Computing Maximum Agreement Forests without Cluster Partitioning is Folly*

Zhijiang Li[1] and Norbert Zeh[2]

1 **Microsoft Canada, Vancouver, BC, Canada**
`zhijiang.li@dal.ca`

2 **Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada**
`nzeh@cs.dal.ca`

Abstract

Computing a maximum (acyclic) agreement forest (M(A)AF) of a pair of phylogenetic trees is known to be fixed-parameter tractable; the two main techniques are kernelization and depth-bounded search. In theory, kernelization-based algorithms for this problem are not competitive, but they perform remarkably well in practice. We shed light on why this is the case. Our results show that, probably unsurprisingly, the kernel is often much smaller in practice than the theoretical worst case, but not small enough to fully explain the good performance of these algorithms. The key to performance is *cluster partitioning*, a technique used in almost all fast M(A)AF algorithms. In theory, cluster partitioning does not help: some instances are highly clusterable, others not at all. However, our experiments show that cluster partitioning leads to substantial performance improvements for kernelization-based M(A)AF algorithms. In contrast, kernelizing the individual clusters before solving them using exponential search yields only very modest performance improvements or even hurts performance; for the vast majority of inputs, kernelization leads to no reduction in the maximal cluster size at all. The choice of the algorithm applied to solve individual clusters also significantly impacts performance, even though our limited experiment to evaluate this produced no clear winner; depth-bounded search, exponential search *interleaved* with kernelization, and an ILP-based algorithm all achieved competitive performance.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases Fixed-parameter tractability, agreement forests, hybridization, subtree prune-and-regraft

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.56

1 Introduction

Phylogenetic trees are the classical model of evolution. All extant taxa are assumed to descend from the same common ancestor and diverge in a tree-like fashion through speciation events. While this is still the accepted model for the evolution of individual genes, the evolution particularly of microbial organisms and plants is complicated by *reticulation events*, such as *lateral gene transfer* (LGT) and *hybridization*, which are known to play an important role, for example, in the development of antibiotic resistance of bacteria [20]. LGT allows an organism to acquire genetic material from an unrelated species in the same habitat. Hybridization allows an organism to inherit genetic material from more than one ancestor. Different genes shared by a group of taxa then have different tree-like evolutionary histories, which we

* This work was supported in part by NSERC and the Canada Research Chairs programme.

25th Annual European Symposium on Algorithms (ESA 2017).
Editors: Kirk Pruhs and Christian Sohler; Article No. 56; pp. 56:1–56:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

call *gene trees*. The differences between these trees provide the basis for discovering likely reticulation events in the evolution of this set of taxa.

A single LGT has the effect that, in the tree representing the transferred gene, the descendants of the recipient taxon appear genetically most similar to the descendants of the donor taxon, while all other relationships between taxa are preserved. Thus, the "true tree" can be transformed into the gene tree for the transferred gene by cutting off a subtree and grafting it onto the donor edge, a *subtree prune-and-regraft* (SPR) operation [11]. A series of LGTs translates into a sequence of SPR operations that transforms one input tree into the other. A set of hybridizations yields a network that *displays* each gene tree, that is, each tree can be obtained from the network by deleting edges and suppressing degree-2 nodes [2].

Reticulation events are assumed to be rare. Thus, it is common to assume that the smallest set of reticulations consistent with the input trees is the most likely scenario, and we aim to construct a hybridization network of the input trees with as few hybridizations as possible (its *hybridization number*) or a minimum-length sequence of SPR operations transforming one input tree into the other. The length of this sequence is the *SPR distance* between the two trees. Both problems are NP-hard [5, 7] and fixed-parameter tractable [7, 8, 14, 15, 18] when parameterized by the hybridization number or SPR distance. Despite these FPT results, solving either problem for more than two trees is challenging in practice. (It is unclear how to even extend the SPR distance to more than two trees. SPR supertrees [27] offer one possible approach.) For *two* input trees, very fast solutions exist [16, 17, 22, 23, 24, 25, 26]. Almost all of them use kernelization or depth-bounded search and compute the SPR distance or hybridization number via *maximum (acyclic) agreement forests* (M(A)AFs). The best known kernel sizes for MAF and MAAF of binary trees are $28k$ [7] and $14k$ [8], where k is the SPR distance or hybridization number. Combined with an $\mathrm{O}(3^n n)$-time M(A)AF algorithm [1], a MAF or MAAF can thus be found in $\mathrm{O}(3^{28k} n)$ or $\mathrm{O}(3^{14k} n)$ time. For multifurcating trees, the best known kernel sizes are $28k$ and $89k$ [18], and the exact M(A)AF algorithm takes $\mathrm{O}(4^n n)$ time. Thus, a MAF or MAAF can be found in $\mathrm{O}(4^{28k} n)$ or $\mathrm{O}(4^{89k} n)$ time. In contrast, the best depth-bounded search algorithms for M(A)AF take between $O(2^k n)$ and $O(5.08^k n)$ time [16, 17, 23, 24, 25, 26] depending on whether a MAF or MAAF is to be computed and whether the input trees are binary or multifurcating. This leaves an astronomical gap between the theoretical running times of kernelization-based and depth-bounded search algorithms for finding M(A)AFs of all but the simplest inputs. Yet, in practice, kernelization-based algorithms perform remarkably well [1, 9].

In this paper, we try to answer two questions: (1) Why do kernelization-based algorithms perform much better in practice than predicted in theory? (2) Which is the "ultimate" algorithm for computing agreement forests of two trees? Part of the answer to the first question is that the kernel is often much smaller than predicted, less than $4k$. This reduces the difference between the running times of kernelization-based and depth-bounded search algorithms significantly but still leaves a gap of more than 8^k even after porting the improvements from the depth-bounded search algorithms back to exponential search. This gap is massive, since values of $k \geq 50$ are not uncommon. The key to fast running times for almost all existing M(A)AF algorithms is *cluster partitioning* [3, 19], which allows us to break many non-trivial instances into smaller pieces that can be solved independently and whose total SPR distance or hybridization number (roughly) equals the SPR distance or hybridization number of the original input. This has the potential to lead to an exponential speed-up and often does in practice. We verified experimentally that cluster partitioning is the real reason why kernelization-based M(A)AF algorithms are fast: it significantly improves the performance of kernelization-based M(A)AF algorithms (and also of depth-bounded search algorithms [27]), while kernelization leads to only modest performance gains of algorithms using only cluster partitioning and exponential search and often even hurts performance. A recent theoretical result [6] shows that agreement-based phylogenetic distances are fixed-parameter tractable in the *level* of the optimal hybridization network, which is in fact exactly the maximum hybridization number of the clusters in a cluster partition. This sheds light on the effectiveness of cluster partitioning when it is applicable. Our results suggest that many real-world inputs are highly clusterable, that is, their optimal networks have small level. To answer question (2), we investigated which algorithm, used to solve the subproblems in a cluster partition, results in the fastest running time overall. In our somewhat limited experiments for this question, three winners emerged: depth-bounded search, integer linear programming, and *interleaving* of kernelization and exponential search.

