2012

Enumerating All Maximal Frequent Subtrees

Akshay Deepak
Iowa State University, akshaydeepak@gmail.com

David Fernández-Baca
Iowa State University, fernande@iastate.edu

Enumerating All Maximal Frequent Subtrees

Technical Report #12-01, Dept. of Computer Science, Iowa State University

Akshay Deepak[1] and David Fernández-Baca[1]

Department of Computer Science, Iowa State University, Ames, Iowa, USA

Abstract. Given a collection of leaf-labeled trees on a common leafset and a fraction $f \in \left(\frac{1}{2}, 1\right]$, a frequent subtree (FST) is a subtree isomorphically included in at least fraction f of the input trees. The well-known maximum agreement subtree (MAST) problem identifies FST with $f = 1$ and having the largest number of leaves. Apart from its intrinsic interest from the algorithmic perspective, MAST has practical applications as a metric for tree similarity, for computing tree congruence, in detection horizontal gene transfer events and as a consensus approach. Enumerating FSTs extend the MAST problem by definition and reveal additional subtrees not displayed by MAST. This can happen in two ways – such a subtree is included in majority but not all of the input trees or such a subtree though included in all the input trees, does not have the maximum number of leaves. Further, FSTs can be enumerated on collections of trees having partially overlapping leafsets. MAST may not be useful here especially if the common overlap among leafsets is very low. Though very useful, the number of FSTs suffer from combinatorial explosion – just a single MAST can exhibit exponentially many FSTs. This limits both the size of the trees that can be enumerated and the ability to comprehend enumerated FSTs. To overcome this, we propose enumeration of maximal frequent subtrees (MFSTs). A MFST is a FST that is not a subtree to any other FST. The set of MFSTs is a compact non-redundant summary of all FSTs and is much smaller in size. Here we tackle the novel problem of enumerating all MFSTs in collections of phylogenetic trees. We demonstrate its utility in returning larger consensus trees in comparison to MAST. The current implementation is available on the web.

1 Introduction

MAST [11] is a commonly used approach to extract common information in a collection of phylogenetic trees having identical leafset. This has many practical applications such as a metric for comparing phylogenetic trees [12,8,10], computing their congruence index [7,16], identifying horizontal gene transfer events [6], for resolving ambiguity in terraces in phylogenetic tree space [18] and as a consensus approach – where a MAST can be significantly more resolved than the majority rule tree (MRT), which tends to suffer from the '*rogue taxon*' effect [21,13]. The MAST problem is polynomially-solvable for two trees, but is NP-hard for three or more input trees having unbounded degrees [1].

Our motivation to enumerate all MFSTs is that it naturally extends the MAST problem and provides additional phylogenetic information. Specifically the set of MFSTs can contain subtrees that are more informative in following ways:

1. A MFST that has more number of leaves than a MAST but is not supported by all the input trees. Figure 1 shows an example for this case.
2. A MFST that has less number of leaves than a MAST but is not displayed by any of the MASTs. Such a subtree can either be be supported by all the input trees or by the majority fraction. Figure 2 shows an example for this case.
3. MFSTs can be enumerated for collections of trees having partially overlapping leafsets. This cannot be applied to MASTs especially in the case where the common overlap among all the input trees is very low. Figure 3 shows an example of this case.

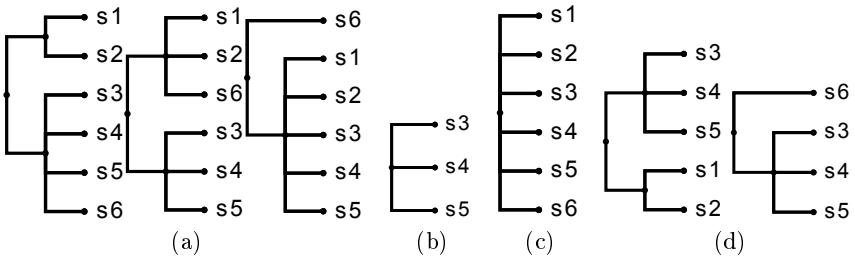


Fig. 1. Input trees are shown in 1a. 1b shows the MAST, which is uninformative (star-like). 1d shows two MFSTs – each is supported by two input trees. Each MFST is resolved and is larger than the MAST. This also illustrates the utility of MXSTs as consensus trees as the corresponding MRT (shown in 1c) is also uninformative.

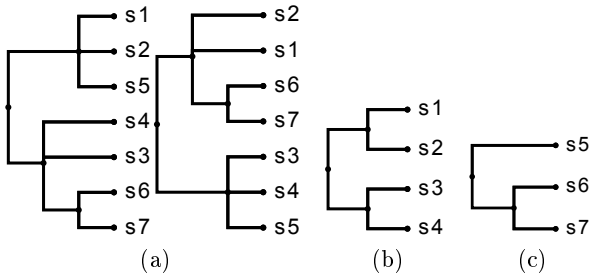
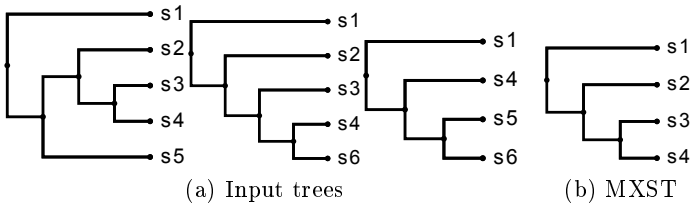


Fig. 2. 2a shows the input trees. 2b shows the MAST. 2c shows an MXST, which is smaller than the MAST but is not displayed by it.



(a) Input trees (b) MXST

Fig. 3. 3a shows the input trees with partially overlapping leafsets. 3b shows a MXST. MAST or MRT cannot be applied effectively as the common overlap consists of only two leaves.

The set of all FSTs can be quite large making it difficult to compute and make sense of. The set of all MFSTs is a compact summary of the set of all FSTs and is much smaller in size. The set of all MFSTs has the special property that every FST is a subtree to some MFST but every MFST is not a subtree to any other FST. Thus, every MFST reveals some unique phylogenetic information that is not displayed by any other MFST. To our knowledge the problem of enumerating all MFSTs has not been dealt before. Here we describe a new algorithm MfstMiner for this task. Experiments indicate a polynomial behavior with respect to the total number of MFSTs. We compare MfstMiner with Phylominer [25] – a polynomial time algorithm for enumerating all FSTs. We demonstrate the ability of MFSTs in producing larger consensus trees in comparison to the MAST. The current implementation of MfstMiner can be downloaded from `http://www.cs.iastate.edu/~akshayd/mfstMiner/`.

1.1 Related Work

Due its utility and inherent complexity, the MAST problem has been of particular interest to computational biologists and mathematicians alike. It was first studied by Finden and Gordon [11]. It is polynomially solvable for two trees and over the years its complexity has progressively improved [1,20,15]. However, for more than two trees with unbounded degrees it becomes NP-hard [1]. For trees with bounded degree, it again becomes solvable in polynomial time [9,4].

