informs
http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0899-1499, EISSN 1526-5528

Submitted to *INFORMS Journal on Computing*

Sempervirens: A Fast Reconstruction Algorithm for Noisy and Incomplete Binary Matrix Representations of Trees

Neelay Junnarkar*,
Department of Electrical Engineering and Computer Sciences UC Berkeley,
neelay.junnarkar@berkeley.edu

Can Kızılkale* †
Department of Electrical Engineering and Computer Sciences UC Berkeley and Lawrence Berkeley National Laboratory,
cankizilkale@berkeley.edu

Nevena Golubovic
Department of Computer Science UC Santa Barbara, nevena@cs.ucsb.edu,

Murat Arcak
Department of Electrical Engineering and Computer Sciences UC Berkeley, arcak@berkeley.edu,

Aydın Buluç
Department of Electrical Engineering and Computer Sciences UC Berkeley and Lawrence Berkeley National Laboratory, abuluc@lbl.gov.

Abstract. Applications such as reconstructing cell lineage trees (represented as phylogenetic trees) from single-cell sequencing data require reconstructing a $\{0,1\}$-matrix that has many errors and missing entries. We introduce Sempervirens, a very fast matrix reconstruction algorithm for noisy and incomplete matrix representations of phylogenetic trees. Sempervirens uses an iterative maximum-likelihood approach to determine the topology tree represented by the corrupted data. We show that Sempervirens is at least three orders of magnitude faster than other methods on thousand by thousand matrices with the speed gap widening with larger matrices. We also show that Sempervirens matches state-of-the-art methods in reconstruction accuracy. The speed of Sempervirens enables it to be tractably applied to reconstructing much larger matrices than those that other methods can reconstruct. In addition to experimental results, we justify the algorithm with a mathematical treatment of its subprocedures.

Key words: phylogenetic trees; matrix reconstruction, algorithms
History:

* First authors.

† Corresponding Author

1. Introduction

A phylogenetic tree is a rooted tree[1] that represents the evolutionary relationships among different species or groups of organisms. It is the evolutionary history representation, or phylogeny, of organism sets. Phylogenetic trees can be represented using $\{0,1\}$-matrices, where each column corresponds to a mutation and each row corresponds to an organism. For a thorough exploration of efficient algorithms for determining whether a $\{0,1\}$-matrix represents a phylogenetic tree, and inferring the phylogenetic tree if so, readers are encouraged to consult (Gusfield 1991).

In applications such as reconstructing cell lineage trees from single-cell sequencing data (single-cell sequencing analyzes nucleic acids from individual cells, offering high-resolution insights into cellular differences and functions within their microenvironment), the matrix tends to contain significant numbers of false positives, false negatives, and missing entries (Navin 2014). Thus, we consider the problem of inferring a phylogenetic tree from a corrupted $\{0,1\}$-matrix. It is assumed that the true $\{0,1\}$-matrix represents a phylogenetic tree. However, elements of the observed matrix may be flipped or completely erased. The objective is to reconstruct a $\{0,1\}$-matrix that accurately represents the phylogenetic tree inferred by the true matrix, despite the corruption in the observed data.

Several methods applicable to this problem have been proposed. Branch and bound approaches, like the one proposed in (Sadeqi Azer et al. 2020), aim to minimize the number of flips. These have exponential time complexity and thus are impractical for matrices exceeding several hundred rows and columns. There are also polynomial time algorithms, such as SPhyR (El-Kebir 2018). This method first clusters rows and columns and then employs an integer linear programming (ILP) formulation to construct a conflict-free matrix. The execution time of this ILP formulation grows exponentially in the number of clusters. As shown in (Kızılkale et al. 2022), this approach is inefficient for large matrices with a large number of clusters and often needs a large number of clusters for good reconstructions. ScisTree (Wu 2020) is among the most notable algorithms with polynomial time complexity, giving highly accurate reconstructed matrices. This method combines local search over trees with likelihood calculations to search for the most likely phylogenetic tree. However, it scales poorly with the number of rows. HUNTRESS (Kızılkale et al. 2022) is, to the best of our knowledge, one of the fastest reconstruction algorithms for larger matrices, scaling well with the number of rows. Its accuracy is on par with other state-of-the-art methods while being faster, and it does not require the exact knowledge of false positive and negative probabilities. It scales well with the number of cores, and the running time is independent of the error probabilities. However, it suffers from a lengthy tuning stage, does not exploit the topology of the tree, and does not make use of the distribution information on false positives/negatives fully when it is available.

[1] Rooted tree: a tree where one node is designated as the root, serving as the starting point from which all other nodes and branches originate

In this paper, we introduce Sempervirens, a likelihood-based approach for reconstructing phylogenetic trees from corrupted data. Sempervirens leverages the tree topology in the data to find subtrees with high accuracy while eliminating any need for tuning. We show that not only is Sempervirens orders of magnitude faster than the fastest state-of-the-art methods, but also that it matches these methods in reconstruction accuracy. This enables Sempervirens to handle very large matrices that are not computationally tractable with the state-of-the-art methods.

The remainder of the paper is structured as follows. Section 2 summarizes the relationship between a $\{0,1\}$-matrix and a phylogenetic tree, and formalizes the model of data corruption. Section 3 presents our algorithm for reconstructing phylogenetic trees from noisy matrices. Section 4 analyzes optimality of sub-procedures of this algorithm, and outlines the algorithm time complexity. Section 5 presents experimental results, demonstrating the reconstruction accuracy, speed, and robustness of this algorithm in comparison to state-of-the-art algorithms.

2. Modeling

In this section, after a brief presentation of notation, we describe the relationship between a phylogenetic tree and a $\{0,1\}$-matrix, including an illustrative example, and the model of noise we use for corruption, whereby entries are removed and false positives and negatives are introduced.

2.1. Notation

For a matrix M, we denote the i'th row by $M(i,:)$ and the j'th column by $M(:,j)$. For a vector x, we denote the i'th element by $x(i)$. We denote matrices and vectors with elements in $\{0,1\}$ as $\{0,1\}$-matrices and $\{0,1\}$-vectors respectively. We interpret $\{0,1\}$-vectors as indicator functions for the set of objects present and, by a slight abuse of notation, apply set operations to $\{0,1\}$-vectors: for $\{0,1\}$-vectors x and y of the same length, we represent subset, superset, union, intersection, and set difference ($\{i|x(i)=1\}\{\subseteq,\supseteq,\cup,\cap,\setminus\}\{i|y(i)=1\}$) by $x\{\subseteq,\supseteq,\cup,\cap,\setminus\}y$. For example, $\{i|x(i)=1\}\cup\{i|y(i)=1\}$ is represented by $x\cup y$. Additionally, we represent the cardinality of $\{i|x(i)=1\}$ by $|x|$, and the complement of the set, $\{i|x(i)=0\}$, by x^c. Missing entries are denoted by "$\square$".

2.2. Tree Representation as Matrix

We will be using some of the terminology from (Gusfield 1991) to describe phylogenetic trees and $\{0,1\}$-matrices.

A matrix $M\in\{0,1\}^{n\times m}$ represents n objects described by the absence/presence of m characters. The matrix M is said to be *conflict-free*, or equivalently to "have a phylogenetic tree", if there exists a rooted tree T with $n'\leq n$ nodes such that each character is associated with exactly one edge and each object is associated with exactly one node. The row corresponding to an object is an indicator function for a set of characters whose associated edges form a path in the tree, starting at the tree root node. The object is

associated with the node at the end of that path. Note that an edge can be associated with multiple characters and a node can be associated with multiple objects. This occurs if there are identical columns or identical rows in M respectively. Also note that we allow a conflict-free matrix to refer to a "forest" of phylogenetic trees (i.e. for there to be multiple disconnected phylogenetic trees) since we can always treat the object with all characters absent, whether or not it is in the matrix, as the common root of the phylogenetic trees.

All phylogenetic trees can be represented by a $\{0,1\}$-matrix: for a phylogenetic tree T with n nodes, construct a matrix $M \in \{0,1\}^{n \times n-1}$ such that $M(i,j)$ equals 1 if the path from node i to the root contains edge j and 0 otherwise. However, not all $\{0,1\}$-matrices have a phylogenetic tree: a conflict occurs when there are two columns that are neither completely distinct nor does one completely contain the other.

LEMMA 1. *A $\{0,1\}$-matrix M is conflict-free if and only if every column pair $(M(:,i), M(:,j))$ satisfies that $M(:,i) \supseteq M(:,j)$, $M(:,i) \subseteq M(:,j)$, or $M(:,i) \cap M(:,j) = \emptyset$.*

A proof of this is available in (Meacham 1983).

If the matrix M has a phylogenetic tree T, then $M(:,i)$ can be interpreted as the objects associated with the nodes that are in the subtree T_S of T such that the first edge on the path from the root node of T_S to the root node of T is associated with character i. We call the T_S the subtree of column $M(:,i)$, or the subtree of character i, and the edge associated with i the *lead edge* of subtree T_S.

See Figure 1 for an example of a conflict-free matrix and its corresponding phylogenetic tree.