Section 2 formally defines SPR distance, hybridization number, agreement forests, and related concepts. Section 3 gives an overview of the techniques used to compute agreement forests. Section 4 presents our experimental results. Section 5 offers conclusions.

2 Subtree Prune-and-Regraft, Hybridization, and Agreement Forests

A *(rooted phylogenetic) X-tree* is a tree T with a root labelled ρ and with $|X|$ leaves labelled bijectively with the elements in X; ρ has degree 1; all internal nodes have at least two children. Edges are directed away from the root. If all internal nodes have out-degree exactly two, T is *binary*; otherwise it is *multifurcating.* T is a *resolution* of another tree S if S can be obtained from T by contracting edges. Figures 1a,c illustrate these definitions.

A *(rooted phylogenetic) X-network* is a directed acyclic graph (DAG) with a single source ρ and $|X|$ sinks labelled bijectively with the elements in X. The nodes with in-degree at least two are called *hybrid nodes.* An X-network N *displays* an X-tree T if T can be obtained from N by deleting edges and suppressing unlabelled out-degree-1 nodes. Since we always suppress unlabelled out-degree-1 nodes, we do not state this explicitly from here on. N is a *hybridization network* of a pair of X-trees (S, T) if it displays both S and T. The *hybridization number* of N is the number of edges we need to delete to obtain a tree. The hybridization number $hyb(S, T)$ of a pair of X-trees (S, T) is the minimum hybridization number of all hybridization networks of (S, T). Figure 1e illustrates these definitions.

A *subtree prune-and-regraft (SPR) operation* on a binary X-tree T deletes the parent edge of some node v, splits some edge by introducing a new node u, and makes v a child of u. The *SPR distance* $d_{\mathrm{SPR}}(S, T)$ between two binary X-trees S and T is the minimum number of SPR operations needed to transform S into T; see Figure 1d.

A *(rooted binary) X-forest* is a forest F that can be obtained from a (binary) X-tree by deleting edges. An X-forest F_1 *refines* another X-forest F_2 if F_1 can be obtained from F_2 by deleting edges. An X-forest F is an *agreement forest* (AF) of a pair of binary X-forests (F_S, F_T) if it refines both F_S and F_T. A *maximum agreement forest* (MAF) of (F_S, F_T) is an AF of (F_S, F_T) with the minimum number of components; see Figure 1f. For a component C of a forest F that refines an X-tree T, let $\mathrm{LCA}_T(C)$ be the lowest common ancestor in T of all leaves of C. A component C_1 of F is an *ancestor* of another component C_2 of F in T if $\mathrm{LCA}_T(C_1)$ is an ancestor of $\mathrm{LCA}_T(C_2)$. The *ancestry graph* $G_T(F)$ of F w.r.t. T has the components of F as its nodes and contains a directed edge (C_1, C_2) if the ancestors of C_1 are exactly the proper ancestors of C_2. For an AF F of a pair of X-trees (S, T), let $G_{S,T}(F) = G_S(F) \cup G_T(F)$. We call F an *acyclic agreement forest* (AAF) of (S, T) if $G_{S,T}(F)$ is a DAG. A *maximum acyclic agreement forest* (MAAF) of (S, T) is an AAF of (S, T) with the minimum number of components. Figures 1f–i illustrate these definitions.