In data mining literature, maximal subtree mining [23,24,5] and maximal subgraph mining [14,22] have received prominent attention. However, these cannot be applied here as phylogenetic trees possess a very special structure – only leaves are labeled and non-leaf nodes must be of degree two or more. Thus, it demands a separate approach. The problem of mining frequent phylogenetic subtrees (subtrees included in majority of the input trees) has been studied by Zhang et al. in [25] where they proposed a polynomial time algorithm (Phylominer) to mine all frequent subtrees in a collection of phylogenetic trees. By definition the set of all frequent subtrees include the set of all MFSTs, however the number of frequent subtrees can be exponentially more than the number of MFSTs. Thus, mining MFSTs exclusively saves a lot of time and the result set is much smaller to analyze. To our knowledge the problem of mining MFSTs for

evolutionary trees has not been studied before. Hence, our work is the first one to deal with it.

2 Preliminaries

A ***phylogenetic tree*** is an unordered rooted or unrooted leaf-labeled tree. Leaf labels represent the taxonomic units (species) under study. A node is *internal* if it is not a leaf node. If a phylogeny T is rooted, we require that each internal node have at least two children; if T is unrooted, every internal node is required to have degree at least three. Our current work deals with rooted phylogenetic trees. Consider a phylogenetic tree T. Let $\mathcal{L}_T$ denote the set of labels of the leaves of T, and Φ_T denote the bijection that maps the leaf nodes to their unique labels. For convenience, we refer to the set of leaf nodes by their labels in $\mathcal{L}_T$. For brevity, we will often refer to a phylogenetic tree simply as a *tree*.

From this point onwards, unless the context requires making a distinction, we will drop the subscripts in $\mathcal{L}_T$ and Φ_T, and write $\mathcal{L}$ and Φ respectively. We assume that the reader is familiar with the usual notions of ancestor/descendant and parent/child for rooted trees. The *depth* of a node u, denoted depth(u), is the number of edges from the root to that node; thus the root node is at depth 0. We denote the lowest common ancestor (LCA) of two nodes u and v by LCA(u, v). A *k-leaf tree* is a tree with k leaves.

In a graph (not necessarily a phylogenetic tree), suppose u is a degree-two node with neighbors v and w. Then, *suppressing* u means deleting u and replacing edges (v, u) and (u, w) by a single edge (v, w). Consider a tree T and a subset of its leaves $\mathcal{L}'$. The *restriction* of T to $\mathcal{L}'$, denoted by $T|_{\mathcal{L}'}$, is the tree on the leaf set $\mathcal{L}'$ obtained from the minimal subgraph T' of T spanning $\mathcal{L}'$ by suppressing any degree-two node except the root [19]. The root of this restricted tree is the node closest to the root of T. A tree T' is called a ***subtree*** of another tree T, denoted as $T' \equiv T|_{\mathcal{L}_{T'}}$, if $\mathcal{L}_{T'} \subseteq \mathcal{L}_T$ and T' is isomorphic to $T|_{\mathcal{L}_{T'}}$. In such a case, T *displays* T'.

Given a collection of trees on a common leafset, the *support* or *frequency* of a subtree is the fraction of the input trees in which it is included. For a fraction $f \in \left(\frac{1}{2}, 1\right]$, a ***frequent subtree*** (FST) is a subtree with support greater than or equal to f. A ***maximal frequent subtree*** (MFST) is a FST subtree that is not a subtree of any other FST. Our goal is to enumerate all MFSTs in a collection of trees on a common leafset.

Next we give the counterpart of the above definitions for the specific case of $f = 1$. An ***agreement subtree*** (AST) is a FST with $f = 1$. A ***maximum agreement subtree*** (MAST) is a FST with $f = 1$ and having the largest number of leaves. Clearly the set of all MASTs is a subset of the set of all MFSTs. A ***maximal agreement subtree*** (MXST) is a MFST with $f = 1$. Though the set of all MXSTs is a subset of the set of all MFSTs, its enumeration allows certain simplifications over the enumeration of all MFSTs for $f \in \left(\frac{1}{2}, 1\right)$ and is more efficient. Thus, it is implemented separately in our current tool.

3 Algorithmic Framework

We first discuss our algorithmic framework in the context of ASTs and MXSTs as its enumeration is simpler to deal with. We then extend it for FSTs and MFSTs. Consider the solution space of all ASTs. The set of all MXSTs is a subset of this solution space. Our approach efficiently mines all MXSTs from the solution space of all ASTs. It uses the anti-monotone property of enumerating frequent patters - the frequency of a super-pattern is always less than the frequency of a sub-pattern. For the case of ASTs, this means that a k-leaf subtree would be an AST only if all k of its $(k-1)$-leaf subtrees are ASTs. This gives an efficient way to enumerate larger ASTs - by trying to join smaller ASTs that can result in this larger AST. In fact, we show that every k-leaf AST can be enumerated by combining two unique $(k-1)$-leaf ASTs. Our goal is to enumerate MXSTs. Thus, we enumerate only those ASTs that can potentially lead to MXSTs. To do so efficiently there are three main issues that need to be addressed: non-redundant enumeration of each MXST, avoiding the combinatorial explosion due to the number of ASTs, and efficient frequency counting of a subtree to classify it as an AST.

Redundant enumeration can occur due to two reasons: enumeration of different isomorphic representations of the same MXST and multiple enumerations of the same ordered representation of a MXST. To deal with isomorphism, we enumerate subtrees in an ordered representation or in their 'canonical form'. To enumerate every canonical representation once, we define a parent-child relationship over the solution space of all ASTs. This actually induces an enumeration tree over the solution space. Each node represents a collection of ASTs grouped together via an equivalence relation. Leaf nodes represent potential MXSTs and each MXST belongs to a unique leaf node. This scheme can be seen as an instance of the reverse search technique [2] used for designing efficient algorithms for hard enumeration problems.

One way to enumerate all MXSTs is to visit all the leaf nodes representing all potential MXSTs. However, this will involve traversing the complete tree and can lead to a combinatorial explosion due to the number of ASTs. To overcome this, we introduce a pruning strategy that decides in polynomial time if the subtree rooted at an intermediate node in the enumeration tree cannot lead to a MXSTs. An AST is enumerated by combining two smaller ASTs. However, the two smaller ASTs can be joined in more than one way. A subtree arising out of their combination would be an AST only if the two ASTs join in the same way in all the input trees. Thus, counting the support of the larger AST involves identifying how the smaller ASTs combine in a given input tree. For this, we propose a one-time LCA based preprocessing step – polynomial in the number of trees and the size of the leafset – that can answer this in constant time. We next describe each of these steps.

3.1 Canonical Form

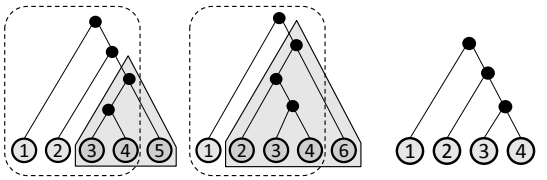
To enumerate only one subtree from a collection of isomorphic MXSTs, we represent subtrees in a canonical form such that all trees in an isomorphic collection have the same canonical form and such that trees from different isomorphic collections have different canonical forms. We use the canonical form proposed by [25]. Assume without loss of generality that the leaf label set $\mathcal{L}$ consists of integers in the range $[1, |\mathcal{L}|]$ so that the labels are ordered. The canonical form assigns every internal node a virtual label in $\mathcal{L}$, which is the minimum among all its leaf descendants. The children of an internal node are ordered from left to right based on the sequence in which they are encountered in an inorder dept first traversal (IDFT), the leftmost child being encountered first. A tree T is in ***canonical form*** if for every internal node its children are ordered from left to right based on their virtual labels.