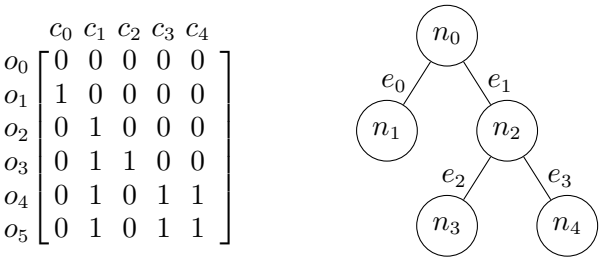

Figure 1 **A conflict-free matrix and its phylogenetic tree. This matrix representation shows six objects (rows), $o_0, \ldots, o_5$, described by five characters (columns), $c_0, \ldots, c_4$. The corresponding phylogenetic tree consists of five nodes, $n_0, \ldots, n_4$, and four edges, $e_0, \ldots, e_3$. The nodes and the objects they represent are $n_0 = \{o_0\}$, $n_1 = \{o_1\}$, $n_2 = \{o_2\}$, $n_3 = \{o_3\}$ and $n_4 = \{o_4, o_5\}$, and the edges and the characters they represent are $e_0 = \{c_0\}$, $e_1 = \{c_1\}$, $e_2 = \{c_2\}$ and $e_3 = \{c_3, c_4\}$.**

2.3. Corruption Process

We model the corruption process through which a noisy matrix $\hat{M}$ is formed from the ground-truth matrix, which is conflict-free and consists of only 0s and 1s, using false positive probability α, false negative probability β, and missing entry probability γ. The transition diagram is depicted in Figure 2. We model the corruption of the ground-truth matrix as a single step of this transition diagram. The noisy matrix $\hat{M}$ represents the measured data used as input in a reconstruction algorithm.

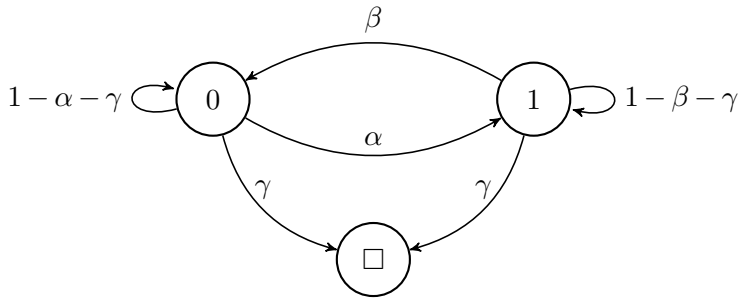

Figure 2 **Transition probabilities between state 0, representing the absence of a character, state 1, representing the presence of a character, and state □, representing missing information on a character. Note that the ground-truth matrix contains only 0s and 1s so the transitions from □ are omitted. The false positive probability is α, the false negative probability is β, and the missing entry probability is γ.**

3. Algorithm

In this section, we introduce Sempervirens, an algorithm for reconstructing a conflict-free matrix M from a noisy matrix $\hat{M}$. The only parameters for this algorithm are the noisy matrix $\hat{M}$, the false positive probability α, the false negative probability β, and the missing entry probability γ.

3.1. Main Routine

Pseudocode for Sempervirens is given in Algorithm 1. It has two major components: a procedure to construct a conflict-free matrix containing candidate reconstructed columns, and a maximum likelihood refinement process that matches these columns with the noisy columns. An illustration of the reconstruction process is given in Figure 3.

Algorithm 1: Sempervirens

Input: Noisy matrix $\hat{M} \in \{0,1,\square\}^{n\times m}$, false positive probability α, false negative probability β, and missing entry probability γ.

Output: Conflict-free matrix $M \in \{0,1\}^{n\times m}$.

$M \leftarrow \hat{M}$;

$M(i,j) \leftarrow 0$ for (i,j) such that $\hat{M}(i,j)$ is unknown;

$P \leftarrow \{1,\ldots,m\}$;

RECONSTRUCT(P, M, α, β);

$M \leftarrow$ MAXIMUMLIKELIHOODREFINEMENT$(M, \hat{M}, \alpha, \beta, \gamma)$;

return M;

Note that the missing elements are simply replaced with zeros before starting reconstruction. We now go over the sub-procedures of Algorithm 1. Analysis of some of the decision rules is left to Section 4.

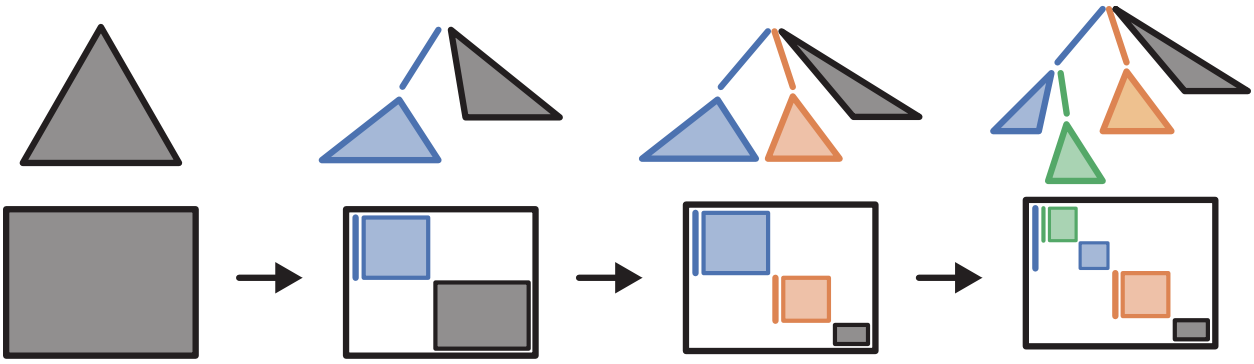

Figure 3 **Illustration of the matrix reconstruction process, with the corresponding tree at each step. Shaded and colored matrix areas represent possible non-zero values, while white areas represent zeros. Initially, non-zero values may be anywhere. Shaded triangles in the tree represent non-reconstructed subtrees. Each call to RECONSTRUCT takes a block of the matrix, reconstructs a column, places it in the matrix (represented by a line), and divides the block into two sub-blocks: one block of columns that are subsets of the reconstructed column (shaded the same color as the reconstructed column), and one block of columns that are disjoint with the reconstructed column (shaded the color of the original block). In this illustration, the first call to RECONSTRUCT selects the black block and constructs a column corresponding to the blue edge (top-left column/block and left branch in second figure). The second call selects the black block and constructs the orange column (middle block and branch in third figure). The third call selects the blue block and constructs the green column (top-left new column/block and second-from-the-left branch in final figure). This continues until all columns are reconstructed. Note that a reconstructed matrix will only appear blocky after applying an appropriate permutation to the rows and columns.**

3.1.1. Reconstruct: We now go over the first major component, RECONSTRUCT, of Algorithm 1, which constructs a conflict-free matrix from a set of columns P of the $\{0,1\}$-matrix M given the false positive and false negative probabilities α and β. Pseudocode for this algorithm is in Algorithm 2. For terminology, we call a subset of a set of columns a *subtree* if all columns in the subset belong to a subtree that is contained in the set. We first give an overview of the RECONSTRUCT procedure and then detail its subprocedures.

This procedure operates as follows. The column set P is partitioned into two: columns representing the largest subtree in P, and its complement. The leading edge of this largest subtree is reconstructed and replaces the most similar column of the matrix in the subtree. This similar column is removed from the set representing the largest subtree. The function then recurses on the remaining columns in the subtree and on its complement in P. The recursion terminates when the set P is empty.

We now detail the subprocedures of RECONSTRUCT. Please refer to the references for how these subprocedures are used in relation to each other.

- FINDLARGESTMAXIMALSUBTREE: See Algorithm 3. A subtree is *maximal* when there is no other subtree of the subgraph, derived from the given set of columns, that strictly contains it. This procedure

Algorithm 2: Reconstruct a set of columns of the matrix.

Input: Set of columns P, matrix M, false positive probability α, false negative probability β.

Function RECONSTRUCT(P, M, α, β) **is**
  if $|P| = 0$ **then**
    return; `// Base case.`
  end
  $S_0 \leftarrow$ FINDLARGESTMAXIMALSUBTREE(P, M, α, β);
  $M \leftarrow$ ROOTNODEBASEDRECONSTRUCTION(S_0, M, β);
  $e \leftarrow$ RECONSTRUCTSUBTREELEADEDGE(S_0, P, M);
  $S \leftarrow$ COLUMNSINSUBTREEOFNOISELESSPIVOT(e, P, M);
  $M(:, P) \leftarrow$ BACKWARDSPASS(e, S, P, M);
  $u \leftarrow$ SELECTCOLUMNTOREPLACE(e, P, M);
  $M(:, u) \leftarrow e$;
  RECONSTRUCT($S \setminus \{u\}$, M, α, β);
  RECONSTRUCT($P \setminus (S \cup \{u\})$, M, α, β);
end

divides the input column set into maximal subtrees and returns the maximal subtree with the greatest number of elements.

• COLUMNSINSUBTREEOFNOISYPIVOT: Given a noisy pivot column and a set of non-reconstructed columns, this function identifies which columns in the set are likely in the same subtree as the pivot column according to the criteria outlined in Section 4.

• ROOTNODEBASEDRECONSTRUCTION: The root node of a subtree, which can be interpreted as the list of characters present in all objects of the subtree, is reconstructed using the set of columns in the subtree. A character is determined to be present in the root node if the number of objects which have the character is at least $1 - \beta$ times the maximum number of objects any character in the subtree has. Then, rows of the matrix that are in the subtree are modified to be supersets of the root node. A row is considered to be in the subtree if the size of its intersection with the root is at least half the size of the root.