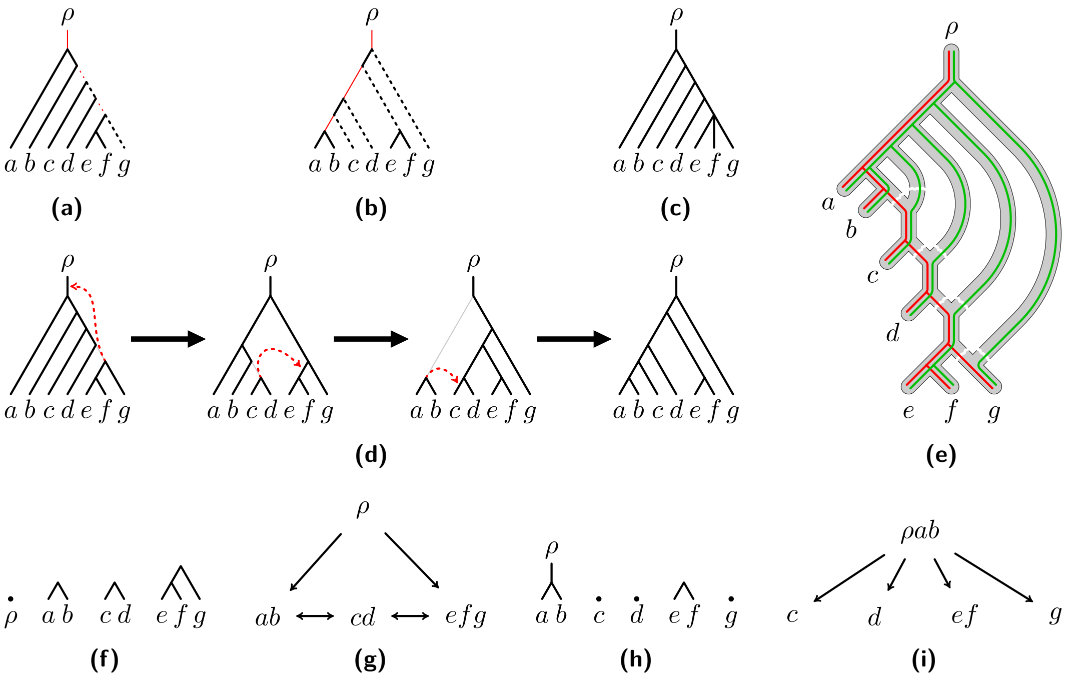


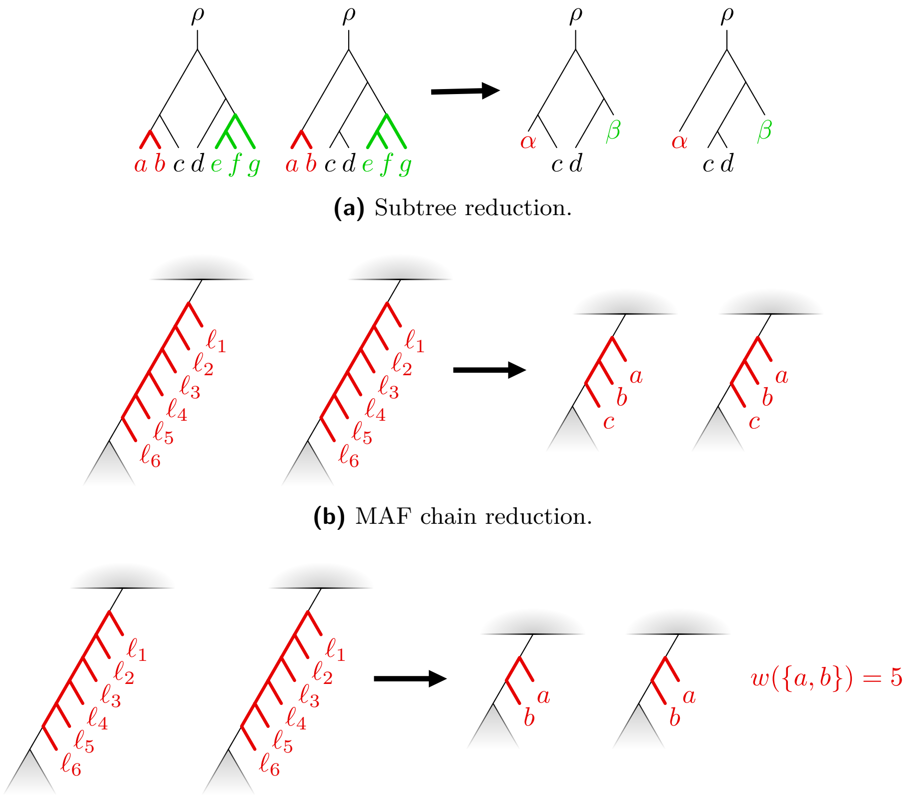
Figure 1 **(a,b)** Two binary X-trees S and T. **(c)** A multifurcating X-tree that has S as a resolution. **(d)** A sequence of SPR operations that turns S into T. **(e)** A hybridization network of (S, T). **(f)** A MAF of (S, T) that can be obtained by deleting the thin red (dashed or solid) edges in S and T. These are exactly the parent edges of the subtrees of S that are moved by SPR operations in (d). This is not an AAF, since its ancestry graph shown in **(g)** has a cycle. **(h)** A MAAF of (S, T) that can be obtained by deleting the dotted edges in S and T. Its ancestry graph shown in **(i)** is acyclic. We can also obtain this MAAF by deleting the parent edges of all hybrid nodes of the network in (e), which in turn correspond to the dotted edges in S and T.

Let $m(S, T)$ be the size (number of components) of a MAF of (S, T) and let $\tilde{m}(S, T)$ be the size of a MAAF of (S, T). As shown in [2, 7], we have $m(S, T) = 1 + d_{\text{SPR}}(S, T)$ and $\tilde{m}(S, T) = 1 + hyb(S, T)$. In fact, it is easy to convert back and forth between any (A)AF and a corresponding SPR sequence or hybridization network, as illustrated in Figures 1d,e,f,h.

Multifurcating trees usually arise due to lack of confidence in the order of speciation events derived using statistical inference methods. In order to avoid the inference of spurious reticulation events necessary only to reconcile differences in the ordering of these events in different gene trees, low-confidence edges are contracted, resulting in nodes with more than two children. Given this source of multifurcations, it is common to define the SPR distance or hybridization number of two multifurcating trees S and T to be the minimum SPR distance or hybridization number of all pairs of binary resolutions of S and T; a M(A)AF of (S, T) is the smallest M(A)AF over all pairs of binary resolutions of S and T.

3 Techniques for Computing Agreement Forests

Almost every existing algorithm for computing a M(A)AF of two trees uses a combination of four techniques: *kernelization*, *exponential search*, *depth-bounded search*, and *cluster partitioning*. In this section, we review these techniques. We discuss only binary trees here. The techniques for multifurcating trees are similar albeit more complicated.



(a) Subtree reduction.

(b) MAF chain reduction.

(c) MAAF chain reduction.

Figure 2 Kernelization rules for binary trees.

3.1 Kernelization

A *pendant subtree* of a binary X-tree T is a subtree induced by the descendants of a node in T. An m*-chain* of T is a sequence of leaves $\langle \ell_1, \ell_2, \ldots, \ell_m \rangle$ of T whose parents $p_1, p_2, \ldots, p_m$ form a directed path from p_1 to p_m in T. The kernelization algorithm for M(A)AF of binary trees uses two reduction rules, with separate chain reductions for MAF and MAAF:

Subtree reduction: Let P be a maximal common pendant subtree of S and T. Remove all nodes of P except the root from both S and T. This turns the root of P into a common leaf of S and T. Give both these leaves the same label, distinct from all labels already in X. See Figure 2a. This preserves $m(S,T)$ and $\tilde{m}(S,T)$.

Chain reduction (MAF): Replace every maximal common m-chain of S and T with $m > 3$ with a 3-chain $\langle a, b, c \rangle$ in both trees, where a, b, c are three new leaves currently not in X. See Figure 2b. This preserves $m(S,T)$.

Chain reduction (MAAF): Replace every maximal common m-chain of S and T with $m > 2$ with a 2-chain $\langle a, b \rangle$. This does *not* preserve $\tilde{m}(S,T)$, but $\tilde{m}(S,T)$ (along with a corresponding MAAF) can still be computed from the *weight* $w(F')$ of an appropriate AAF F' of the kernel (S',T'). As a basis for defining $w(F')$ below, add $\{a,b\}$ to a collection W of subsets of X and define $w(\{a,b\}) = m - 1$. See Figure 2c.