3.2 Enumeration Tree

The next set of definitions are essential in describing the enumeration tree. The ***rightmost leaf*** of tree T is the last leaf encountered in the IDFT of T. A useful property of the canonical form is that pruning of either the last leaf (deleting the leaf and suppressing the degree two nodes) or the second last leaf encountered in the IDFT, results in a subtree that is also canonical [25]. The resulting subtree after pruning the rightmost leaf is called the ***prefix tree***. For a tree T, we refer to its prefix subtree as simply *prefix*. The ***heaviest subtree*** [25] is the subtree rooted at the parent of the rightmost leaf.

Two trees in their canonical form are considered equivalent or are said to belong to the same ***equivalence class*** if they share a common prefix. We call this common prefix tree the ***core tree***. Thus, an equivalence class of k-leaf trees will have a $(k-1)$-leaf core tree. Any two trees in an equivalence class differ only with respect to their rightmost leaf, therefore, topologically their difference is restricted to their heaviest subtrees. Figure 4 illustrates the defined concepts. The equivalence relation partitions any set of canonical trees into disjoint subsets – each subset is an equivalence classes identified by its unique core tree. A canonical tree T belongs to a unique equivalence class E if the prefix of T is the core tree of E. We use this as the starting point in defining the enumeration tree – each node in the enumeration tree represents a unique equivalence class. An equivalence class E is the parent of F if the core tree of F belongs to E. Clearly each node has a unique parent. Based on this we call an AST T to be the parent of another AST T' if the equivalence class that has T as its core tree is the parent of the equivalence class that has T' as its core tree i.e. T is the prefix of T'. A leaf node represents an empty equivalence class and is recognized by its unique core tree. It is special as it indicates that its core tree cannot be extended further to be contained as the prefix in any other AST. Thus, every MXST must correspond to the core tree of some leaf node. The root of this enumeration tree is an empty node that has all the equivalence classes containing 3-leaf ASTs as its immediate children. This is because three is the minimum number of leaves

on which phylogenetic inference can be meaningful. For an equivalence class E, the *branch* at E represents the subtree induced by all the leaf descendants of E. ASTs X and Y are considered to be of a *common descent* if neither is a descendant of the other.



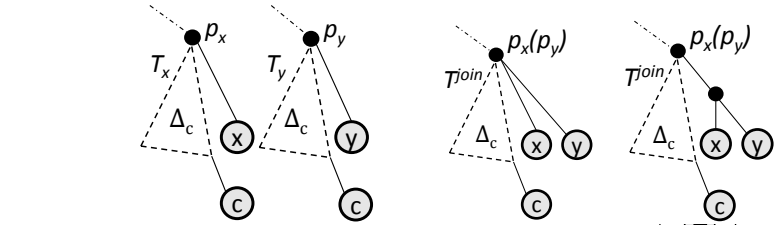
(a) Equivalence class (b) Prefix tree

Fig. 4. 4a shows two trees belonging to the same equivalence class. The common prefix tree is encircled by the dotted lines; the respective rightmost leaves are the ones outside the dotted lines. The shaded part represents the respective heaviest subtrees. 4b shows the common prefix tree for the trees in 4a.

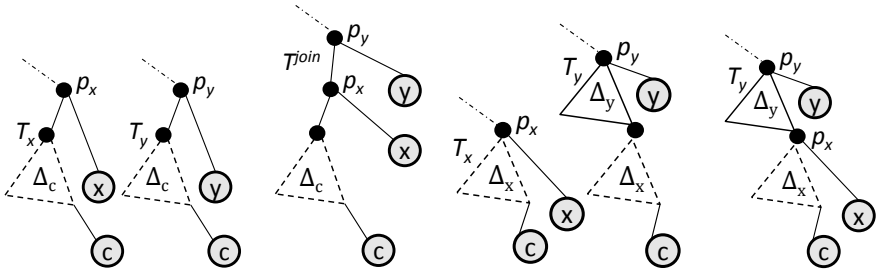
3.3 Pairwise Join

Consider equivalence classes E and F, where E is the parent of F. Consider tree $T_f \in F$. Let T_f^1 and T_f^2 be the subtrees of T_f obtained after pruning the last and the second last leaf in the IDFT of T_f. Then, T_f^1 represents the core tree of F and belongs to E. Further, T_f^1 and T_f^2 are canonical and share a common prefix. Therefore, T_f^2 also belongs to E. This shows that any such $T_f \in F$ can be obtained by 'joining' a unique ordered pair (T_f^1, T_f^2) in E. This leads to a natural formulation for generating all children of E. For every $T_x \in E$, create a new child F of E that has T_x as its core tree. For every $T_y \in E$ such that $T_x \neq T_y$, check if (T_x, T_y) are joined in a unique way in the all the trees in the input collection. A join refers to the type of subtree displayed by an input tree over the leafset $\mathcal{L}_{T_x} \cup \mathcal{L}_{T_y}$. If this subtree is of the same type across all the input trees, then we say that (T_x, T_y) are joined in a unique way in all the trees in the input collection. Such a unique join is an AST and is added to F.

Given (T_x, T_y) in an equivalence class E, note that for any join on (T_x, T_y) to be considered as an AST, it must be (a) canonical, (b) have T_x as its prefix and (c) have T_y as its subtree. Let (a)—(c) be referred to as condition C. We next describe the four possible ways in which the join of T_x and T_y can exist in an input tree. We refer to the joins as Type $1-4$. In the subsequent discussion, let x and y denote the rightmost leaf of T_x and T_y respectively, p_x and p_y denote the parents of x and y respectively, and T^{core} represent the core of the equivalence class E. For an internal node u, let numChild(u) denote its number of children. Different type of joins arise due to the relative values of depth(p_y) and depth(p_x).



(a) T_x and T_y for Type 1 and 2 (b) Type 1 join (c) Type 2 join



(d) T_x and T_y for Type 3 (e) Type 3 join (f) T_x and T_y for Type 4 (g) Type 4 join

Fig. 5. Different types of pairwise join. A dotted triangle represents part of the tree which may be empty while a solid triangle represents a non-empty part of the tree. Δ reflects topologies of the heaviest subtrees. 'c' denotes the rightmost leaf of the common core tree.

Type 1 and 2: These require $\text{depth}(p_y) = \text{depth}(p_x)$. Figure 5a shows the participating trees for join Type 1 and 2. Leaves x and y are attached at the same depth on the rightmost path of T^{core}. Figure 5b and 5c show the resulting joins for Type 1 and 2 respectively. Here, x and y are attached as siblings to the same pendant node in the joined tree. Thus, for the resulting joined tree to be canonical, $\Phi(x) < \Phi(y)$ must hold.

Type 3: This join is a special case of $\text{depth}(p_y) = \text{depth}(p_x)$ where p_y becomes the parent of p_x in the resulting tree as shown in Figure 5e. For this to be true $\text{numChild}(p_y) = \text{numChild}(p_x) = 2$ must hold; else p_x and p_y cannot exist at the same depth on the rightmost path of T^{core}. Figure 5d illustrates the participating trees for Type 3 join. Clearly, it is a special case of the input trees for Type 1 and 2.