• RECONSTRUCTSUBTREELEADEDGE: Given a maximal subtree and the set of non-reconstructed columns which have possibly non-zero intersection with columns in the subtree, this procedure reconstructs the most likely lead edge column. This is the column that maximizes the probability that it is a superset of every column in the subtree and disjoint with every non-reconstructed column outside the subtree. The decision rules are described in Section 4.

• COLUMNSINSUBTREEOFNOISELESSPIVOT: Given a reconstructed column and a set of non-reconstructed columns, this function identifies which columns in the set are likely in the subtree whose lead edge is the reconstructed column, according to criteria outlined in Section 4.

• BACKWARDSPASS: See Algorithm 4. This function takes the reconstructed lead edge, its associated subtree of non-reconstructed columns, and the set of non-reconstructed columns which have possibly non-empty intersection with the lead edge. Each column in the subtree is set to its intersection with the lead edge, and each non-reconstructed column outside the subtree is set to its intersection with the complement of the lead edge. This process ensures that these columns are either disjoint with or subsets of the reconstructed column, which is needed to guarantee a conflict-free reconstruction.

• SELECTCOLUMNTOREPLACE: This function identifies the noisy column that is most likely to be the noisy version of a reconstructed column. This is determined as the noisy column with the largest intersection with the reconstructed column.

Algorithm 3: Find Largest Maximal Subtree

Input: A set of non-reconstructed columns P, matrix M, false positive probability α, false negative probability β.

Output: The maximal subtree S of columns of P with the most elements.

Function FINDLARGESTMAXIMALSUBTREE(P, M, α, β) **is**
  $\mathcal{P} \leftarrow \emptyset$;
  while $|P| > 0$ **do**
    $u_0 \leftarrow \arg\max_{u \in P} |M(:, u)|$;
    $S_0 \leftarrow$ COLUMNSINSUBTREEOFNOISYPIVOT$(u_0, P, M, \alpha, \beta)$;
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{S_0\}$;
    $P \leftarrow P \setminus S_0$;
  end
  $S \leftarrow \arg\max_{S_0 \in \mathcal{P}} |S_0|$;
  return S;
end

3.1.2. Maximum Likelihood Refinement: The second component of Algorithm 1 is a maximum likelihood refinement procedure, MAXIMUMLIKELIHOODREFINEMENT, which takes a conflict-free matrix and treats it as a bag of rows and columns to construct a refined matrix that better matches the original noisy matrix. This procedure is run on the conflict-free matrix produced by RECONSTRUCT.

First, for each row of the noisy matrix, we identify which row of the conflict-free matrix maximizes the probability of measuring the noisy row. A refined matrix is constructed by assigning that row from the conflict-free matrix in place of the noisy row. Second, a similar procedure is applied to the columns of the refined matrix, and the result is returned. Note that this is not strictly a one-to-one matching: multiple rows (or columns) of the conflict-free matrix can be associated with the same row (or column) of the original noisy matrix.

Algorithm 4: Backwards Pass

Input: Column e, a set of non-reconstructed columns P, a subset of columns $S \subseteq P$ in the subtree of e, matrix M, false positive probability α, false negative probability β.

Output: Modified $M(:, P)$.

Function BACKWARDSPASS(e, S, P, M) **is**
  for $i \in S$ **do**
    $M(:, i) \leftarrow M(:, i) \cap e$;
  end
  for $i \in P \setminus S$ **do**
    $M(:, i) \leftarrow M(: i) \cap e^c$;
  end
  return $M(:, P)$;
end

4. Analysis

In this section, we analyze the algorithm, Sempervirens, presented in the previous section. We start by showing that Algorithm 1 produces a conflict-free matrix.

THEOREM 1. *Given a matrix $\hat{M} \in \{0, 1, \square\}^{n \times m}$, false positive probability α, false negative probability β, and missing entry probability γ, Algorithm 1 produces a matrix M that is conflict-free (i.e. has a phylogenetic tree).*

Proof We will show that the call to RECONSTRUCT in Algorithm 1 constructs a conflict-free matrix. This conflict-free matrix is put through maximum-likelihood refinement. Observe that reordering rows and columns, even with replacement, cannot introduce conflicts. Hence, the output of Algorithm 1 is conflict-free.

Let $\mathcal{R}$ be the set of reconstructed columns, and consider a call to RECONSTRUCT with parameters P and M. We will abuse notation and refer to indices of columns in M as the columns themselves. Define the *immediate parent* of a set of non-reconstructed columns P, when it exists, as the column in $\mathcal{R}$, denoted by e_p, such that every column in P is a subset of e_p and any column in $\mathcal{R}$ that is a strict subset of e_p is disjoint with every column in P. Assume that $\mathcal{R}$ is conflict-free and that either every column in P is disjoint with every column in $\mathcal{R}$ or that P has an immediate parent. These assumptions are vacuously satisfied on the first call to RECONSTRUCT since $\mathcal{R}$ is empty.

Note that ROOTNODEBASEDRECONSTRUCTION modifies M by changing 0s to 1s, but only in columns of P, and only in rows that have characters in P. Thus, after these modifications, every column in P is a subset of the union of columns in P pre-modifications, and so the relationship between P and $\mathcal{R}$ is maintained.

The function RECONSTRUCTSUBTREELEADEDGE constructs e such that e is a subset of the union of columns in P. This reconstructed column replaces column $u \in P$ and then $P \setminus \{u\}$ is divided into $S \setminus \{u\}$ and $P \setminus (S \cup \{u\})$ such that, through the backwards pass, every column in $S \setminus \{u\}$ is a subset of e and every column in $P \setminus (S \cup \{u\})$ is disjoint with e. If every column in P is disjoint with every column in $\mathcal{R}$, then since e is a subset of the union of columns in P, it is disjoint with every column in $\mathcal{R}$, so $\mathcal{R} \cup \{e\}$ is conflict-free. Moreover, e is the immediate parent of $S \setminus \{u\}$ and every column in $P \setminus (S \cup \{u\})$ is disjoint with every column in $\mathcal{R}$. If P has an immediate parent e_p, then since e is a subset of the union of columns in P, it is a subset of e_p and also disjoint with any columns in $\mathcal{R}$ that are strict subsets of e_p. Additionally, e is a subset of any columns in $\mathcal{R}$ that e_p is a subset of, and e is disjoint with any columns in $\mathcal{R}$ that e_p is disjoint with. Therefore, $\mathcal{R} \cup \{e\}$ is conflict-free. Moreover, e is the immediate parent of $S \setminus \{u\}$ and e_p is the immediate parent of $P \setminus (S \cup \{u\})$. Hence, the assumptions to recurse on $S \setminus \{u\}$ and on $P \setminus (S \cup \{u\})$ are satisfied in both cases. Furthermore, both $|S \setminus \{u\}|$ and $|P \setminus (S \cup \{u\})|$ are less than $|P|$, so the recursion terminates, and the resulting $\mathcal{R}$ will be conflict-free.

■

4.1. Optimality of Sub-procedures

We analyze decision rules used in COLUMNSINSUBTREEOFNOISYPIVOT, RECONSTRUCTSUBTREE-LEADEDGE, and COLUMNSINSUBTREEOFNOISELESSPIVOT, using a likelihood-based approach. To do so, we make the following independence assumption on the noise.

ASSUMPTION 1. *We assume that each entry of the ground-truth matrix is corrupted independently: for two ground-truth random variable entries x and y, and the noisy random variables $\hat{x}$ and $\hat{y}$, we assume*

$$\Pr\{\hat{x} = a, \hat{y} = b | x = c, y = d\} = \Pr\{\hat{x} = a | x = c\} \Pr\{\hat{y} = b | y = d\}.$$

We additionally make the following mild assumption on the false positive and false negative probabilities.

ASSUMPTION 2. *We assume that both the false positive and false negative probabilities are less than $\frac{1}{2}$.*

This assumption simplifies the maximizations in the proofs of Lemmas 2, 3, and 4. One intuitive implication of this assumption is that, when looking at a single measured entry independently of all other entries, it is most likely that the true element is the same as the measured entry.

Now we introduce a pseudo-likelihood. This construction enables us to determine the most "likely" relationship between the ground-truth columns (or rows), given the corresponding known columns (or rows). Similar to a likelihood, it is only the comparison between two pseudo-likelihoods that is important for determining the relationship, not the value of the pseudo-likelihood itself. We determine the relationship that is most "likely" between the ground-truth columns (or rows) to be the relationship which maximizes this pseudo-likelihood.