Bordewich and Semple [7] proved that subtree reduction and MAF chain reduction preserve $m(S,T)$ and produce a kernel (S',T') of size at most $28m(S,T)$. In the case of MAAF, a *legitimate* AAF F' of the kernel (S',T') is an AAF where, for every pair $\{a,b\} \in W$, either a and b are singletons (i.e., are each in their own component) or belong to the same component. Let $W_s \subseteq W$ be the subset of pairs $\{a, b\} \in W$ such that a and b are singletons in F' and let $w(F') = |F'| + \sum_{\{a,b\} \in W_s} w(\{a, b\})$. Bordewich and Semple [8] proved that subtree reduction and MAAF chain reduction produce a kernel (S', T') of size at most $14\tilde{m}(S, T)$, every legitimate AAF F' of (S', T') corresponds to an AAF of (S, T) of size $w(F')$, and one of these AAFs of (S, T) is in fact a MAAF of (S, T). Thus, it suffices to find a minimum-weight legitimate AAF of (S', T'), which is easily done by augmenting the exponential search algorithm in Section 3.2 so it ignores non-legitimate AFs.

3.2 Exponential Search

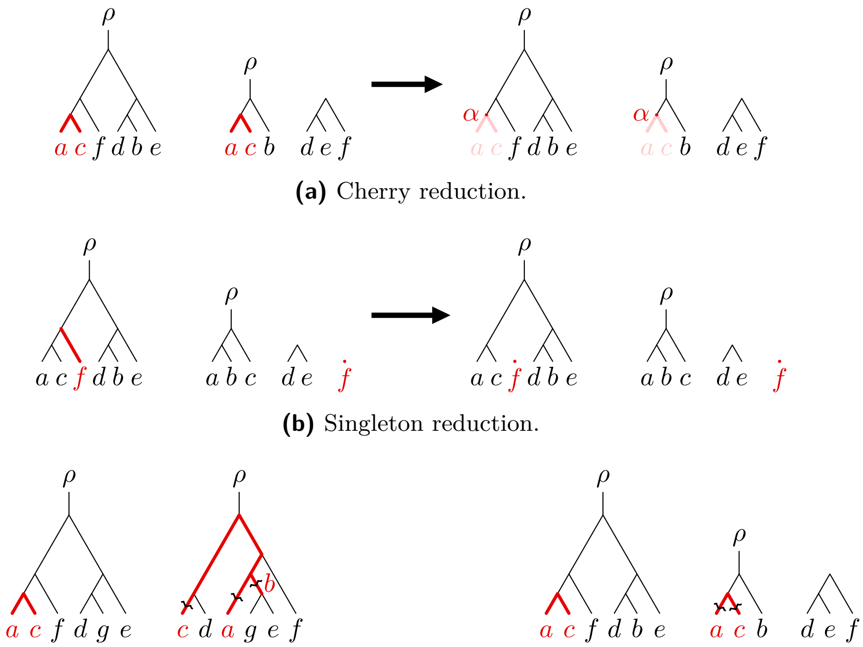
The exponential search algorithm for finding a M(A)AF of (S, T) [1] uses a recursive procedure M(A)AF(F_S, F_T, F), where F_S refines S; F_T refines T, F is an AF of (F_S, F_T), and every component of F is a pendant subtree of both F_S and F_T. MAF(F_S, F_T, F) computes a MAF of (F_S, F_T). MAAF(F_S, F_T, F) computes the smallest AAF of (S, T) that refines F_T and is refined by F; if no such AAF exists, MAAF(F_S, F_T, F) reports failure. Thus, the top-level invocation M(A)AF(S, T, F_X), where F_X has one singleton component per element in X, finds a M(A)AF of (S, T). Since M(A)AF(F_S, F_T, F) treats components of F as indivisible units, we describe the algorithm as if each of these components were replaced by a single leaf in both F_S and F_T. M(A)AF(F_S, F_T, F) first applies the following two rules:

Cherry reduction (MAF only): Let a and c be two sibling leaves of F_S, a *cherry*. If a and c are siblings also in T, then merge a and c, that is, contract them into their common parent in both F_S and F_T, and replace them with a single node in F. See Figure 3a. F remains an AF of (F_S, F_T).

Singleton reduction: If F_T has a singleton leaf that is not a singleton in F_S, then cut its parent edge in F_S. F remains an AF of (F_S, F_T). See Figure 3b.

Let F'_S, F'_T, and F' be the forests obtained once neither rule is applicable. If $F' = F'_T$ (and hence $F' = F'_S$), then F' is a MAF of (F'_S, F'_T) and, after undoing all cherry reductions, of (F_S, F_T), so MAF(F_S, F_T, F) returns F' in this case. Since MAAF(F_S, F_T, F) does not apply cherry reduction, we have $F' = F$ and $F'_T = F_T$ in MAAF(F_S, F_T, F). Thus, if $F' = F'_T$, F is the only forest that refines F_T and is refined by F. MAAF(F_S, F_T, F) checks whether F is an AAF of (S, T) and either returns F or reports failure. If $F' \neq F'_T$, then there exists a cherry (a, c) in F'_S. If (a, c) is not a cherry of F'_T and w.l.o.g. a's depth in F_T is no less than c's, then a has a sibling b in F'_T that is not an ancestor of c and any M(A)AF of (F'_S, F'_T) is a M(A)AF of $(F'_S, F'_T \mathbin{/\!\!/} \{a\})$, $(F'_S, F'_T \mathbin{/\!\!/} \{b\})$ or $(F'_S, F'_T \mathbin{/\!\!/} \{c\})$, where $F \mathbin{/\!\!/} V$ is the forest obtained from F by cutting the parent edges of all nodes in V (see e.g. [1]). See Figure 3c. Thus, M(A)AF(F_S, F_T, F) makes three recursive calls M(A)AF(F'_S, F''_T, F') where $F''_T \in \{F'_T \mathbin{/\!\!/} \{a\}, F'_T \mathbin{/\!\!/} \{b\}, F'_T \mathbin{/\!\!/} \{c\}\}$. If (a, c) *is* a cherry of F'_T, which is possible only for MAAF(F_S, F_T, F) because MAF(F_S, F_T, F) applies cherry reduction, then any AAF F'' of (S, T) that refines F'_T either refines $F_T \mathbin{/\!\!/} \{a\}$ or $F_T \mathbin{/\!\!/} \{c\}$ or (a, c) is a cherry of F''. Thus, MAAF(F_S, F_T, F) makes three recursive calls MAAF(F'_S, F''_T, F''), where either $F'' = F'$ and $F''_T = F'_T \mathbin{/\!\!/} \{a\}$ or $F''_T = F'_T \mathbin{/\!\!/} \{c\}$, or $F''_T = F'_T$ and F'' is obtained from F' by applying cherry reduction to (a, c). See Figure 3d. Since each invocation makes three recursive calls and the recursion depth can be shown to be at most n, the running time is $O(3^n n)$.