Type 4: In this case $\text{depth}(p_y) < \text{depth}(p_x)$. Figure 5f shows the participating trees for this case. On the rightmost path of T^{core}, leaf y is attached at a lesser depth than leaf x. As shown in Figure 5g, there is only one way in which T_x and T_y can be joined because the resulting join must satisfy condition C. Here, p_y becomes an ancestor of p_x in the joined tree.

Observe that for $\text{depth}(p_y) > \text{depth}(p_x)$, a join operation is not possible since T_x cannot be the prefix tree of any join and this violates condition C. ASTs from such joins are enumerated when considering the ordered pair (T_y, T_x).

3.4 Support Estimation

Given (T_x, T_y) in an equivalence class and an input tree T, we say the join induced by (T_x, T_y) in T is of Type A or (T_x, T_y) are joined in T as Type A if the subtree induced in T by the combined leafset $\mathcal{L}_{T_x} \cup \mathcal{L}_{T_y}$ corresponds to Type A join with respect to (T_x, T_y). Let T^{join} denote the restriction of T over the combined leafset. This step classifies T^{join} as one of the four join types. If a particular join is supported by all the input trees i.e. $f = 1$, then the corresponding join is an AST. A straight forward way to identity T^{join} could be to actually restrict T over the combined leafset and identify the restricted tree as one of the four join types. However, this will require time linear in the size of T. Here we describe a least common ancestor (LCA) based formulation that identifies T^{join} in constant time. The LCA values are computed as a preprocessing step. The next result gives precise conditions for identifying T^{join}. The meaning of the symbols c, x, y, p_x and p_y is the same as in Section 3.3. Superscripts indicate the reference tree.

Theorem 1. *1. T^{join} is of Type 1 join if and only if*

(a) $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(c, y))$
(b) $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(x, y))$
(c) $\Phi(x) < \Phi(y)$

2. T^{join} is of Type 2 join if and only if

(a) $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(c, y))$
(b) $\text{depth}(\text{LCA}^T(c, x)) < \text{depth}(\text{LCA}^T(x, y))$
(c) $\Phi(x) < \Phi(y)$

3. T^{join} is of Type 3 join if and only if

(a) $\text{depth}(\text{LCA}^T(c, x)) > \text{depth}(\text{LCA}^T(c, y))$
(b) $\text{depth}^{T_x}(p_x) = \text{depth}^{T_y}(p_y)$

4. T^{join} is of Type 4 join if and only if

(a) $\text{depth}^{T_x}(p_x) > \text{depth}^{T_y}(p_y)$

Proof. Let us consider each of the cases separately.

1. Clearly if T^{join} is of Type 1 join, then it satisfies 1a-1c. To prove the *only if* part, let 1a-1c be satisfied. Since T_x and T_y are obtained by attaching x and y respectively to the rightmost path of T^{core}, 1a implies that $\text{depth}(p_y) = \text{depth}(p_x)$. Thus, T^{join} is either of Type 1, 2 or 3 join. Type 3 join requires the parent of y to be an ancestor of the parent of x in T^{join} – a case that is ruled out by 1a. Further, 1b and 1c imply that the join must be of Type 1.
2. The proof proceeds in a similar fashion as that for part 1. Again T^{join} can be either of Type 1 or 2 join. Conditions 2b and 2c imply that the join must be of Type 2.
3. Condition 3b implies that T^{join} must be of Type 1, 2 or 3 join. Condition 3a rules out Type 1 and 2 joins. Thus the join must be of Type 3.
4. As per 4a, T^{join} can be only of Type 4 join. □

Clearly the conditions 1—4 are mutually exclusive and each of the cases can be evaluated in constant time. The above scheme takes only constant time because it exploits the fact that T_x and T_y are already a subtree of T, and, T^{join} differs from each of T_x and T_y only with respect to their right most leaf. Observe that all conditions in Theorem 1 involve comparison of the depth of the LCAs of two pairs of leaf nodes in T, rather than requiring the actual values. Moreover, every such two pairs have one leaf in common. Thus, we need not verify the join type in all the input trees individually. As a preprocessing step we create a map that for every set of three leaves (i, j, k) stores this comparison for pairs (i, j) and (i, k) where i is the common leaf among the two pairs. If a comparison is not the same for all the trees in the input collection, then the corresponding entry is flagged indicating that no one type of join is exhibited across all the input trees. Thus, in the case of ASTs for a given (T_x, T_y) it can be known in constant time if they join to result in an AST.

3.5 Pruning Strategy

Enumerating all MXSTs by visiting all the leaf nodes of the enumeration tree can lead to a combinatorial explosion due to the number of ASTs. To prevent this we introduce a pruning strategy that prunes a branch at a node in the enumeration tree if none of its leaf descendants contain a MXST. In the next set of definitions, X and Y represent ASTs.

Definition 1. ***(Pruned).*** *Y is considered* pruned *if for every descendant Y' of Y there exists an X such that X and Y are of a common descent and X displays Y'. In such a case none of the descendants of Y can be a MXST. Thus, the branch at Y can be safely pruned.*

Definition 2. ***(Singly Pruned).*** *Y is* singly pruned *by X if Y is pruned, X and Y are of a common descent, and for every descendant Y' of Y there exists at least one descendant X' of X that displays Y'.*

Note that if Y is pruned but not singly pruned, then the branch at Y can be partitioned into sub-branches each of which is singly pruned. In the the worst case, each such sub-branch will correspond to a leaf node. The next result characterizes singly pruned nodes at the level of equivalence class. Let E denote an equivalence class with c as the rightmost leaf of its core tree. Let X, Y, Z be ASTs in E with x, y, z as their rightmost leaves respectively. Let E_x, E_y, E_z be the corresponding equivalence classes having X, Y, Z as their respective core trees. For any X and Y, let T_{xy} denote the pairwise join such that T_{xy} has X as its prefix. Let $\mathcal{L}_{xy}, \mathcal{L}_{yz}, \mathcal{L}_{xz}$ denote the leafsets of T_{xy}, T_{yz}, T_{xz} respectively. We say $[i, j, k]$ is an *agreement triplet* if the corresponding triplet on the leafset $\{i, j, k\}$ is the same for all the trees in the input collection.

Theorem 2. *1. X singly prunes Y if either of the following holds:*

(a) T_{xy} exists and is not of join Type 2.

(b) *T_{xy} exists as join Type 2 and for every $Z \in E$ such that T_{yz} exists, $[x, y, z]$ is an agreement triplet.*

2. *If T_{xy} and T_{yz} exist and T_{xy} is of join Type 2, then T_{xy} singly prunes T_{yz} if T_{yz} is not of join Type 2.*

Proof. 1. (a) Let T_{xy} be of join Type 1 with triplet $[c, x, y]$ of type (c, x, y). Consider any $T_{yz} \in E_y$. If T_{yz} is of join Type 1 with triplet $[c, y, z]$ of type (c, y, z), then any tree that displays both (c, x, y) and (c, y, z) must display (c, x, y, z) as well. Thus, $[c, x, z]$ and $[x, y, z]$ are also agreement triplets. Thus, $T_{xz} \in E_x$ must exist. Similarly, for T_{yz} of join Type 2, 3 and 4, it can be shown that $[c, x, z]$ and $[x, y, z]$ are agreement triplets and $T_{xz} \in E_x$ exists. Let this be called fact F.