DEFINITION 1 (PSEUDO-LIKELIHOOD). For noisy data $\hat{\theta}$, we define the pseudo-likelihood that the ground-truth counterpart θ satisfies a composite relationship ζ as the following:

$$\begin{aligned}\mathcal{L}(\theta \text{ satisfies } \zeta; \hat{\theta}) \triangleq \max_{\theta} \ & \Pr\{\hat{\theta}|\theta\} \\ & \text{s.t. } \theta \text{ satisfies } \zeta\end{aligned} \tag{1}$$

Going forward, we refer to the above pseudo-likelihood as "likelihood" when it is clear from context. Definition 1 is given on composite set relationships over θ. For example, when $\hat{\theta}$ is $(\hat{x}, \hat{y})$ where $\hat{x}$ and $\hat{y}$ are noisy vectors, θ is (x, y) where x and y are the true counterparts of $\hat{x}$ and $\hat{y}$, and ζ represents $x \subseteq y$, we can apply Definition 1 to determine the likelihood of x being a subset of y as the following.

$$\begin{aligned}\mathcal{L}(x \subseteq y; \hat{x}, \hat{y}) &= \max_{x,y} \ \Pr\{\hat{x}|x\} \Pr\{\hat{y}|y\} \\ &\quad \text{s.t. } x \subseteq y \\ &= \max_{x,y} \ \Pi_i \Pr\{\hat{x}(i)|x(i)\} \Pr\{\hat{y}(i)|y(i)\} \\ &\quad \text{s.t. } x \subseteq y\end{aligned}$$

We now go over the decision rules used in COLUMNSINSUBTREEOFNOISYPIVOT, RECONSTRUCTSUBTREELEADEDGE, COLUMNSINSUBTREEOFNOISELESSPIVOT, and ROOTNODEBASEDRECONSTRUCTION. We show that, in each step of the reconstruction, we maximize the likelihood in our decisions. While for high false positive/negative probabilities we may still have errors in the order of the reconstructed columns, we see in the experimental results that the cumulative error does not have a significant negative effect in the resulting reconstructed matrix.

4.1.1. Columns in Subtree of Noisy Pivot: We now apply the definition of pseudo-likelihood to COLUMNSINSUBTREEOFNOISYPIVOT, which accepts a "pivot" column as input and identifies the set of columns representing the maximal subtree that includes this pivot. This process involves comparing a noisy column with the pivot to determine if they belong to the same subtree representation. The likelihood-derived decision rules for this procedure are presented in Lemma 2. By following these decision rules, COLUMNSINSUBTREEOFNOISYPIVOT generates a precise maximal subtree containing the pivot.

LEMMA 2. *Let $\hat{x}$ and $\hat{y}$ be noisy $\{0,1\}$-columns, let x and y be their unknown true counterparts, and let α and β be the false positive and false negative probabilities respectively. If $\beta \geq \alpha$, then $x \subseteq y$ is more likely than $x \cap y = \emptyset$ if $|\hat{x} \cap \hat{y}| \log\left(\frac{1-\beta}{\alpha}\right) \geq |\hat{x} \cap \hat{y}^c| \log\left(\frac{1-\alpha}{\beta}\right)$, and $x \supseteq y$ is more likely than $x \cap y = \emptyset$ if $|\hat{x} \cap \hat{y}| \log\left(\frac{1-\beta}{\alpha}\right) \geq |\hat{x}^c \cap \hat{y}| \log\left(\frac{1-\alpha}{\beta}\right)$. If $\alpha > \beta$, then $x \subseteq y$ is more likely than $x \cap y = \emptyset$ if $|\hat{x} \cap \hat{y}| \geq |\hat{x} \cap \hat{y}^c|$, and $x \supseteq y$ is more likely than $x \cap y = \emptyset$ if $|\hat{x} \cap \hat{y}| \geq |\hat{x}^c \cap \hat{y}|$.*

Proof By Lemma 1, since x and y are columns of a conflict-free matrix, the possible relationships between x and y are: $x \subseteq y$, $x \supseteq y$, and $x \cap y = \emptyset$. We proceed by computing the likelihood of each and showing when either $x \subseteq y$ or $x \supseteq y$ are more likely than $x \cap y = \emptyset$.

First, we compute the likelihood that $x \subseteq y$:

$$\mathcal{L}(x \subseteq y; \hat{x}, \hat{y}) = \max_{x \subseteq y} \Pi_i \Pr\{\hat{x}(i)|x(i)\} \Pr\{\hat{y}(i)|y(i)\}. \tag{2}$$

To do this, we find a maximizer (x, y) and compute $\Pr\{x, y|\hat{x}, \hat{y}\}$. Since the probabilities are independent, we pick $(x(i), y(i))$ to maximize $\Pr\{\hat{x}(i)|x(i)\} \Pr\{\hat{y}(i)|y(i)\}$ for each i. By Assumption 2, $\alpha, \beta < \frac{1}{2}$, so $\Pr\{\hat{x}(i) = 1|x(i) = 0\} = \alpha \leq 1 - \beta = \Pr\{\hat{x} = 1|x(i) = 1\}$ and $\Pr\{\hat{x}(i) = 0|x(i) = 1\} = \beta \leq 1 - \alpha = \Pr\{\hat{x} = 0|x(i) = 0\}$, and similar for y. Hence, for any i such that $\hat{x}(i) \leq \hat{y}(i)$, picking $x(i) = \hat{x}(i)$ and $y(i) = \hat{y}(i)$ will maximize the likelihood.

When $\hat{x}(i) > \hat{y}(i)$ (i.e. $(\hat{x}(i), \hat{y}(i)) = (1, 0)$), the pairs $(x(i), y(i))$ could be any of $(0, 0)$, $(0, 1)$, and $(1, 1)$. For each of these, we compute $\Pr\{\hat{x}(i)|x(i)\} \Pr\{\hat{y}(i)|y(i)\}$ and choose the maximizing pair $(x(i), y(i))$: $(0, 0)$ gives $\Pr\{1|0\} \Pr\{0|0\} = \alpha(1 - \alpha)$, $(0, 1)$ gives $\Pr\{1|0\} \Pr\{0|1\} = \alpha\beta$, and $(1, 1)$ gives $\Pr\{1|1\} \Pr\{0|1\} = (1 - \beta)\beta$. Under Assumption 2 where $\alpha, \beta < \frac{1}{2}$, the maximizer of this triple is $(1, 1)$ when $\beta \geq \alpha$, and $(0, 0)$ when $\alpha > \beta$. For brevity, we will complete the proof for the case where $\beta \geq \alpha$, but the likelihoods can be derived by the same methods for the case where $\alpha < \beta$.

Plugging in the maximizer $(x(i), y(i)) = (\hat{x}(i), \hat{y}(i))$ if $\hat{x}(i) \leq \hat{y}(i)$ and otherwise $(x(i), y(i)) = (1, 1)$ gives the following likelihood. For brevity of notation, define $k_{00} = |\hat{x}^c \cap \hat{y}^c|, k_{01} = |\hat{x}^c \cap \hat{y}|$, $k_{10} = |\hat{x} \cap \hat{y}^c|$, and $k_{11} = |\hat{x} \cap \hat{y}|$.

$$\begin{aligned}
\mathcal{L}(x \subseteq y; \hat{x}, \hat{y}) &= (\Pr\{0|0\} \Pr\{0|0\})^{k_{00}} (\Pr\{0|0\} \Pr\{1|1\})^{k_{01}} (\Pr\{1|1\} \Pr\{0|1\})^{k_{10}} (\Pr\{1|1\} \Pr\{1|1\})^{k_{11}} \\
&= (1 - \alpha)^{2k_{00}} ((1 - \alpha)(1 - \beta))^{k_{01}} ((1 - \beta)\beta)^{k_{10}} (1 - \beta)^{2k_{11}} \\
&= \beta^{k_{10}} (1 - \alpha)^{2k_{00} + k_{01}} (1 - \beta)^{k_{01} + k_{10} + 2k_{11}}
\end{aligned}$$

By symmetry,

$$\mathcal{L}(x \supseteq y; \hat{x}, \hat{y}) = \beta^{k_{01}} (1 - \alpha)^{2k_{00} + k_{10}} (1 - \beta)^{k_{10} + k_{01} + 2k_{11}}$$

A similar process can be used to calculate $\mathcal{L}(x \cap y = \emptyset; \hat{x}, \hat{y})$. In this case, the pairs $(\hat{x}(i), \hat{y}(i))$ that violate the relation are those equal to $(1, 1)$. In this case, $(x(i), y(i))$ equaling either $(0, 1)$ or $(1, 0)$ both maximize the likelihood regardless of whether $\beta \geq \alpha$ or $\alpha > \beta$, giving the following likelihood.

$$\mathcal{L}(x \cap y = \emptyset; \hat{x}, \hat{y}) = \alpha^{k_{11}} (1 - \alpha)^{2k_{00} + k_{01} + k_{10}} (1 - \beta)^{k_{01} + k_{10} + k_{11}}$$

We now compare the likelihood that $x \subseteq y$ to the likelihood that $x \cap y = \emptyset$.

$$\mathcal{L}(x \subseteq y; \hat{x}, \hat{y}) \geq \mathcal{L}(x \cap y = \emptyset; \hat{x}, \hat{y})$$
$$\beta^{k_{10}}(1-\alpha)^{2k_{00}+k_{01}}(1-\beta)^{k_{01}+k_{10}+2k_{11}} \geq \alpha^{k_{11}}(1-\alpha)^{2k_{00}+k_{01}+k_{10}}(1-\beta)^{k_{01}+k_{10}+k_{11}}$$
$$k_{11}\log\left(\frac{1-\beta}{\alpha}\right) \geq k_{10}\log\left(\frac{1-\alpha}{\beta}\right)$$

This gives a condition for when $x \subseteq y$ is more likely than $x \cap y = \emptyset$. The condition for $x \supseteq y$ being more likely than $x \cap y = \emptyset$ is given by symmetry.

■

4.1.2. Reconstruct Subtree Lead Edge: The function RECONSTRUCTSUBTREELEADEDGE takes in a maximal subtree and list of non-reconstructed columns and determines the lead edge of the subtree. The lead edge is an indicator vector of the set of objects in the subtree, so the function proceeds by checking whether each object is more likely to be in the subtree or outside of it. Lemma 3 provides optimal decision rules to determine whether an object is in the subtree when the maximal subtree is correct.