Similar ideas give an $O(4^n n)$-time algorithm for multifurcating trees (see [13, 25]). Faster algorithms for both binary and multifurcating M(A)AF are possible, either by using dynamic programming [12] or by porting some of the ideas from the currently fastest depth-bounded search algorithms [23, 25, 26] back to exponential search.



(a) Cherry reduction.

(b) Singleton reduction.

(c) If (a, c) is not a cherry of F_T, M(A)AF(F_S, F_T) makes three recursive calls, cutting the parent edges of a, b, and c in F_T in turn; b may not be a leaf of F_T and may have no corresponding node in F_S.

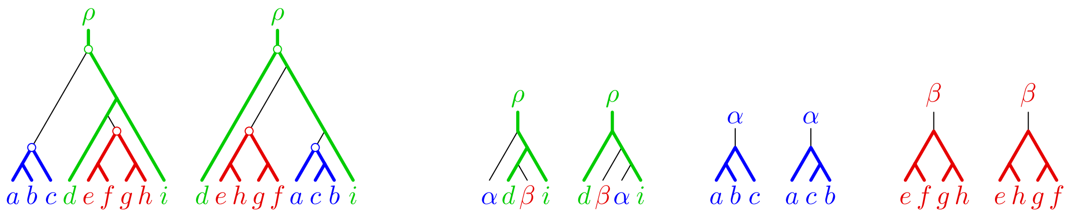
(d) If (a, c) is a cherry of F_T, MAAF(F_S, F_T) makes three recursive calls, two cutting the parent edges of a and c and one that applies the cherry reduction shown in (a).

Figure 3 Reduction and branching rules in the exponential search algorithm for binary trees.

3.3 Depth-Bounded Search

The depth-bounded search algorithm for MAF is practically identical to the exponential search algorithm. The invocation MAF(F_S, F_T, F, k) now takes an additional parameter k and decides whether there exists an AF of (F_S, F_T) of size at most k. If $|F_T| > k$, MAF(F_S, F_T, k) can immediately report failure, which limits the search depth to k because $|F_T|$ increases by at least one from one level of recursion to the next. Thus, the running time of MAF(S, T, F_X, k) is $O(3^k n)$ for binary trees and $O(4^k n)$ for multifurcating trees. A MAF can be found in $O(3^{m(S,T)} n)$ or $O(4^{m(S,T)} n)$ time, by running MAF(S, T, F_X, k) with parameter $k = 1, 2, \ldots$ until we find the first AF. This approach combined with improved branching rules and other techniques results in the currently fastest MAF algorithms, with running time $O(2^k n)$ for binary trees [23, 26] and $O(2.42^k n)$ for multifurcating trees [25].

The exponential search algorithm for MAAF cannot be translated directly into a depth-bounded search algorithm because, when (a, c) is a common cherry of F'_S and F'_T, the algorithm makes three recursive calls and the branch that applies cherry reduction cuts no edges. To obtain a depth-bounded search algorithm for MAAF, we apply the MAF algorithm (including cherry reduction!) to find a collection of AFs. It turns out that, given a parameter $k \geq \tilde{m}(S, T)$, MAF(S, T, X, k) finds an AF F that can be refined to a MAAF of (S, T) by cutting more edges [24]. Thus, to find an AAF of (S, T) of size at most k, if it exists, we run MAF(S, T, X, k) and, for each AF F it finds, check whether an AAF of (S, T) of size at most k can be obtained by cutting more edges in F. This takes $O(n) \cdot \sum_{i=0}^{k-|F|} \binom{|F|-1}{i}$ time [24]. The currently fastest hybridization algorithms for two trees use this approach and take $O(3.18^k n)$ time for binary trees [24] and $O(5.08^k n)$ time for multifurcating trees [16, 17].



(a) **(b)**

Figure 4 (a) A pair of X-trees (S, T). The highlighted nodes are shared by S and T. **(b)** A cluster partition of (S, T) corresponding to the highlighted subtrees of S and T.

3.4 Cluster Partitioning

Every node of an X-tree defines a *cluster* consisting of the labels of its descendant leaves. An X-tree is fully described by the set of clusters of its nodes, so we can view it as a set of clusters and define $\mathcal{C} = (S \cap T) \setminus \{X \cup \{\rho\}\}$ to be the set of non-root nodes shared by S and T. Each cluster $C \in \mathcal{C}$ defines two subtrees S_C and T_C of S and T consisting of the parents of C in S and T and all nodes that are subsets of C but not proper subsets of any cluster $C' \in \mathcal{C}$ with $C' \subset C$. Let $\mathcal{L} = \{\ell(C) \mid C \in \mathcal{C}\}$ be a label set disjoint from $X \cup \{\rho\}$. We label the roots of S_C and T_C with $\ell(C)$ and each leaf $C' \in \mathcal{C}$ of S_C and T_C with $\ell(C')$. The cluster partition of (S, T) is the collection of instances $\{(S_C, T_C) \mid C \in \mathcal{C}\}$; see Figure 4.

Remarkably, a MAAF of (S, T) can be obtained by computing a MAAF for each pair (S_C, T_C) with $C \in \mathcal{C}$ [3]: A MAAF F_C of each pair (S_C, T_C) can be obtained by cutting a set of edges of S_C. Cutting every edge of S that belongs to the union of these sets produces a MAAF of (S, T). A MAF of (S, T) can similarly be obtained from a collection of AFs of the clusters, but the details are more complicated [19, 27].

4 Experimental Evaluation

4.1 The Competitors

We implemented the techniques discussed in Section 3 in C++, compiled with gcc -O2, and evaluated different combinations of these techniques for finding M(A)AFs of binary and multifurcating trees. Our platform was a 2.4GHz AMD Opteron workstation with 16GB of DDR-1333 RAM running Debian GNU/Linux 7. The algorithms we evaluated were:

K: Apply kernelization and then solve the kernel using exponential search.

CP: Apply cluster partitioning and then solve each cluster using exponential search.

CP+K: Apply cluster partitioning, apply kernelization to each cluster, and then solve each cluster kernel using exponential search.

CP+DBS: Apply cluster partitioning and solve each cluster using depth-bounded search.

We also included two competing algorithms in our evaluation. Since we did not have the (source) code available, we are only able to refer to the experimental results reported by the authors or were able to run the compiled code provided by the authors:

ILP [28]: This solution expresses the problem of finding a MAAF as an integer linear program and then uses CPLEX to solve this instance. The experimental results were obtained on a 3.2GHz Intel Xeon workstation using the Poaceae data set below [28].