Let T be a descendant of Y. Let $\mathcal{L}_d = \mathcal{L}_T - \mathcal{L}_Y$. If there exists an AST T' on $\{\mathcal{L}_T \cup x\}$, then it must be a descendant of X as $T_{xy} \in E_x$. For such a T' to exist, for every $\{a, b\} \in \mathcal{L}_d$, joins $\{T_{xa}, T_{xb}\} \in E_x$ must exist, and, $[x, y, a]$, $[x, y, b]$, $[x, a, b]$ must be agreement triplets. Since T_{xy} is of join Type 1 and $T_{ya} \in E_y$, as per fact F: $T_{xa} \in E_x$ exists and $[x, y, a]$ is an agreement triplet. Similarly, $T_{xb} \in E_x$ exists and $[x, y, b]$ is an agreement triplet. Since, $[x, y, a]$, $[x, y, b]$ and $[y, a, b]$ are agreement triplets, $[x, a, b]$ necessarily is an agreement triplet. Thus for every descendant T of Y, there exists at least one descendant T' of X that displays T. Thus, X singly prunes Y if T_{xy} is of join Type 1. Using similar arguments, it can be shown that for T_{xy} of join Type 3 and 4, X singly prunes Y.

(b) For any T_{yz}, $[x, y, z]$ is an agreement triplet. Further, since T_{yz} and T_{xy} exist, $[c, x, y]$ and $[c, y, z]$ are also agreement triplets. Thus, $[c, x, z]$ is necessarily an agreement triplet. Thus, $T_{xz} \in E_x$ exists. Let this be called fact F'.

Again, let T be a descendant of Y. Let $\mathcal{L}_d = \mathcal{L}_T - \mathcal{L}_Y$. If there exists an AST T' on $\{\mathcal{L}_T \cup x\}$, then it must be a descendant of X as $T_{xy} \in E_x$. For such a T' to exist, for any $\{a, b\} \in \mathcal{L}_d$, joins $\{T_{xa}, T_{xb}\} \in E_x$ must exist, and, $[x, y, a]$, $[x, y, b]$, $[x, a, b]$ must be agreement triplets. Since T_{ya} exists, as per fact F': $T_{xa} \in E_x$ exists and $[x, y, a]$ is an agreement triplet. Similarly, $T_{xb} \in E_x$ exists and $[x, y, b]$ is an agreement triplet. Again, since $[x, y, a]$, $[x, y, b]$ and $[y, a, b]$ are agreement triplets, $[x, a, b]$ necessarily is an agreement triplet. Thus for every descendant T of Y, there exists at least one descendant T' of X that displays T. Thus, X singly prunes Y.

2. Let T_{xy} be of join Type 2 with triplet $[c, x, y]$ of type $(c, (x, y))$. Let E_{xy} denote the equivalence class that has T_{xy} as its core tree. If T_{yz} is of join Type 1 with triplet $[c, y, z]$ of type (c, y, z), then any tree that displays both $(c, (x, y))$ and (c, y, z) must display $(c, (x, y), z)$ as well. Thus, $[c, x, z]$ and $[x, y, z]$ are also agreement triplets. Thus, $T_{xz} \in E_x$ exists. Since $\{T_{xy}, T_{xz}\} \in E_x$ and $[x, y, z]$ is an agreement triplet, the join T_{xy-z} on (T_{xy}, T_{xz}) exists and $T_{xy-z} \in E_{xy}$. Similarly, for T_{yz} of join Type 3 and 4, it can be shown that $[c, x, z]$ and $[x, y, z]$ are agreement triplets, $T_{xz} \in E_x$ and $T_{xy-z} \in E_{xy}$ exists. Let this be called fact F''.

Let T be a descendant of T_{yz}. Let $\mathcal{L}_d = \mathcal{L}_T - \mathcal{L}_{yz}$. If there exists an AST T' on $\{\mathcal{L}_T \cup x\}$, then it must be a descendant of E_{xy} as $T_{xy-z} \in E_{xy}$. For such a T' to exist, for any $\{a, b\} \in \mathcal{L}_d$, joins $\{T_{xy-a}, T_{xy-b}\} \in E_{xy}$ must exist, and $[x, y, a]$, $[x, y, b]$, $[x, z, a]$, $[x, z, b]$, $[x, a, b]$ must be agreement triplets. Consider join T_{ya}. Let it be of Type 2. Thus, triplet $[c, y, a]$ is of type $(c, (y, a))$. Since T is a descendant of T_{yz}, thus the join T_{yz-a} on (T_{yz}, T_{ya}) must exist and $T_{yz-a} \in E_{yz}$. If T_{yz} is of Type 1 join with triplet $[c, y, z]$ of type (c, y, z), then any tree that displays both $(c, (y, a))$ and (c, y, z) must also display $(c, (y, a), z))$. However, $(c, (y, a), z))$ cannot exist in T_{yz-a} as a cannot be the rightmost leaf. Similarly for T_{yz} of join Type 3 or 4, and for T_{ya} of join Type 2, it can be shown that a cannot be the rightmost leaf in T_{yz-a}. Thus, T_{ya} is not of join Type 2. Thus, by fact F'', $T_{xy-a} \in E_{xy}$ exists and $[x, y, a]$ is an agreement triplet. Since $[x, y, z]$, $[x, y, a]$, $[y, z, a]$ are agreement triplets, $[x, z, a]$ is necessarily an agreement. Similarly, $T_{xy-b} \in E_{xy}$ exists and, $[x, y, b]$, $[x, z, b]$ are agreement triplets. Since, $[x, y, a]$, $[x, y, b]$ and $[y, a, b,]$ are agreement triplets, $[x, a, b]$ is necessarily an agreement triplet. Thus, for every descendant T of T_{yz}, there exists at least one descendant T' of E_{xy} that displays T. Thus, E_{xy} (or T_{xy}) singly prunes T_{yz}. □

Pruner-list. Note that 1a and 1b can be evaluated in constant time while 2 will take linear time. Neither of these require enumeration of the pruned branch at Y. However, the case when Y is not singly pruned by any X but a descendant Y' of Y is singly pruned by X cannot be identified using Theorem 2. For such a case, the branch at Y need to be enumerated looking for such Y'. To do this efficiently, we maintain a pruner-list for every child of E_y. For ASTs X, Y, Z in an equivalence class E, such that joins T_{xy}, T_{yz} exist and none of the cases of Theorem 2 hold, *pruner-list* of T_{yz} contains x if $[x, y, z]$ is an agreement triplet. To describe how the pruner-list is propagated in the branch at Y, let T_1, T_2, T_{12} be descendants of Y such that T_{12} is a join on $\{T_1, T_2\}$ and, $T_{12} \in E_1$ – the equivalence class with T_1 as its core tree. Then the pruner-list of T_{12} is the intersection of the pruner lists of T_1 and T_2. Now, E_1 is singly pruned by X if all members of E_1 have x in their pruner-list. This can be shown by using arguments similar to the proof of 1b.