LEMMA 3. *Let $\hat{r}$ be a noisy $\{0,1\}$-row, r be its true counterpart, U be a subset of columns, $S \subseteq U$ be a set of columns representing a maximal subtree in U, and let α and β be the false positive and false negative probabilities respectively. Then, r is most likely in the subtree represented by S if $|\hat{r} \cap S| > 0$ and $|\hat{r} \cap S| > |\hat{r} \cap S^u|$.*

Proof For brevity define $k_{a11} \triangleq |\{i|\hat{r}(i) = a, i \in U, i \in S\}|$, $k_{a01} \triangleq |\{i|\hat{r}(i) = a, i \notin U, i \in S\}|$, $k_{a10} \triangleq |\{i|\hat{r}(i) = a, i \in U, i \notin S\}|$, and $k_{a00} \triangleq |\{i|\hat{r}(i) = a, i \notin U, i \notin S\}|$. In a conflict-free matrix, the subset of columns in U and in the set indicated by the true row r, will be a subset of S or will be disjoint from it (S). It is a subset of S if both of the following hold:

$$\mathcal{L}(r \cap U \setminus S = \emptyset, r \cap S \neq \emptyset; \hat{r}, U, S) \geq \mathcal{L}(r \cap U = \emptyset; \hat{r}, U, S),$$
$$\mathcal{L}(r \cap U \setminus S = \emptyset, r \cap S \neq \emptyset; \hat{r}, U, S) \geq \mathcal{L}(r \cap U \setminus S \neq \emptyset, r \cap S = \emptyset; \hat{r}, U, S).$$

Computing and comparing likelihoods follow similarly to the proof of Lemma 2. The likelihood that r has columns in S and not in $U \setminus S$ is as follows.

$$\mathcal{L}(r \cap U \setminus S = \emptyset, r \cap S \neq \emptyset; \hat{r}, U, S) = \min_{r} \ \Pi_{i \in \{1,\ldots,m\}} \Pr\{\hat{r}(i)|r(i)\}$$
$$\text{s.t. } r \cap U \setminus S = \emptyset$$
$$r \cap S \neq \emptyset$$

When $k_{111} > 0$, the maximizer is r such that $r(i) = 0$ for i such that $(\hat{r}(i), U(i), S(i)) = (1,1,0)$ and $r(i) = \hat{r}(i)$ otherwise. This simplifies the likelihood to the following.

$$\mathcal{L}(r \cap U \setminus S = \emptyset, r \cap S \neq \emptyset; \hat{r}, U, S) = \alpha^{k_{110}}(1-\alpha)^{k_{000}+k_{010}+k_{011}}(1-\beta)^{k_{100}+k_{111}}$$

Using similar methods, the other likelihoods are as follows.

$$\mathcal{L}(r \cap U = \emptyset; \hat{r}, U, S) = \alpha^{k_{110}+k_{111}}(1-\alpha)^{k_{000}+k_{010}+k_{011}}(1-\beta)^{k_{100}}$$
$$\mathcal{L}(r \cap U \setminus S \neq \emptyset, r \cap S = \emptyset; \hat{r}, U, S) = \alpha^{k_{111}}(1-\alpha)^{k_{000}+k_{010}+k_{011}}(1-\beta)^{k_{100}+k_{110}}$$

These can be compared to get the conditions in the statement.

■

4.1.3. Columns in Subtree of Noiseless Pivot: The function COLUMNSINSUBTREEOFNOISELESSPIVOT finds the subtree of a column. This is similar to COLUMNSINSUBTREEOFNOISYPIVOT, with the key difference being that this function assumes the input column is correct. Therefore, COLUMNSINSUBTREEOFNOISELESSPIVOT determines whether columns are likely to be subsets of the lead edge, instead of simply likely to have non-empty intersection. This function is used in finding the subtree of a reconstructed column. Lemma 4 shows the optimal decision rule for determining whether a noisy column is in the subtree of a column when the column is correct.

LEMMA 4. *Let $\hat{x}$ be a noisy $\{0,1\}$-column, let x be its unknown true counterpart, let y be a $\{0,1\}$-column of the same length, and let α and β be the false positive and false negative probabilities respectively. Then, $x \subseteq y$ is more likely than $x \cap y = \emptyset$ if $|\hat{x} \cap y| \geq |\hat{x} \cap y^c|$.*

Proof This proof follows similarly to the proof of Lemma 2. The likelihood that $x \subseteq y$ is

$$\mathcal{L}(x \subseteq y; \hat{x}, y) = \max_{x: x \subseteq y} \Pi_i \Pr\{\hat{x}(i)|x(i)\}.$$

For a particular i, $\Pr\{\hat{x}(i)|x(i)\}$ is maximized when $x(i) = \hat{x}(i)$, so the maximizer x is such that $x(i) = \hat{x}(i)$ if $\hat{x}(i) \leq y(i)$. When $(\hat{x}(i), y(i)) = (1,0)$, then $x(i)$ must be 0 since $y(i)$ is known and assumed correct. Let $k_{00} = |\hat{x}^c \cap \hat{y}^c|, k_{01} = |\hat{x}^c \cap \hat{y}|$, and similar for k_{10} and k_{11}. Then the likelihood becomes the following.

$$\begin{aligned}\mathcal{L}(x \subseteq y; \hat{x}, y) &= \Pr\{0|0\}^{k_{00}} \Pr\{0|0\}^{k_{01}} \Pr\{1|0\}^{k_{10}} \Pr\{1|1\}^{k_{11}} \\ &= \alpha^{k_{10}}(1-\alpha)^{k_{00}+k_{01}}(1-\beta)^{k_{11}}\end{aligned}$$

The likelihood $\mathcal{L}(x \cap y = \emptyset; \hat{x}, y)$ can similarly be computed and then compared against $\mathcal{L}(x \subseteq y; \hat{x}, y)$.

■

4.1.4. Root Node-based Reconstruction: Initially, the procedure ROOTNODEBASEDRECONSTRUCTION counts the number of ones in each column within subtree S. Due to the inherent structure of subtrees where each node must be a superset of its root node, the indices corresponding to the highest values in this sum align with the non-zero elements within the root node. When it comes to handling noise, the method employs the expected number of 1's as the threshold. This choice serves as the most reliable and unbiased estimate available.

4.2. Algorithm Complexity

The worst-case time complexities for each sub-procedure of Sempervirens are $O(n|P|^2)$ for LARGESTMAXIMALSUBTREE and $O(n|P|)$ for COLUMNSINSUBTREEOFNOISYPIVOT, ROOTNODEBASEDRECONSTRUCTION, RECONSTRUCTSUBTREELEADEDGE, COLUMNSINSUBTREEOFNOISELESSPIVOT, BACKWARDSPASS, and SELECTCOLUMNTOREPLACE. The time complexity of MAXIMUMLIKELIHOODREFINEMENT is $O(nm^2)$. Thus, the worst-case time of Sempervirens complexity is $O(nm^3)$.

This worst-case time-complexity of $O(nm^3)$ is achieved when the input data has the star topology, where every pair of columns are disjoint, and no row is a superset of another row. With this topology, each call to RECONSTRUCT divides P such that $S = \{u\}$, so the recursion continues only on $P \setminus \{u\}$. This results in m calls to RECONSTRUCT with the i'th call from the end having $|P| = i$. Now consider a call to reconstruct given the star topology with $|P| = i$. The function FINDLARGESTMAXIMALSUBTREE determines the time complexity in this case. In each iteration of this function, S_0 consists of only u_0, so there are i iterations. Further, the k'th iteration from the end involves the multiplication of a $k \times n$ matrix by a n-length vector. Thus, the complexity of an iteration in FINDLARGESTMAXIMALSUBTREE is $O(nk)$, and so the complexity of FINDLARGESTMAXIMALSUBTREE is $O(ni^2)$. Since RECONSTRUCT is called in a loop with P such that $i = 1, \ldots, m$, the time complexity of RECONSTRUCT is $O(nm^3)$. Therefore, with the star topology, the time complexity of Sempervirens is $O(nm^3)$.

However, the time complexity of Sempervirens depends heavily on the topology of the data. In practical scenarios, extreme cases such as the star topology are rare occurrences. Even the inclusion of substantial subtrees connected to the root can significantly reduce the time complexity. For example, if b subtrees are connected to the root with $b \ll m$, and each subtree has significantly less than n nodes and m edges, then even if each subtree has the worst-case time complexity, the overall time complexity will be drastically reduced compared to the worst-case.

The best-case topology is a tree that regularly branches into subtrees with equal numbers of edges. For example, take the case where every subtree has two subtrees of equal numbers of edges. Then at each call to RECONSTRUCT, the procedure FINDLARGESTMAXIMALSUBTREE will have time-complexity $O(n|P|)$, and each recursion will be of size roughly $\frac{1}{2}|P|$. Hence, at depth k, there are 2^k recursion calls on $|P| \approx \frac{1}{2^k}m$. Therefore, the overall time-complexity in this case is $O(nm\log(m))$.

5. Experiments

In this section we present experimental results comparing our algorithm to others. We begin by introducing our implementation and the accuracy measures we consider for comparing the quality of different reconstructions. We then provide experimental results on simulated datasets of various sizes. Finally, we present additional experimental results on the robustness of our algorithm compared to others.