INTER [9]: This algorithm uses cluster partitioning and solves each cluster using kernelization and exponential search, applying kernelization in each recursive call. The authors provided a Java implementation as a JAR file.

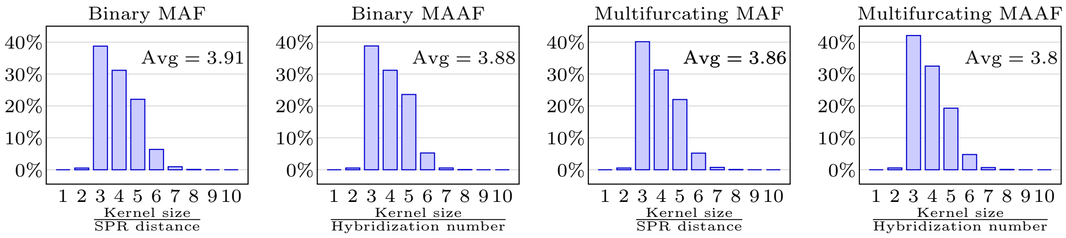


Figure 5 Kernel sizes for the Aquificae data set. The x-axis shows the ratio between kernel size and SPR distance or hybridization number, grouped into buckets where the ith bucket contains all instances with a ratio in the interval $(i-1, i]$. The y-axis shows the percentage of the inputs in each bucket.

We excluded a number of competitors from our evaluation. One is the dynamic programming algorithm of [12]. It achieves a running time of $O(2^n \text{poly}(n))$ but uses exponential space, which is prohibitive. Faster depth-bounded search algorithms with running times of $\mathrm{O}(2^k n)$ for binary trees [23, 26] and $\mathrm{O}(2.42^k n)$ for multifurcating trees [25] exist and translate into corresponding improvements for exponential search. However, both algorithms are difficult to implement. An implementation of the $\mathrm{O}(2^k n)$-time algorithm for binary trees exists [27], while the $\mathrm{O}(2.42^k n)$-time algorithm for multifurcating trees has not been implemented yet. In order to avoid performance differences due only to differences in the implementation, we chose to implement all competitors (except ILP and INTER) ourselves and opted for the simpler algorithms discussed in Section 3. Since any improvement applicable to depth-bounded search is applicable to exponential search and vice versa, the qualitative conclusions of our results apply also to faster branching algorithms.

4.2 Data Sets

Aquificae. This data set was provided by Robert Beiko [4] and contained gene trees of the phylum Aquificae, which is generally believed to have a high rate of reticulation events in its history. The input trees were unrooted. A rooting was obtained by Chris Whidden [27], who used a subset of 40,463 of these trees over a set of 1,251 taxa, computing an MRP supertree and rooting the gene trees to match the MRP supertree. Each tree had between 4 and 74 taxa. Comparisons between pairs of trees were made only on the subtrees induced by their common taxa. The original trees were binary. Multifurcating versions were obtained by collapsing bipartitions with support below 0.8. We carried out pairwise comparisons between all pairs of these 40,463 trees. Our experimental evaluation excluded all pairs with SPR distance 0, leaving us with roughly 170,000 non-trivial input pairs.

Poaceae. This data set was provided by Heiko Schmidt [21], who constructed rooted binary trees from the sequence data of six loci (ITS, ndhF, phyB, rbcL, rpoC2, and waxy) provided by the Grass Phylogeny Working Group [10]. The resulting data set contained 15 tree pairs. We used this data set in our final experiment that included ILP and INTER because we did not have the code of ILP available and the authors of [28] reported results on this data set.

4.3 Results

Kernel size. The first question we wanted to answer was: How much smaller than the theoretical prediction are the kernels produced by the kernelization algorithms in practice? Figure 5 shows the kernel sizes observed for the Aquificae data set in our experiments.

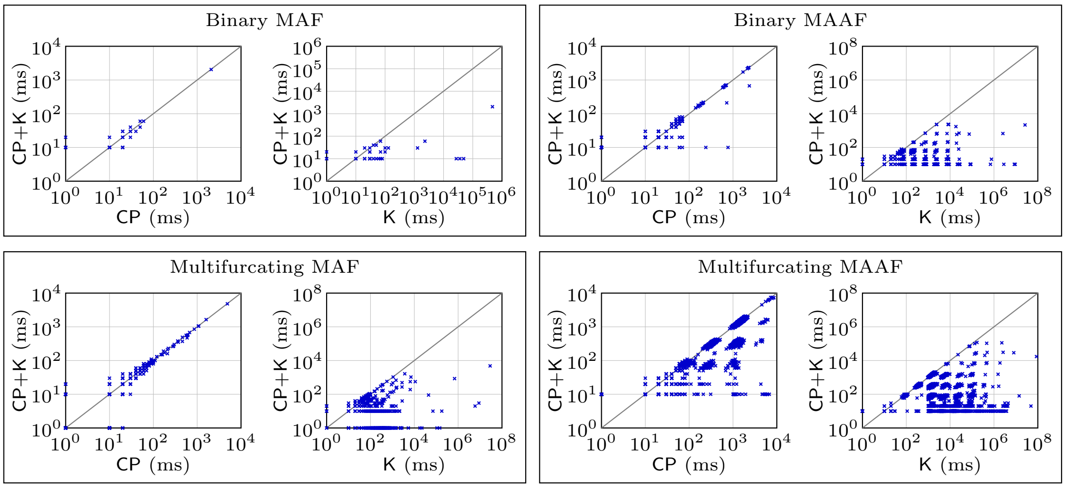


Figure 6 Running times of CP, K, and CP+K on the Aquificae data set. Each instance is represented as a point with the running time of CP or K as the x-coordinate and the running time of CP+K as the y-coordinate. Points below the diagonal indicate that CP+K performed better.

The average kernel size was less than $4k$ for both MAF and MAAF and for binary and multifurcating trees, and the kernel sizes were quite tightly concentrated in the range $[3, 5]$.