3.6 MfstMiner Algorithm

Figure 6 gives a high-level description of the MfstMiner algorithm for the special case of enumerating all MXSTs. C is the collection of input trees. First all AST triplets are enumerated. They are then partitioned as the set of all equivalence classes consisting of ASTs on three leaves denoted by EC_3. Each equivalence class in EC_3 represents a child of the root of the enumeration tree as defined in section 3.2. Subroutine enumerateNode accepts an equivalence class E as input and enumerates the branch at E in the enumeration tree. Children of E that need to be further enumerated are stored in enumList, while those that are potential MXSTs are stored in outputList. Line 13 corresponds to Y being singly pruned by X as per Theorem 2.1a. Line 15 corresponds to T_{xy}

```
MfstMiner(C)
 1: computeLCA_Mappings(C)
 2: At ←enumerateAST_Triplets(C)
 3: EC3 ← computeEquivalenceClasses(At)
 4: for all E ∈ EC3 do
 5:    enumerateNode(E)

    enumerateNode(E)
 6: outputList ← ∅, enumList ← ∅
 7: for all X ∈ E do
 8:    if X is not pruned then
 9:       Ex ← ∅
10:       for all Y ∈ E such that X ≠ Y do
11:          if join Txy exists then
12:             if Txy is not of join Type 2 then
13:                mark Y as pruned
14:                if x.prunerList ≠ ∅ then
15:                   mark Txy as pruned
16:             else {Txy is of join Type 2}
17:                y.prunerList ← y.prunerList ∪ x
18:             Txy.prunerList ← X.prunerList ∩ Y.prunerList
19:             Txy.prunerList ← Txy.prunerList ∪ x.prunerList
20:             Ex ← Ex ∪ Txy
21:       if Ex = ∅ then
22:          outputList ← outputList ∪ X
23:       else
24:          enumList ← enumList ∪ Ex
25: for all X ∈ outputList such that X is not pruned and X.prunerList = ∅ do
26:    print X
27: for all Ey ∈ enumList such that Y is not pruned do
28:    for all x ∈ y.prunerList do
29:       if ∀Tyz ∈ Ey, [x, y, z] is an agreement triplet then
30:          mark Y as pruned
31:    if Y is not pruned then
32:       enumerateNode(Ey)
```

Fig. 6. The case of enumerating all MXSTs.

be singly pruned by some element in x.prunerList as per Theorem 2.2. There are two types of pruner-lists used in the algorithm. X.prunerList represents the pruner-list of the the AST X that has been propagated to it from its ancestor nodes, while x.prunerList represents the pruner-list for the label x generated within the equivalence class E. Line 18 shows the propagation of the pruner-list to the descendant join T_{xy}. In line 19 new members get added to the pruner-list of T_{xy} due to the members of the current equivalence class. In line 22 the empty equivalence class E_x (i.e. a leaf node) corresponds to a potential MXSTs and is added to the outputList. In line 26, it is produced as output if it is neither pruned by members of the current equivalence class nor by the members

of the ancestor equivalence classes. If E_x is not empty then it is added to the enumList for potential enumeration of the branch at E_x. Loop 27-30 corresponds to the pruning of Y in enumList as per Theorem 2.1b. If such a Y is not pruned by any of the cases, then the corresponding equivalence class E_y is recursively enumerated.

The General Case of Enumerating MFSTs. Having outlined the algorithmic framework of MFSTMINER for the special case of enumerating all MXSTs, in this section we discuss the general case of mining MFSTs. The main difference is that the support for ASTs is always $f = 1$, while FSTs can additionally have any $f \in \left(\frac{1}{2}, 1\right)$. This does not affect the enumeration tree and the pairwise join. However, the support estimation and the pruning strategy need to incorporate the general case when the support is not 1. We discuss these steps next.

Support Estimation. Given (T_x, T_y) in an equivalence class, in the case of $f < 1$ a join T^{join} on (T_x, T_y) can be a FST if it is supported by at least fraction f of the input trees. To verify if such a T^{join} exists, we need to go through all the input trees individually to estimate its support. However, any such T^{join} can be supported only by those trees that support both T_x and T_y. For this, for each FST T_x we maintain the list of all trees in the input collection that support T_x. We call this list the *support-list* of T_x. For a FST T_x, let T_x.supList denote its support-list. Thus, to estimate if the join on (T_x, T_y) results in FST, we apply Theorem 1 only on trees in T_x.supList $\cap$ T_x.supList. We store the the support list as a bitmap representation [3] for efficient memory utilization and fast computation of intersection of two support-lists using logical operators.

Pruning Strategy. Given FSTs X and Y, while deciding if Y is pruned by X, we also need to consider the support-list of X and Y. We say $[x, y, z]$ is a *frequent triplet* if it is of the same type in at least fraction f of the input trees. Let $[x, y, z]$.supList denote the support-list of such a frequent triplet. Based on this, for the case of enumerating MFSTs, Theorem 2 can be restated as:

Theorem 3. *1. X singly prunes Y if either of the following holds:*
   (a) T_{xy} exists, Y.supList $\subseteq$ X.supList and T_{xy} is not of join Type 2.
   (b) T_{xy} exists as join Type 2, Y.supList $\subseteq$ X.supList and for every $Z \in E$ such that T_{yz} exists, $[x, y, z]$ is a frequent triplet with Y.supList $\subseteq$ $[x, y, z]$.supList.

2. If T_{xy} and T_{yz} exist and T_{xy} is of join Type 2, then T_{xy} singly prunes T_{yz} if T_{yz}.supList $\subseteq$ T_{xy}.supList and T_{yz} is not of join Type 2.

Pruner-list. Pruning cases not identified by Theorem 3, require the use pruner-list. In the case of MFSTs along with leaf label the pruner-list also contains the list of input trees on which pruning holds. To explain further, let X, Y, Z be FSTs in an equivalence class E, such that joins T_{xy}, T_{yz} exist and none of the cases of Theorem 3 hold, then the next set of conditions describe the use and propagation of pruner-lists along the branch at T_{yz}.

1. T_{yz}.prunerList contains the entry $(x, S_\cap = T_{yz}.\text{supList} \cap X.\text{supList})$ if T_{xy} is not of join Type 2 and $|S_\cap| \geq f$.
2. T_{yz}.prunerList contains the entry $(x, S_\cap = T_{yz}.\text{supList} \cap X.\text{supList} \cap [x, y, z].\text{supList})$ if T_{xy} is of join Type 2, $[x, y, z]$ is a frequent triplet and $|S_\cap| \geq f$.
3. For every label w such that $(w, S^y_\cap) \in Y.\text{prunerList}$ and $(w, S^z_\cap) \in Z.\text{prunerList}$ exist, T_{yz}.prunerList contains the entry $(w, S_\cap = T_{yz}.\text{supList} \cap S^y_\cap \cap S^z_\cap)$ if $|S_\cap| \geq f$.