5.1. Implementation

An implementation of Sempervirens is provided at `https://github.com/nevenag/sempervirens`. A reference version as used in this paper is also provided in (Junnarkar et al. 2023). This implementation is written in Python with the only dependency being NumPy, making it very easy to setup. The inputs to the reconstruction function are just the noisy matrix, the false positive probability, the false negative probability, and the missing entry probability. There are no parameters to be tuned. The output is a conflict-free reconstruction of the noisy matrix. The implementation can either be used directly in other code as a library, or from the command line. We also provide a faster compiled implementation of Sempervirens, written in Rust. We use the Python implementation for all runtimes in our experiments, unless otherwise specified.

5.2. Accuracy Measures

We consider two primary measures for the accuracy of the reconstruction: the ancestor-descendant score and the different-lineages score. We also consider a third measure, the fraction of correct elements, but display these scores only in the Appendix (in the online supplement) due to the results for the best performing algorithms (Sempervirens, HUNTRESS, and ScisTree) being very similar to each other. For the following definitions, let $\tilde{M}$ be the ground-truth matrix and M be the reconstructed matrix.

1. Ancestor-descendant (AD) score: the AD score considers the set of pairs of columns (a, b) such that $a \supset b$ (strictly) in $\tilde{M}$, and defines the score as the fraction of those pairs which maintain the same relation in M (Malikic et al. 2019). The AD score lies in the interval $[0, 1]$ with higher scores being better.

2. Different-lineage (DL) score: the DL score considers the set of pairs of columns (a, b) such that $a \cap b = \emptyset$ in $\tilde{M}$, and defines the score as the fraction of these pairs which maintain the same relation in M (Malikic et al. 2019). As with the AD score, the DL score lies in $[0, 1]$ with higher scores being better.

3. Fraction of correct elements (FC) score: this score is defined as the number of elements in the matrix M that match the corresponding element in $\tilde{M}$, divided by the number of elements in the matrix. Again, this score lies in [0, 1] with higher scores being better.

5.3. Comparisons

We compare our proposed algorithm, Sempervirens, to HUNTRESS (Kızılkale et al. 2022), ScisTree (Wu 2020), and SCITE (Jahn, Kuipers, and Beerenwinkel 2016). These methods are selected since they are among the fastest algorithms while being accurate, as shown in (Kızılkale et al. 2022) and (Wu 2020). All methods we compare against take as input at least false positive and false negative rates. The method SPhyR (El-Kebir 2018) is excluded due to poor performance on ancestor-descendant scores shown in (Kızılkale et al. 2022). We use a modified implementation of ScisTree, labeled ScisTreeP, that leverages parallelization to reduce computation time. We run SCITE with 90,000 iterations and 3 restarts. All parallelized methods are run with all available threads. All methods are run on an Intel Core i7-1360P (a 2023-model laptop

CPU) with 4 P-cores and 8 E-cores, for a total of 16 threads. The system Basic Linear Algebra Subprograms (BLAS) installation is the Intel Math Kernel Library (Intel MKL). This is used indirectly by Sempervirens and HUNTRESS through NumPy.

Note that the implementations of these methods we compare against have been optimized to varying levels, so the computation time of some methods might be improved. For example, HUNTRESS not only benefits from additional cores (in (Kızılkale et al. 2022) it is run on a 32-core machine, hence the lower runtime they report), but also is written in Python with room for additional vectorization. ScisTreeP and SCITE on the other hand are written in C++.

The methods are tested on datasets of matrices of sizes 300×300 ($n \times m$ is n objects described by m characters), 1000×300, 300×1000, and 1000×1000, with various false positive probabilities, false negative probabilities, and missing entry probabilities. To make cross-comparisons possible, we start with the same ground truth matrices as used in (Kızılkale et al. 2022). Then, given a particular false positive probability α, false negative probability β, and missing entry probability γ, we add noise to each matrix by changing 0s to 1s with probability α, 0s to missing with probability γ, 1s to 0s with probability β, and 1s to missing with probability γ. We create six noisy datasets from the conflict-free 300×300 matrices using $(\alpha, \beta) \in \{0.001, 0.003, 0.01\} \times \{0.05, 0.2\}$ and $\gamma = 0.05$. False positive probabilities on the order of 0.001 and false negative probabilities of 0.2 have previously been used in (Kızılkale et al. 2022), (Wu 2020), and (El-Kebir 2018). To check robustness of these algorithms, we have also added datasets with ten times higher false positive probabilities of 0.01, which is unrealistically high in practice. A similar process is used to create the datasets for the other sizes of matrices.

We use Sempervirens, HUNTRESS, ScisTreeP, and SCITE to reconstruct each matrix in each dataset and compute the ancestor-descendant and different-lineage scores of the reconstructions. Figure 4 summarizes the ancestor-descendant scores of each method on each dataset. Sempervirens, HUNTRESS, and ScisTreeP show comparable scores in all datasets, while SCITE performs poorly. On datasets with low false positive probabilities, Sempervirens tends to show the highest average ancestor-descendant score. Additionally, on small matrices (the 300×300s datasets), Sempervirens typically displays the highest worst-case score. Figure 5 summarizes the different-lineage scores of each method on each dataset. As with the ancestor-descendant scores, we find that Sempervirens, HUNTRESS, and ScisTreeP all achieve comparable different-lineage scores, while SCITE performs poorly. The different-lineage scores of Sempervirens, HUNTRESS, and ScisTreeP are all near optimal on all datasets. At the highest false positive probabilities, HUNTRESS edges out Sempervirens and ScisTreeP in different-lineage scores. Strangely, we observe that SCITE's scores improve with higher false positive probabilities.

Figure 6 summarizes the runtimes of each method on each dataset. Sempervirens is by far the fastest method, typically running two orders of magnitude faster than the next fastest method in each dataset. Indeed, across all datasets, the slowest reconstruction time for Sempervirens is 7.191 seconds on a $1000 \times$

1000 matrix, while the fastest reconstruction time for HUNTRESS is 58.698 seconds on a 300×300 matrix, for ScisTreeP is 15.435 seconds on a 300×300 matrix, and for SCITE is 87.016 seconds on a 300×300 matrix. We also note the runtime of Sempervirens shows no increase with increasing false positive or false negative probabilities. This contrasts with ScisTreeP which shows runtime increasing with false positive probabilities and especially dependent on false negative probabilities.

Tables 2, 3, 4, and 5 in the Appendix (in the online supplement) detail the runtimes, ancestor-descendant scores, different-lineage scores, and fractions-correct scores of tested methods on the datasets of matrices of sizes 300×300, 1000×300 and 300×1000, and 1000×1000 respectively.

The high execution speed of Sempervirens enables it to be applied to large matrices while still completing relatively quickly. An advantage of large matrices is that there is more information, enabling higher accuracy reconstructions. For example, rows of the matrix correspond to samples from the phylogenetic tree, and more samples enables higher reconstruction accuracy even if the samples are noisy or incomplete. To emphasize the capabilities of Sempervirens, we run it on datasets of large tall matrices (20000×2000) and large fat matrices (2000×20000). Each dataset contains 10 matrices. Due to the size of these matrices, we use the faster Rust implementation of Sempervirens. Table 1 summarizes the runtimes and reconstruction accuracy scores on these datasets. Sempervirens reconstructs the matrices in these datasets with near-optimal ancestor-descendant and different-lineage scores. Further, it reconstructs each 20000×2000 matrix in under 6 minutes and each 2000×20000 matrix in under 45 minutes. Using other available methods, reconstructing each of these matrices would take many days. Sempervirens makes reconstruction of matrices too large for other methods not only computationally feasible, but feasible on a regular laptop.

Table 1 Sempervirens on Large Matrices. Note $\gamma = 0.05$.

Dataset				Method	Runtime (s)		AD		DL	
n	m	α	β		Min	Max	Mean	Std	Mean	Std
20000	2000	0.001	0.05	Sempervirens (Rust)	214.419	313.275	1.000	0.000	1.000	0.000
		0.001	0.2	Sempervirens (Rust)	214.695	313.408	1.000	0.000	1.000	0.000
2000	20000	0.001	0.05	Sempervirens (Rust)	609.015	2436.962	0.998	0.002	1.000	0.000
		0.001	0.2	Sempervirens (Rust)	701.046	1681.370	0.999	0.001	1.000	0.000