Kernelization vs. cluster partitioning. Next we aimed to quantify the relative impact of cluster partitioning and kernelization on the performance of algorithms that combine these techniques. Figure 6 compares the running times of K, CP, and CP+K on a subset of the Aquificae data set. For binary M(A)AF and multifurcating MAF, we removed trivial instances that all three methods were able to solve in less than 1ms. We also removed all instances that K was not able to solve in 8 hours, even though some of these instances could be solved by CP+K in a reasonable time. Multifurcating MAAF is a much harder problem, so even the easy instances took up to 1s to solve and took roughly the same amount of time to solve with any of the three methods. Thus, for multifurcating MAAF, we removed all instances that could be solved in less than 1s or for which K took more than 8 hours. This left 6,800 binary MAF instances, 25,000 binary MAAF instances, 40,000 multifurcating MAF instances, and 5,300 multifurcating MAAF instances. The right-hand figures in the four panels show that adding cluster partitioning to K led to significant performance improvements for almost all instances. For binary M(A)AF and multifurcating MAF, adding kernelization to CP led to only very modest performance improvements. Moreover, there were about as many instances where the overhead of kernelization hurt performance as there were instances where performance improved. The exception is multifurcating MAAF, where adding kernelization to CP led to more significant performance improvements in many instances.

Another useful comparison can be obtained by summing the running times of each algorithm across all test instances, as this amplifies performance gains made on the difficult instances that took a long time to solve. For all but multifurcating MAAF, kernelization did not help and in fact increased the total running time of CP by a factor of almost 5 in the case of binary MAF. For multifurcating MAAF, a modest speed-up by a factor of 2.2 was achieved. In contrast, the performance improvements achieved by adding cluster partitioning to K ranged from 8.5 for binary SPR to 160.8 for multifurcating SPR, again demonstrating the effectiveness of cluster partitioning. The results for binary SPR are to be treated with caution. Almost all binary SPR instances were solved by CP and CP+K

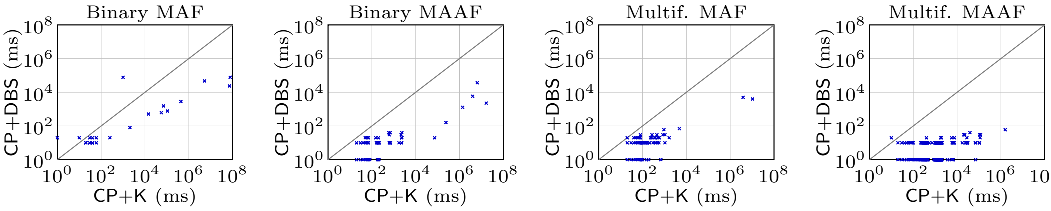


Figure 7 Running times of CP+K and CP+DBS on the Aquificae data set. Each instance is represented as a point with the running time of CP+K as the x-coordinate and the running time of CP+DBS as the y-coordinate. Points below the diagonal indicate that CP+DBS performed better.

in less than 100ms and in less than 10s even using only K. On such "easy" instances, the overhead of any added optimization is fairly large compared to the achievable gain, which we believe explains the detrimental impact of kernelization on the performance of CP and the only modest improvement of performance when adding cluster partitioning to K.

Impact of kernelization on maximal cluster size. Since the running time of an exponential search algorithm across multiple clusters is dominated by the running time on the largest cluster, the lack of impact of kernelization on the performance of CP can be explained by investigating the decrease of the maximal cluster size for each instance due to kernelization. In our experiments, no decrease was achieved in over 99.8% of the inputs for binary M(A)AF and multifurcating MAF, which correlates with our running time comparisons above. For multifurcating MAAF, no reduction was achieved for 93.8% of the inputs while about 6% of the inputs achieved a reduction of the maximal cluster size by 10-30%. While we expected the impact of kernelization on the cluster size to be modest, we were surprised to see that the vast majority of instances did not see *any* decrease in the maximal cluster size.

Kernelization vs depth-bounded search. The next question we aimed to answer was which algorithm to choose to solve individual clusters. Our first experiment with this goal compared CP+K vs CP+DBS. Figure 7 shows that CP+DBS was significantly faster than CP+K. We excluded trivial instances that both methods were able to solve in less than 10ms as well as instances that took CP+K more than 8 hours to solve from the evaluation, even though CP+DBS was able to solve all instances in a reasonable time. A comparison of the total running times of these two methods across all inputs shows that overall CP+DBS was between 135 times (for multifurcating MAAF) and 1,000 times (for binary MAF) faster than CP+K. Since kernelization of the individual clusters was largely ineffective, this is not surprising.

Which is the fastest MAAF algorithm? Since we did not have the source code of ILP and the results in [28] were reported on the Poaceae data set, we chose this data set for a horse race between all the competitors listed in Section 4.1. INTER only computes binary MAAFs, so this was the only type of MAAF we computed in our experiments. As a result, this evaluation is fairly limited. Table 1 shows the results. We report two running times for INTER. The first (A) was obtained on inputs where string labels were replaced with integer labels; the second (B) was obtained using the Poaceae data files bundled with the code of [9], where every leaf was labelled with the name of the taxon. Apart from that, the inputs were identical. We do not know why this change would have such a significant impact on the running time of the implementation. Table 1 shows that ILP, INTER, and CP+DBS each achieved the fastest running time on at least one input and were significantly faster

Table 1 Running times of the different algorithms for computing MAAFs of the instances in the Poaceae data set. For each input consisting of "Tree 1" and "Tree 2", we list its number of taxa (n) and hybridization number (k).

Input				ILP	INTER		CP+	CP	K	CP+K
Tree 1	Tree 2	n	k		A	B	DBS			
ndhF	phyB	40	14	5s	20s	14s	13s	210s	>3h	206s
ndhF	rbcL	36	13	10s	3s	3s	16s	220s	>3h	218s
ndhF	rpoC2	34	12	7s	6s	3s	10s	559s	>3h	563s
ndhF	waxy	19	9	1s	<1s	1s	<1s	2s	56s	2s
ndhF	**ITS**	**46**	**19**	**51s**	**255s**	**1197s**	**78s**	**>3h**	**>3h**	**>3h**
phyB	rbcL	21	4	<1s	<1s	<1s	<1s	1s	6s	1s
phyB	rpoC2	21	7	3s	<1s	<1s	<1s	2s	222s	2s
phyB	waxy	14	3	1s	<1s	<1s	<1s	1s	1s	1s
phyB	ITS	30	8	1s	<1s	<1s	<1s	1s	>3h	1s
rbcL	rpoC2	26	13	14s	7s	5s	32s	662s	>3h	619s
rbcL	waxy	12	7	1s	<1s	<1s	<1s	2s	2s	2s
rbcL	**ITS**	**29**	**14**	**80s**	**586s**	**1979s**	**49s**	**>3h**	**>3h**	**>3h**
rpoC2	waxy	10	1	<1s	<1s	<1s	<1s	1s	1s	1s
rpoC2	ITS	31	15	115s	53s	1650s	17s	>3h	>3h	>3h
waxy	ITS	15	8	1s	<1s	<1s	<1s	6s	67s	6s

than K, CP, and CP+K. ILP and CP+DBS substantially outperformed INTER on the hardest inputs (ndhF/ITS and rbcL/ITS), highlighted in bold. These inputs have among the highest hybridization numbers in this data set, suggesting that INTER cannot keep up with ILP and CP+DBS as the hybridization number increases.