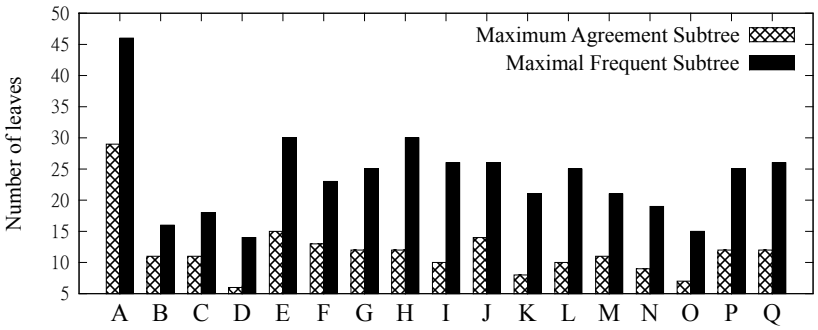
Condition 1 and 2 describe addition of new labels to the pruner-list T_{yz}, while condition 3 describes inheritance of labels from the pruner-list of Y and Z.

This completes the description of MfstMiner for the general case of mining MFSTs. The overall framework is the same as the special case of mining all MXSTs. The difference lies in the finer details of incorporating support-list in the support estimation and the pruning step. These details were discussed in this section and can be easily incorporated in Algorithm 6.

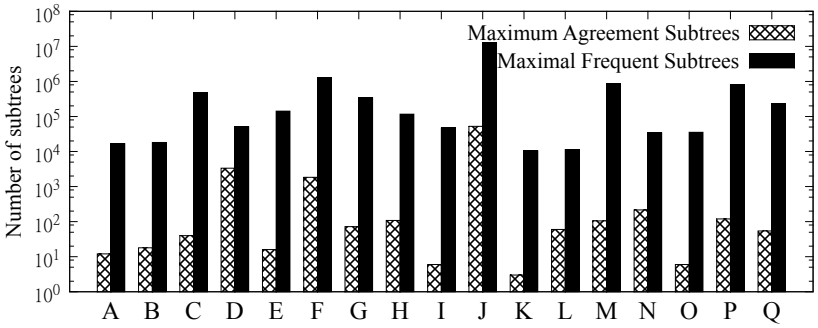
4 Experiments and Results

To demonstrate the effectiveness of MfstMiner, we conducted three category of experiments. The first category demonstrates the advantage of MFSTs over MAST. The second category compares MfstMiner with Phylominer [25] – an algorithm that enumerates all FSTs. The third category evaluates the scalability of MfstMiner with respect to the number of trees and the size of the leafset. We use the dataset of bootstrapped trees used in a previous study [17] on majority rule trees. Trees were constructed using 17 DNA alignments containing 125 up to 2,554 sequences. It spans a diverse range of sequences including rbcL genes, mammalian sequences, bacterial and archaeal sequences, ITS sequences, fungal sequences , and grasses. We order the alignments based on the increasing number of sequences and refer to the datasets as $A - Q$. To extract datasets of different sizes (in terms of the number of leaves and the number of trees), we randomly selected the required number of trees and restricted them on a random set of leaves of the required size.

Utility of MFSTs over MASTs. Figure 7 highlights the advantage of MFSTs over MASTs. Figure 7a compares the size of the MAST with the size of the largest MFST. This experiment was conducted on a set of 100 trees on 50 leaves from each of the datasets. MFSTs were enumerated for $f = 0.51$. In some cases the size of the largest MFST is more than twice the size of the corresponding MAST. This clearly shows that MFSTs are capable of revealing significantly larger consensus subtrees than MAST. Figure 7b compares the number of MASTs with the number of MFSTs for $f = 1$. The number of MFSTs is significantly more. This is especially notable in the light of the fact that any MFST that is not a MAST is also not displayed by any of the MASTs. Thus, such a MFST reveals a unique agreement information among the input trees. This experiment was conducted on a set of 100 trees on 100 leaves from each of the datasets.



(a) MFSTs have more number of leaves

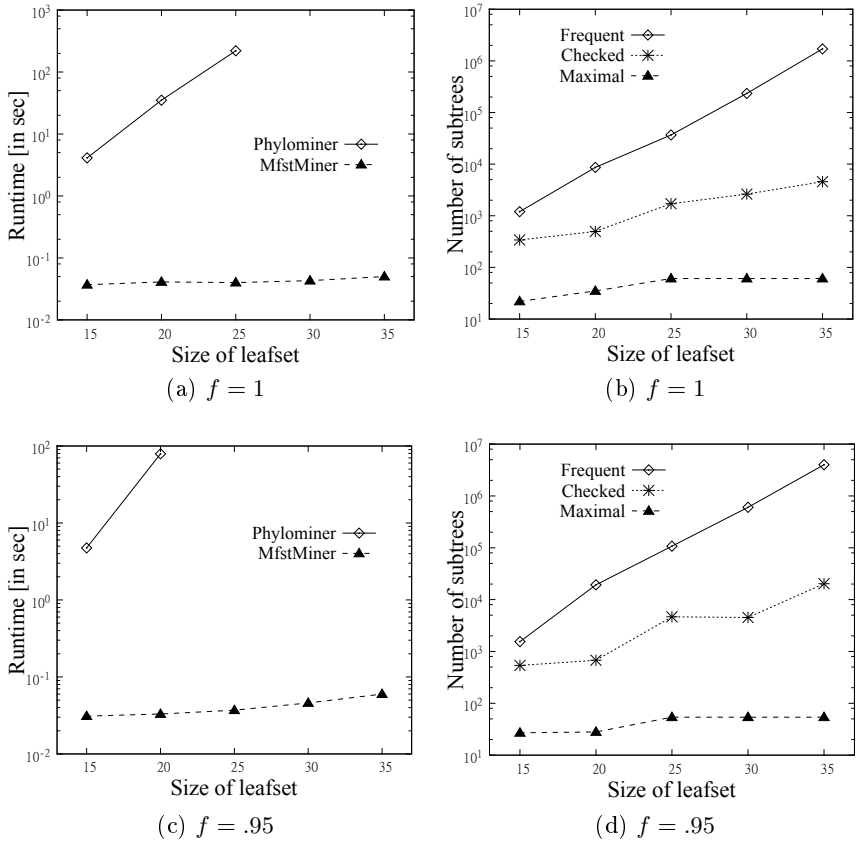


(b) MFSTs are more in number

Fig. 7. Utility of MFSTs over MASTs.

Comparison with Phylominer. Figure 8a compares MfstMiner with Phylominer [25] for $f = 1$. Clearly, enumerating MFSTs is orders of magnitude faster than enumerating all FSTs. Figure 8b shows the corresponding counts for FSTs, MFSTs and the FSTs checked by MfstMiner while enumerating all MFSTs. While the number of MFSTs and the number of FSTs checked by MfstMiner exhibit polynomial behavior, the number of FSTs grow exponentially. This experiment was done on dataset A with 100 trees for each of the leafset. Figure 8c-8d show the corresponding numbers for $f = .95$. For these experiments the physical memory was capped at the 4GB limit. Phylominer uses a scheme that requires all the enumerated FSTs to be kept in the memory. This explains the missing entries in the case of Phylominer. MfstMiner uses a depth-first scheme to traverse the enumeration tree and only needs to keep FSTs along a branch. Thus, it is not confounded by the memory limit.