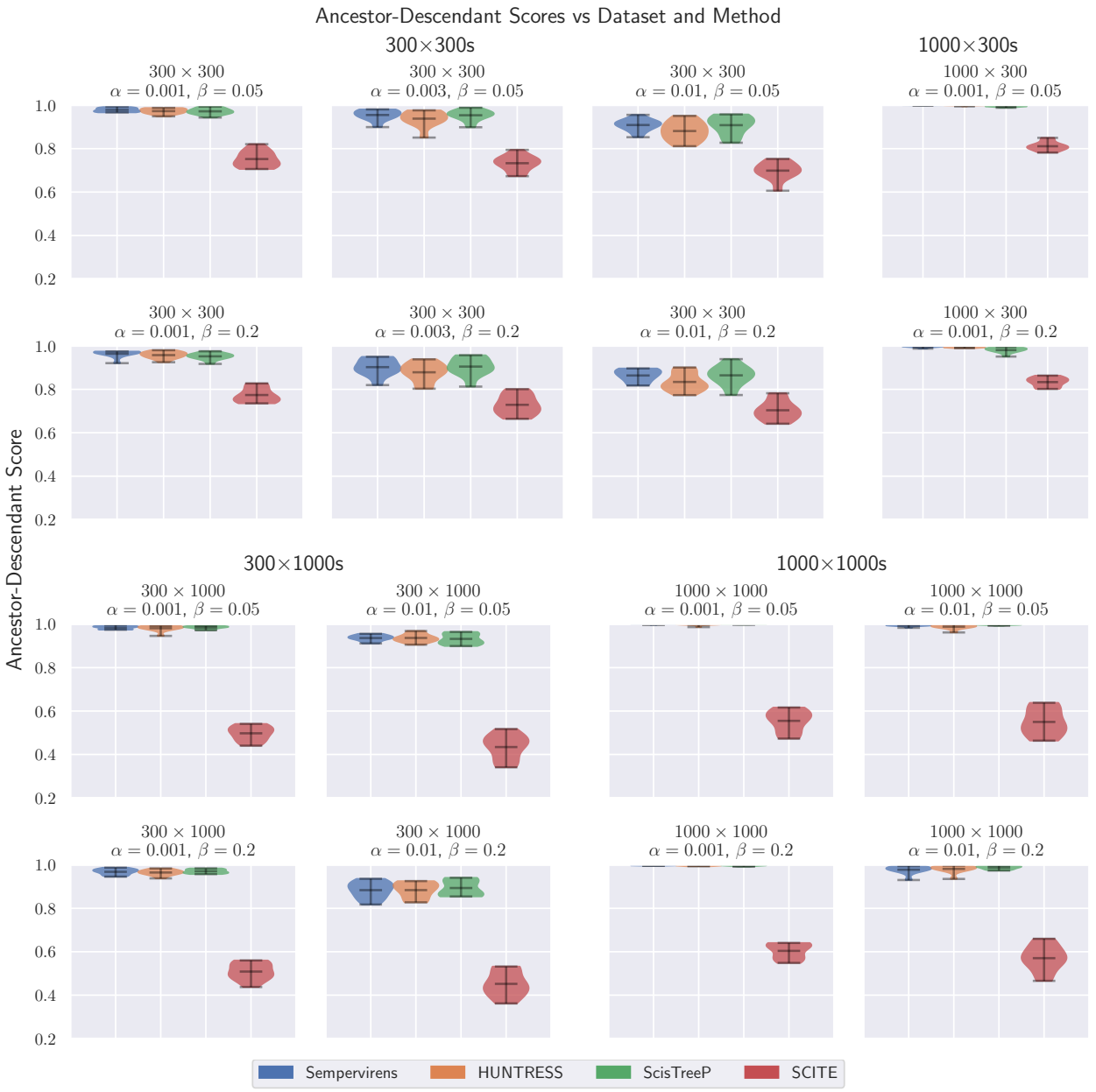

Figure 4 **Ancestor-descendant scores of reconstructions. Each of the 16 charts shows the reconstruction accuracy of each method on a single dataset. Each dataset consists of 10 matrices of size $n \times m$ where n is the number of rows and m is the number of columns. Noise is added to the matrices with false positive probability α, false negative probability β, and missing entry probability $\gamma = 0.05$. The violin plots indicate the minimum, mean, and maximum values with bars. Algorithms are positioned left-to-right in plots as in legend.**

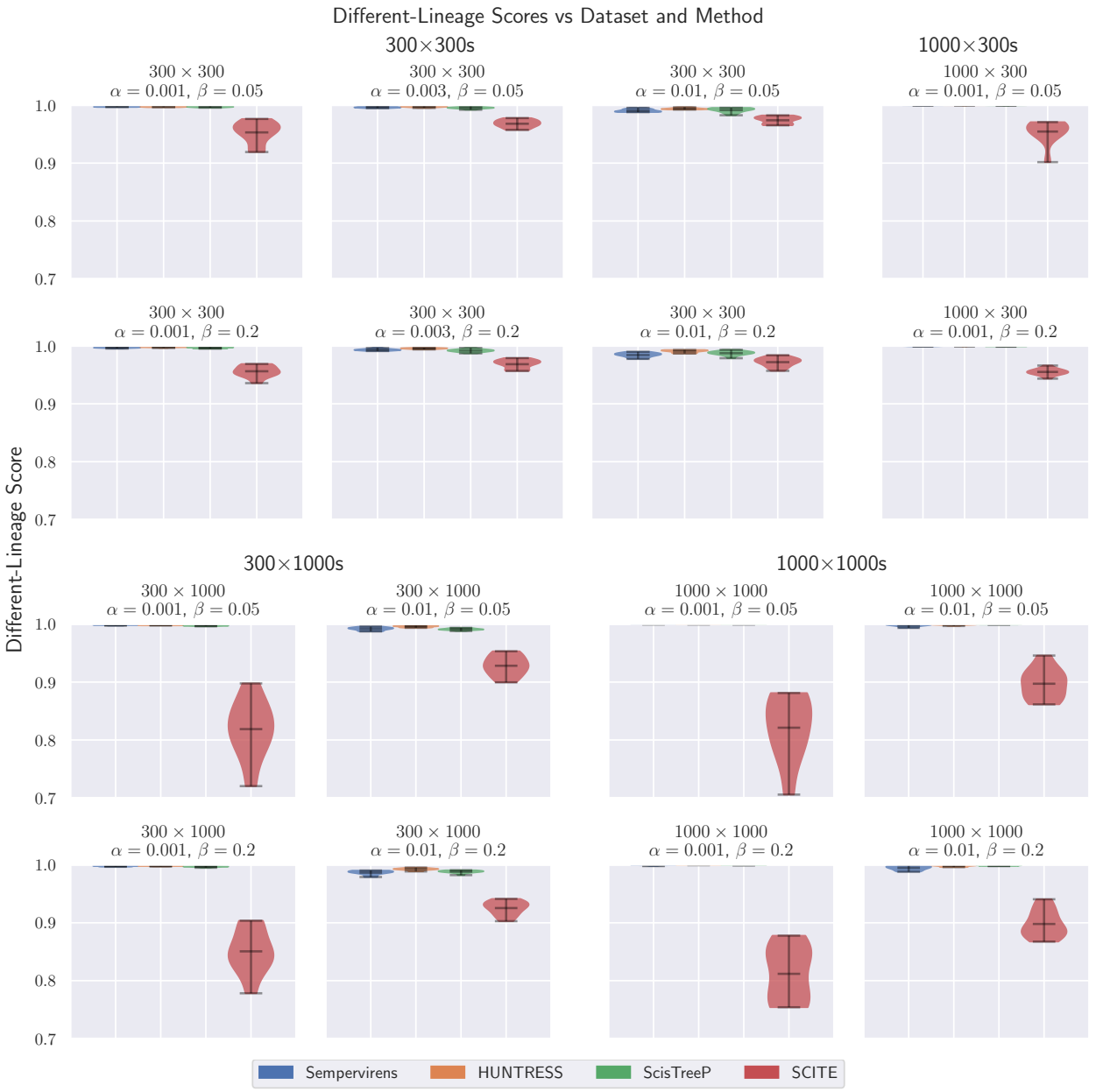

Figure 5 **Different-lineage scores of reconstructions. See Figure 4 for an explanation of the violinplots. Algorithms are positioned left-to-right in plots as in legend.**