5 Conclusions

We investigated the impact of cluster partitioning on the performance of kernelization-based M(A)AF algorithms. Together with results for depth-bounded search reported in [27], our results support the following conclusions: (i) Cluster partitioning is by far the most important tool for obtaining fast M(A)AF algorithms. (ii) When used in conjunction with cluster partitioning, kernelization offers very little benefit and may even hurt performance due to the cost of computing the kernel. (iii) Depth-bounded search offers superior performance over kernelization. The exception is an approach that re-kernelizes the input after each branching step in the exponential search (INTER). Depth-bounded search and INTER have much in common in that the cherry and singleton reductions can be viewed as partially applying kernelization until it is safe to apply the next branching step.

Given the importance of cluster partitioning for the performance of M(A)AF algorithms, an important question is whether cluster partitioning can be improved further. When an input or large cluster cannot be split into smaller clusters, "long-distance" reticulations between distant taxa are often to blame. Empirical evidence suggests that most reticulations happen between fairly closely related taxa, so long-distance reticulations should be rare. If there exists an efficient algorithm for finding these long-distance reticulations, they could be eliminated, resulting in a modified input with only local reticulations that can therefore be split into small clusters for which M(A)AFs can be found efficiently. This would likely allow us to find M(A)AFs for larger and harder inputs currently well beyond our reach.

References

1 Benjamin Albrecht, Céline Scornavacca, Alberto Cenci, and Daniel H. Huson. Fast computation of minimum hybridization networks. *Bioinformatics*, 28(2):191–197, 2012.

2 M. Baroni, S. Grünewald, V. Moulton, and C. Semple. Bounding the number of hybridisation events for a consistent evolutionary history. *Journal of Mathematical Biology*, 51(2):171–182, 2005.

3 M. Baroni, C. Semple, and M. Steel. Hybrids in real time. *Systematic Biology*, 55:46–56, 2006.

4 Robert G. Beiko. Telling the whole story in a 10,000-genome world. *Biology Direct*, 6(1):34, 2011.

5 M. Bordewich and C. Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.

6 Magnus Bordewich, Céline Scornavacca, Nihan Tokac, and Mathias Weller. On the fixed parameter tractability of agreement-based phylogenetic distances. *Journal of Mathematical Biology*, 74(1):239–257, 2017.

7 Magnus Bordewich and Charles Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8(4):409–423, 2005.

8 Magnus Bordewich and Charles Semple. Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(3):458–466, 2007.

9 Joshua Collins, Simone Linz, and Charles Semple. Quantifying hybridization in realistic time. *Journal of Computational Biology*, 18(10):1305–1318, 2011.

10 Grass Phylogeny Working Group. Phylogeny and subfamilial classification of the grasses (poaceae). *Annals of the Missouri Botanical Garden*, page 373–457, 2001.

11 D. M. Hillis, C. Moritz, and B. K. Mable, editors. *Molecular Systematics*. Sinauer Associates, 1996.

12 Leo van Iersel, Steven Kelk, Nela Lekić, and Leen Stougie. A short note on exponential-time algorithms for hybridization number. *CoRR*, abs/1312.1255, 2013.

13 Leo van Iersel, Steven Kelk, Nela Lekić, and Leen Stougie. Approximation algorithms for nonbinary agreement forests. *SIAM Journal on Discrete Mathematics*, 28(1):49–66, 2014.

14 Leo van Iersel and Simone Linz. A quadratic kernel for computing the hybridization number of multiple trees. *Information Processing Letters*, 113(9):318–323, 2013.

15 Steven Kelk and Céline Scornavacca. Towards the fixed parameter tractability of constructing minimal phylogenetic networks from arbitrary sets of nonbinary trees. *CoRR*, abs/1207.7034, 2012.

16 Zhijiang Li. Fixed-parameter algorithm for hybridization number of two multifurcating trees. Master's thesis, Faculty of Computer Science, Dalhousie University, 2015.

17 Zhijiang Li and Norbert Zeh. A fast algorithm for computing a soft hybridization network of two multifurcating trees. Manuscript in preparation.

18 Simone Linz and Charles Semple. Hybridization in nonbinary trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(1):30–45, 2009.

19 Simone Linz and Charles Semple. A cluster reduction for computing the subtree distance between phylogenies. *Annals of Combinatorics*, 15(3):465–484, 2011.

20 V. Rosas-Magallanes, P. Deschavanne, L. Qintana-Murci, R. Brosch, B. Gicquel, and O. Neyrolles. Horizontal transfer of a virulence operon to the ancestor of mycobacterium tuberculosis. *Molecular Biology and Evolution*, 23(6):1129–1135, 2006.

21 Heiko Schmidt. *Phylogenetic Trees from Large Datasets*. PhD thesis, Heinrich-Heine-Universität, Düsseldorf, Germany, 2003.

22 Feng Shi, Jie You, and Qilong Feng. Improved approximation algorithm for maximum agreement forest of two trees. In *Proceedings of the 8th International Workshop on Frontiers in Algorithmics*, pages 205–215, 2014.

23 Chris Whidden. *Efficient Computation of Maximum Agreement Forests and Their Applications*. PhD thesis, Faculty of Computer Science, Dalhousie University, 2013.

24 Chris Whidden, Robert G. Beiko, and Norbert Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013.

25 Chris Whidden, Robert G. Beiko, and Norbert Zeh. Fixed-parameter and approximation algorithms for maximum agreement forests of multifurcating trees. *Algorithmica*, 74(3):1019–1054, 2016.

26 Chris Whidden and Norbert Zeh. Computing the SPR distance of rooted binary trees in $O(2^k n)$ time. Manuscript in preparation.

27 Chris Whidden, Norbert Zeh, and Robert G. Beiko. Supertrees based on the subtree prune-and-regraft distance. *Systematic Biology*, 63(4):566–581, 2014.

28 Yufeng Wu and Jiayin Wang. Fast computation of the exact hybridization number of two phylogenetic trees. In *Proceedings of the 6th International Symposium on Bioinformatics Research and Applications*, pages 203–214. Springer-Verlag, 2010.