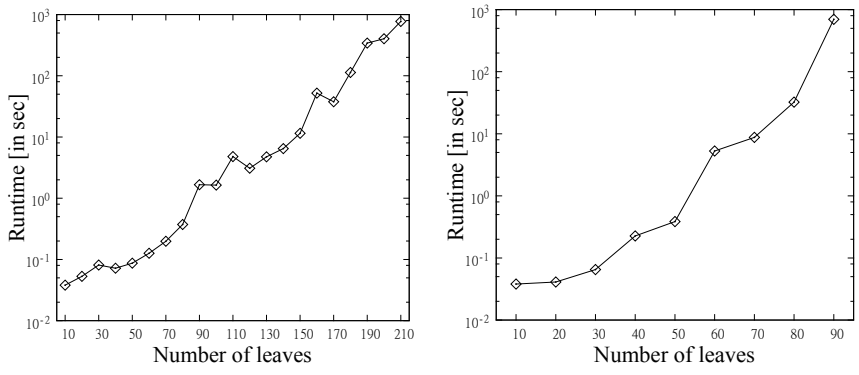
Scalability of MfstMiner. Figure 9 shows the scalability of MfstMiner with respect to the number of leaves. The first experiment for $f = 1$ (Figure 9a) was done on dataset P. The second experiment for $f = .95$ (Figure 9b) was done



(a) $f = 1$ (b) $f = 1$

(c) $f = .95$ (d) $f = .95$

Fig. 8. Comparison with Phylominer



(a) $f = 1$ (b) $f = .95$

Fig. 9. Scalability of MfstMiner

on dataset Q. For each leafset, 100 trees were extracted for the corresponding dataset.

5 Conclusion

We introduced a new algorithm to mine MFSTs in collections of phylogenetic trees. We highlighted the utility of MFSTs over MASTs as being capable of producing larger and novel agreement subtrees not displayed by any of the MASTs. At the same time the set of all MFSTs is a compact and non-redundant summary of the set of all FSTs. We demonstrated this through experiments on biological datasets, compared the efficiency of our approach with Phylominer [25] – an algorithm to enumerate all FSTs, and showed its scalability for larger leafsets. The current implementation of MfstMiner can be downloaded from http://www.cs.iastate.edu/~akshayd/mfstMiner/. The current implementation works for up to 250 leaves and 10,000 trees. Further, our approach can enumerate MFSTs for collections of trees having partially overlapping leafsets. This cannot be applied to MASTs especially in the case where the common overlap among all the input trees is low.

As a future work, we intend to use MFSTs in practical applications that involve MASTs [12,7,6]. We note that the enumeration of MFSTs can take long for larger leafsets. In this regard, we intend to develop a scheme that can randomly sample MFSTs – giving a smaller result set that is randomly sampled from the entire solution set. We note that while enumeration of FSTs is possible for $f \in \left(0, \frac{1}{2}\right]$ as well, this can potentially lead to two different FSTs over the same leafset. Further, as a consensus approach the subtrees supported by less than majority cannot be favorably considered. However, it would be interesting to explore applications of FSTs for $f \in \left(0, \frac{1}{2}\right]$. With little modification in the pruning strategy, the proposed algorithms can be extended to enumerate such FSTs.

Acknowledgments

This work was supported in part by National Science Foundation grant DEB-0829674. The authors thank Drs. Sen Zhang and Jason T. L. Wang for sharing the source code of Phylominer and discussions on their work. They also thank Dr. Nicholas D. Pattengale for sharing the Bootstrapped dataset.

References

1. Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees. SIAM Journal on Computing 26, 758–769 (1994)
2. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65(1), 21–46 (1996)

3. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 429–435. ACM (2002)
4. Bryant, D.: Building trees, hunting for trees and comparing trees. Ph.D. thesis, Univ. of Canterbury, New Zealand (1997)
5. Chi, Y., Xia, Y., Yang, Y., Muntz, R.: Mining closed and maximal frequent subtrees from databases of labeled rooted trees. IEEE Trans. Knowl. Data Eng. 17, 190–202 (2005)
6. Daubin, V., Gouy, M., Perrière, G.: A phylogenomic approach to bacterial phylogeny: evidence of a core of genes sharing a common history. Genome Research 12(7), 1080–1090 (2002)
7. De Vienne, D., Giraud, T., Martin, O.: A congruence index for testing topological similarity between trees. Bioinformatics 23(23), 3119–3124 (2007)
8. Dong, S., Kraemer, E.: Calculation, visualization, and manipulation of masts (maximum agreement subtrees). Proceedings IEEE Computational Systems Bioinformatics Conference CSB IEEE Computational Systems Bioinformatics Conference 0(Csb), 405–414 (2004)
9. Farach, M., Przytycka, T., Thorup, M.: On the agreement of many trees. Information Processing Letters 55(6), 297–301 (1995)
10. Farach, M., Thorup, M.: Fast comparison of evolutionary trees. In: In Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 481–488 (1994)
11. Finden, C., Gordon, A.: Obtaining common pruned trees. Journal of Classification 2(1), 255–276 (1985)
12. Goddard, W., Kubicka, E., Kubicki, G., McMorris, F.: The agreement metric for labeled binary trees. Mathematical Biosciences 123(2), 215–226 (1994)
13. Guillemot, S., Berry, V.: Fixed-parameter tractability of the maximum agreement supertree problem. IEEE/ACM Trans. Comput. Biol. Bioinformatics 7(2), 342–353 (2010)
14. Huan, J., Wang, W., Prins, J., Yang, J.: Spin: mining maximal frequent subgraphs from graph databases. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 581–586. ACM (2004)
15. Kao, M., Lam, T., Sung, W., Ting, H.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. Journal of Algorithms 40(2), 212–233 (2001)
16. Lapointe, F., Rissler, L.: Congruence, consensus, and the comparative phylogeography of codistributed species in california. The American Naturalist 166(2), 290–299 (2005)
17. Pattengale, N., Aberer, A., Swenson, K., Stamatakis, A., Moret, B.: Uncovering Hidden Phylogenetic Consensus in Large Datasets. IEEE/ACM Trans. Comput. Biol. Bioinformatics 8-4(99), 1–1 (2011)
18. Sanderson, M., McMahon, M., Steel, M.: Terraces in phylogenetic tree space. Science 333(6041), 448 (2011)
19. Steel, M.: The complexity of reconstructing trees from qualitative characters and subtrees. Journal of Classification 9(1), 91–116 (1992)
20. Steel, M., Warnow, T.: Kaikoura tree theorems: computing the maximum agreement subtree. Information Processing Letters 48(2), 77–82 (1993)
21. Swenson, K., Chen, E., Pattengale, N., Sankoff, D.: The Kernel of Maximum Agreement Subtrees. In: Proc. International Symposium on Bioinformatics Research and Applications. pp. 123–135. Springer (2011)

22. Thomas, L., Valluri, S., Karlapalem, K.: Margin: Maximal frequent subgraph mining. In: Proc. IEEE International Conference on Data Mining. pp. 1097–1101. IEEE (2006)
23. Wang, K., Liu, H.: Discovering typical structures of documents: a road map approach. In: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval. pp. 146–154. ACM (1998)
24. Xiao, Y., Yao, J.: Efficient data mining for maximal frequent subtrees. In: Proc. IEEE International Conference on Data Mining. pp. 379–386. IEEE (2003)
25. Zhang, S., Wang, J.: Discovering frequent agreement subtrees from phylogenetic data. IEEE Trans. Knowl. Data Eng. 20(1), 68–82 (2008)