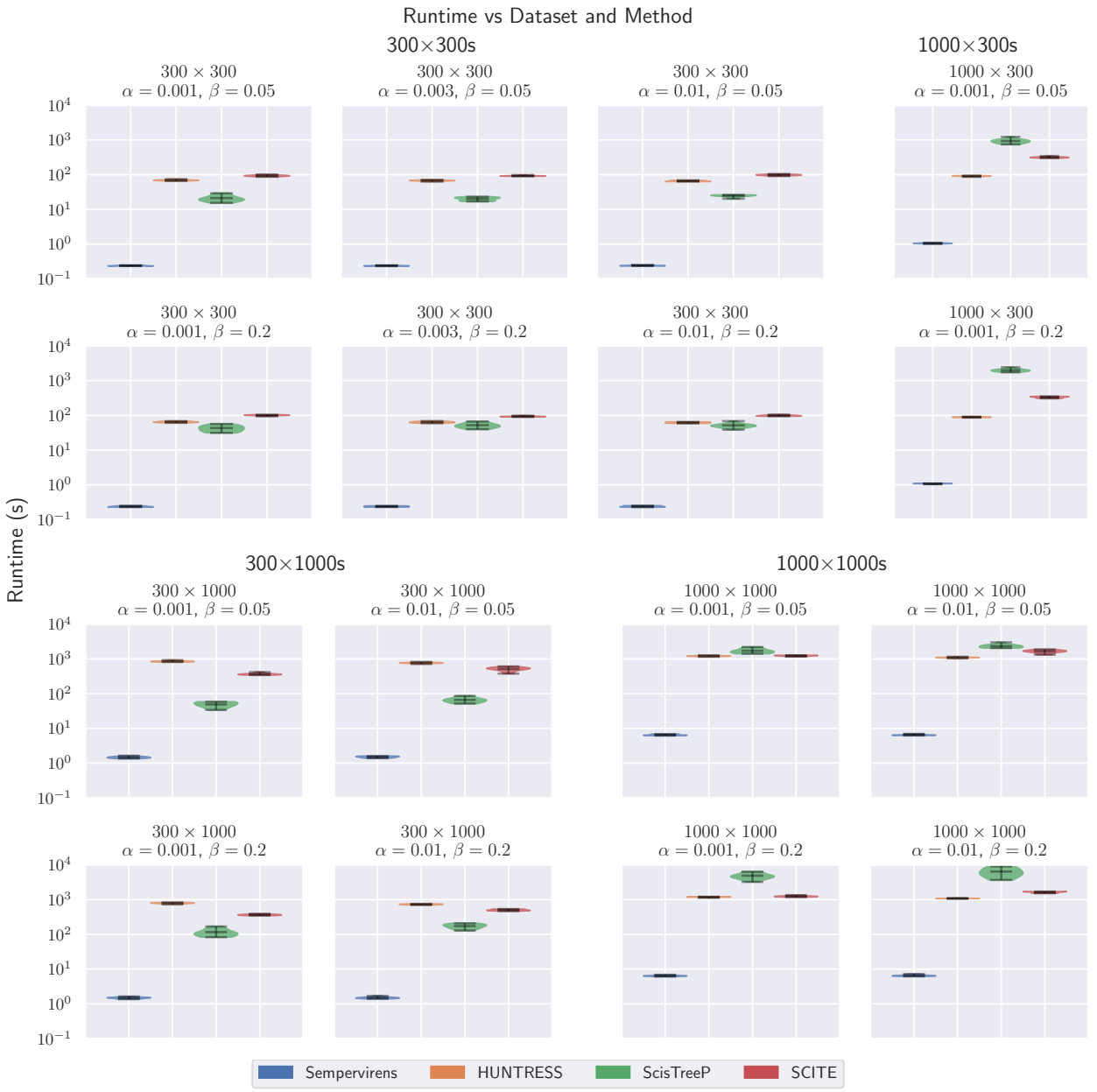

Figure 6 **Runtimes of each reconstruction. Note the y-axis is in log scale. See Figure 4 for an explanation of the violin plots. Algorithms are positioned left-to-right in plots as in legend.**

5.4. Robustness

In this section, we examine the robustness of our algorithm to an increased missing entry probability, to ambiguity in the false positive probability, and to ambiguity in the false negative probability. Details of construction of the datasets for these experiments and the experimental data can be found in the appendix (in the online supplement): for experimental data with higher missing entry probabilities, see Appendix B; for robustness to ambiguity in false positive and false negative probabilities, see Appendices C and D respectively. We find our algorithm, Sempervirens, and HUNTRESS to be the most robust.

To evaluate the impact of higher missing entry probabilities, we test Sempervirens, HUNTRESS, ScisTree, and SCITE with missing entry probabilities of 0.1 and 0.15 (compared to the probability of 0.05 used in Section 5.3). See Tables 6 and 7 in Appendix B. Sempervirens, HUNTRESS, and ScisTree all maintain high accuracy without any significant decrease compared to scores on the datasets with missing entry probability of 0.05. SCITE continues to under-perform compared to the other algorithms.

To evaluate the impact of ambiguity in the false positive probability, we run the algorithms assuming false positive probabilities ten times higher and ten times lower than the actual probability. See Table 8 in Appendix C. Sempervirens, HUNTRESS, and ScisTree all maintain high accuracy, with changes in score (with respect to the scores when run with the correct probabilities) of less than 1%. SCITE's scores change as much as 50%, both in the positive and negative direction.

Finally, to evaluate the impact of ambiguity in the false negative probability, we run the algorithms assuming false negative probabilities ten times higher (when possible: when the real probability is 0.2, ten times higher would mean a probability of 2; hence we have used 0.4 instead) and ten times lower than the actual probability. See Table 9 in Appendix D. While Sempervirens and HUNTRESS maintain high accuracy, ScisTree was highly inaccurate for 2 out of the 8 datasets. SCITE failed to run on the datasets with very low assumed false negative probabilities, but otherwise its scores changed little compared to its scores when run with the correct false negative probability (unfortunately, SCITE scores poorly with the correct false negative probability).

The robustness of Sempervirens against ambiguity in false positive and false negative probabilities primarily arises from the fact that the pseudo log-likelihood computations are not highly sensitive to variations in α and β. As demonstrated in Section 4.1, the comparison tests between two vectors are largely influenced by the number of matching and differing elements—in fact, all but two tests are independent of α and β. The tests that are influenced by α and β depend linearly on the terms $\log(1-\beta)$, $\log(1-\alpha)$, $\log\beta$, and $\log\alpha$. Notably, these logarithmic terms change slowly with respect to variations in α and β: a variation of $\alpha \to \alpha^c$ is required to change $\log\alpha \to c\log\alpha$. It is also important to note that simultaneous ambiguity in α and β has a cumulative effect in the worst case since their effect is log linear. Like Sempervirens, HUNTRESS depends mostly on counts, except in the refinement stage (which is similar to the maximum likelihood refinement stage of Sempervirens), so it was expected to be resilient to ambiguity in α and β.

ScisTree is mostly resilient to ambiguity when α and β are small. However, we observe that as β gets close to 0.5, ScisTree performs poorly, likely due its local-search nature.

6. Conclusion And Future Research

In this work, we have introduced Sempervirens, an algorithm of exceptional speed built for reconstructing conflict-free matrices from noisy and incomplete data. Not only have we established the optimality of each subprocedure, but our experiments on simulated data have shown that Sempervirens achieves accuracy levels on par with, or even surpassing, those achieved by other state-of-the-art methods. Moreover, Sempervirens achieves this accuracy while running orders of magnitude faster than these methods. This makes Sempervirens suitable to run on much larger datasets, which contain more information and enable high accuracy reconstructions. Furthermore, our experiments demonstrate that the algorithm remains robust despite uncertain false positive and false negative probabilities. The only limitation of the algorithm appears when the matrix is small, specifically with just a few tens of rows and columns. For small matrices, brute-force search methods such as branch and bound can be employed.

Directions for future research involve expanding the scope of Sempervirens to handle correlated noise distributions and establishing error rate bounds for the algorithm's overall output. Additionally, although providing reconstruction accuracy guarantees for arbitrary matrices may not be feasible, identifying tree topologies where such guarantees are possible presents another avenue for exploration.

7. Acknowledgements

This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript or allow others to do so, for U.S. Government purposes.

We sincerely appreciate the anonymous reviewers for their constructive feedback, which helped us refine the presentation of our work.

References

Gusfield, Dan (1991). "Efficient algorithms for inferring evolutionary trees". In: *Networks* 21.1, pp. 19–28. ISSN: 1097-0037. DOI: 10.1002/net.3230210104. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230210104.

Jahn, Katharina, Jack Kuipers, and Niko Beerenwinkel (May 5, 2016). "Tree inference for single-cell data". In: *Genome Biology* 17.1, p. 86. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0936-x. URL: https://doi.org/10.1186/s13059-016-0936-x.

Junnarkar, Neelay et al. (2023). *Sempervirens: A Fast Reconstruction Algorithm for Noisy and Incomplete Binary Matrix Representations of Phylogenetic Trees*. Available for download at https://github.com/INFORMSJoC/2023.0373. DOI: 10.1287/ijoc.2023.0373.cd. URL: https://github.com/INFORMSJoC/2023.0373.

El-Kebir, Mohammed (Sept. 1, 2018). "SPhyR: tumor phylogeny estimation from single-cell sequencing data under loss and error". In: *Bioinformatics* 34.17, pp. i671–i679. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty589. URL: https://doi.org/10.1093/bioinformatics/bty589.

Kızılkale, Can et al. (Sept. 2022). "Fast intratumor heterogeneity inference from single-cell sequencing data". In: *Nature Computational Science* 2.9, pp. 577–583. ISSN: 2662-8457. DOI: 10.1038/s43588-022-00298-x. URL: https://www.nature.com/articles/s43588-022-00298-x.

Malikic, Salem et al. (June 21, 2019). "Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data". In: *Nature Communications* 10.1, p. 2750. ISSN: 2041-1723. DOI: 10.1038/s41467-019-10737-5. URL: https://www.nature.com/articles/s41467-019-10737-5.

Meacham, Christopher A. (1983). "Theoretical and Computational Considerations of the Compatibility of Qualitative Taxonomic Characters". In: *Numerical Taxonomy*. Ed. by Joseph Felsenstein. NATO ASI Series. Berlin, Heidelberg: Springer, pp. 304–314. ISBN: 978-3-642-69024-2. DOI: 10.1007/978-3-642-69024-2_34. URL: https://doi.org/10.1007/978-3-642-69024-2_34.

Navin, Nicholas E. (Aug. 30, 2014). "Cancer genomics: one cell at a time". In: *Genome Biology* 15.8, p. 452. ISSN: 1474-760X. DOI: 10.1186/s13059-014-0452-9. URL: https://doi.org/10.1186/s13059-014-0452-9.

Sadeqi Azer, Erfan et al. (July 2020). "PhISCS-BnB: a fast branch and bound algorithm for the perfect tumor phylogeny reconstruction problem". In: *Bioinformatics* 36.Supplement$_1$, pp. i169–i176. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa464. URL: https://doi.org/10.1093/bioinformatics/btaa464.

Wu, Yufeng (Feb. 1, 2020). "Accurate and efficient cell lineage tree inference from noisy single cell data: the maximum likelihood perfect phylogeny approach". In: *Bioinformatics* 36.3, pp. 742–750. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz676. URL: https://doi.org/10.1093/bioinformatics/btz676.