LSS* | 5 | 71.8523 | 19.8988 | 39499.2563 |
| | 10 | 33.5862 | 15.0740 | 21475.1222 |
| | 15 | 8.6099 | 11.5266 | 8254.5006 |
| | 20 | 5.5886 | 9.2652 | 3041.7406 |
| | 25 | 4.0473 | 8.2447 | 1280.3685 |
| | 30 | 2.9364 | 9.4942 | 725.1093 |

Table 2. GW losses and sketch errors of sketching the *Corner Flow* dataset with increasing $k$.

| Sketching Method | $k$ | GW loss MST | GW loss LSST | Sketch error |
|---|---|---|---|---|
| *NMF* | 5 | 6.9777 | 0.6956 | 933.3919 |
| | 10 | 4.9383 | 0.5673 | 627.3957 |
| | 15 | 2.8533 | 0.5011 | 440.2140 |
| | 20 | 2.5431 | 0.5047 | 329.9397 |
| | 25 | 1.5934 | 0.4128 | 246.5278 |
| | 30 | 1.3152 | 0.3717 | 183.8884 |
| *IFS* | 5 | 3.7679 | 0.7960 | 1230.5378 |
| | 10 | 2.1023 | 0.5802 | 830.7574 |
| | 15 | 2.3979 | 0.5181 | 553.0398 |
| | 20 | 1.4692 | 0.3485 | 397.0191 |
| | 25 | 1.0142 | 0.2724 | 283.8923 |
| | 30 | 0.7326 | 0.2579 | 201.8570 |
| *LSS* | 5 | 3.8604 | 0.8583 | 1629.3500 |
| | 10 | 2.2035 | 0.5904 | 910.2818 |
| | 15 | 2.3660 | 0.4774 | 603.2064 |
| | 20 | 1.6389 | 0.4137 | 437.7015 |
| | 25 | 1.1700 | 0.3275 | 313.6370 |
| | 30 | 0.8721 | 0.3665 | 220.9125 |

Table 3. GW losses and sketch errors of sketching the *Red Sea* dataset with increasing $k$.

| Dataset | Sketching Method | $k$ | Avg. GW loss MST | Avg. GW loss LSST | Avg. Sketch error |
|---|---|---|---|---|---|
| *Heated Cylinder* | *NMF* | 3 | 0.3420 | 0.4825 | 38.1245 |
| | | 5 | 0.2927 | 0.4247 | 12.1199 |
| | *IFS* | 3 | 0.1756 | 0.5981 | 46.8917 |
| | | 5 | 0.1159 | 0.4736 | 10.0892 |
| | *LSS* | 3 | 0.1574 | 0.7299 | 52.3285 |
| | | 5 | 0.1416 | 0.5152 | 14.4842 |
| *Corner Flow* | *NMF* | 15 | 21.9481 | 12.8625 | 5119.6605 |
| | | 30 | 13.2744 | 12.1794 | 1188.9742 |
| | *IFS* | 15 | 6.5156 | 12.0558 | 4780.1059 |
| | | 30 | 2.9452 | 6.0414 | 412.4219 |
| | *LSS* | 15 | 8.6099 | 11.0902 | 8254.5006 |
| | | 30 | 2.9364 | 9.1941 | 725.1093 |
| *Red Sea* | *NMF* | 15 | 3.2322 | 0.4936 | 442.6564 |
| | | 30 | 1.3653 | 0.4398 | 188.4722 |
| | *IFS* | 15 | 2.3979 | 0.5362 | 553.0398 |
| | | 30 | 0.7600 | 0.2706 | 201.8789 |
| | *LSS* | 15 | 2.3660 | 0.5181 | 603.2064 |
| | | 30 | 0.8721 | 0.3512 | 220.9125 |

Table 4. Average GW loss and Sketch error of 10 different runs for the sketching algorithms.

resulting in small column-wise GW losses. This is the intuition behind our "elbow method".

In terms of the global sketch error, *LSS* strategy appear to have the worst performance when $k$ is small, while *NMF* consistently performs the best for the *Red Sea* dataset with the most complicated topological variations. *IFS* and *LSS* overall have similar performances in all datasets, while *IFS* usually performs slightly better than *LSS* when $k$ is small. We report below the exact errors for a single run (with a fixed seed for the randomization) across increasing $k$ values for the three datasets. We compare across three sketching techniques, *NMF*, *LSS*, and *IFS*. We compare GW losses obtained using both MST and LSST strategy.

***Heated Cylinder.*** In Table 1, we see that global GW losses with MST become stable for $k \geq 3$. However, the global GW losses with LSST do not converge as $k$ increases to 5. This is partially due to the fact that LSST does not recover the merge tree as well as the MST. Among three sketching methods, at $k = 3$, *IFS* and *LSS* have better performance than *NMF* for $k \geq 3$ *w.r.t.* the GW loss, while *NMF* has the best performance on the sketch error.

***Corner Flow.*** In Table 2, sketch errors decrease drastically as $k$ increases. For the two CSS sketching methods – *IFS* and *LSS* – the "elbow" point of the GW loss with MST is at $k = 15$. Therefore we report our results using $k = 15$ basis trees. Based on the performance of GW loss for $k \geq 15$, *IFS* performs the best among the three sketching methods.

***Red Sea.*** In Table 3, the "elbow" point is not as obvious. This is because the *Red Sea* dataset is the hardest to sketch, as the input trees contain significantly diverse topological structures. Similar to the results of other two datasets, *IFS* has overall the best performance *w.r.t.* the GW loss, *LSS* the second, and *NMF* the worst.

**Average performance across multiple runs.** Our pipeline includes randomization to set initial states, including *NMF*, *IFS*, and the LSST algorithm. Therefore, we report the average GW loss and sketch error across 10 runs, each with a distinct random seed. By comparing Table 4 with Table 1, Table 2, and Table 3, we see that the average GW losses and sketch errors across 10 different runs are close to the results of a single run with a fixed random seed. This shows that our pipeline has a reasonably stable performance on sketching merge trees.