Computing a consensus of multilabeled trees

Katharina T. Huber* Vincent Moulton* Andreas Spillner† Sabine Storandt‡
Radosław Suchecki*

Abstract

In this paper we consider two challenging problems that arise in the context of computing a consensus of a collection of multilabeled trees, namely (1) selecting a compatible collection of clusters on a multiset from an ordered list of such clusters and (2) optimally refining high degree vertices in a multilabeled tree. Forming such a consensus is part of an approach to reconstruct the evolutionary history of a set of species for which events such as genome duplication and hybridization have occurred in the past. We present exact algorithms for solving (1) and (2) that have an exponential runtime in the worst case. To give some impression of their performance in practice, we apply them to simulated input and to a real biological data set highlighting the impact of several structural properties of the input on the performance.

1 Introduction

In this paper we develop exact algorithms for two NP-hard problems involving so-called multilabeled trees or MUL-trees, for short. These are rooted trees $\mathcal{T}$ whose leaves are labeled with the elements of a finite set X where an element $x \in X$ may label more than one leaf of $\mathcal{T}$. Since these problems admit polynomial time algorithms in case every label is allowed to occur at most once, we identify parameters that seem to capture quite well how hard it is to deal with a particular input. We focus here on two particular problems that arise in biology in the context of an approach to reconstruct the evolutionary history of a set X of species in which events such as genome duplication and hybridization have occurred (see e.g. [9, 10]). These parameters and the structural properties of MUL-trees employed in our new algorithms also appear to have some promise for tackling other problems involving such trees.

To reconstruct the evolutionary history for a set X of species, one can first reconstruct a collection $\mathbf{T}$ of MUL-trees, each describing the evolutionary history of a single gene sequenced for each species in X and then form a consensus of the trees in $\mathbf{T}$ [12]. The consensus will again be a MUL-tree that can then be turned into a network describing the evolutionary history more explicitly. Note that the trees in $\mathbf{T}$ are multilabeled since, due to genome duplication or hybridization events in the past, several copies of the same gene might be present in the genome of a species in X and some of these copies might have evolved along different evolutionary paths.

The method for computing a consensus of $\mathbf{T}$ presented in [8] works by first breaking the trees in $\mathbf{T}$ into subsets of a multiset, so-called clusters. These clusters are ranked in an ordered list $\mathcal{L}$ according to how often a particular cluster occurs in the trees in $\mathbf{T}$. Then, to form a consensus MUL-tree, the following two problems need to be solved:

(1) Given the ordered list $\mathcal{L}$ of clusters on a multiset, select a suitable subcollection $\mathcal{C}$ of clusters that is compatible, that is, can be represented by a single MUL-tree.

(2) Compute a MUL-tree which induces a supercollection of the clusters in $\mathcal{C}$ that minimizes the number of genome duplication and hybridization events needed to explain that tree.

Note that, in view of the fact that it is an NP-hard problem to decide whether a given unordered collection of clusters on a multiset is compatible [4], it follows that (1) is NP-hard and, using a similar reduction as employed in [4], one can show that (2) also leads to an NP-hard subproblem. As mentioned above, we present exact algorithms for solving (1) and (2) and demonstrate, using simulated inputs and a biological data set that, with the help of these algorithms, the approach presented in [8] is feasible for computing the consensus of collections of MUL-trees each having several hundred leaves.

*University of East Anglia, School of Computing Sciences, Norwich, NR4 7TJ, UK; katharina.huber@cmp.uea.ac.uk, vincent.moulton@cmp.uea.ac.uk, r.suchecki@uea.ac.uk

†Universität Greifswald, Institut für Mathematik und Informatik, 17487 Greifswald, Germany; andreas.spillner@uni-greifswald.de

‡Universität Stuttgart, Institut für Formale Methoden der Informatik, 70569 Stuttgart, Germany; sabine.storandt@fmi.uni-stuttgart.de

Related work includes algorithms for comparing MUL-trees [1], for computing MUL-trees that optimally represent collections of so-called triplets [2], and for optimally pruning MUL-trees to obtain a usual phylogenetic tree [11], that is, a rooted tree in which the leaves are in one-to-one correspondence with the underlying set X.

2 Preliminaries

2.1 Basic definitions We view a multiset M as a map $M : X \to \mathbb{N}_{>0}$ for some finite set X and call $M(x)$ the *multiplicity* of $x \in X$. The set X is called the *underlying set* of M and denoted by $\underline{M}$. We will also write $x \in M$ if $x \in X$ holds and, to describe specific multisets, we will just list the elements with their multiplicities, e.g. $\{a, a, a, b, b, b, c\}$. In addition, for any multiset M, we define the *size* $|M| = \sum_{x \in X} M(x)$, the *thin part* $M^* = \{x \in X : M(x) = 1\}$, and the *deviation* $\Delta(M) = \sum_{x \in X}(M(x) - 1)$ of M from its underlying set X. A multiset M' with $\underline{M'} \subseteq \underline{M}$ and $M'(x) \leq M(x)$ for all $x \in \underline{M'}$ is called a *submultiset* of M and we use $M' \subseteq M$ to denote this fact. Non-empty submultisets of M are also referred to as *clusters* on M. A cluster is *trivial* if it has size 1. The union $M \cup M'$ of two multisets M and M' with the same underlying set X contains every $x \in X$ with multiplicity $M(x) + M'(x)$. Similarly, the difference $M - M'$ contains every $x \in X$ with multiplicity $\max\{0, M(x) - M'(x)\}$.

A MUL-tree $\mathcal{T} = (T, \varphi)$ on a multiset M consists of (i) a rooted tree $T = (V, E, \rho)$ with root ρ in which every vertex has either none or at least two children, and (ii) a *labeling map* $\varphi : L(T) \to \underline{M}$ from the set $L(T)$ of leaves of T onto $\underline{M}$ such that, for every $x \in \underline{M}$, $|\{v \in L(T) : \varphi(v) = x\}| = M(x)$ holds (cf. Figure 1(a)). We say that $\mathcal{T}$ is *binary* if every non-leaf vertex of $\mathcal{T}$ has precisely 2 children. Two MUL-trees $\mathcal{T}_1 = (T_1, \varphi_1)$ and $\mathcal{T}_2 = (T_2, \varphi_2)$ are *isomorphic* if there exists a graph isomorphism ι from T_1 to T_2 such that $\varphi_1(v) = \varphi_2(\iota(v))$ holds for all $v \in L(T_1)$ and, in addition, the root of T_1 is mapped to the root of T_2.

For every vertex v of a MUL-tree $\mathcal{T}$ on M, we denote by $\mathcal{T}_v$ the rooted subtree of $\mathcal{T}$ consisting of those vertices u of $\mathcal{T}$ for which the path from u to ρ contains v. The cluster C_v *induced* by v is the submultiset of M formed by the labels of the leaves of $\mathcal{T}_v$ (cf. Figure 1(b)). In addition, let $\mathcal{C}(\mathcal{T})$ denote the collection of clusters induced by the vertices of $\mathcal{T}$ with multiplicities taken into account, that is, $\mathcal{C}(\mathcal{T})$ is also considered as a multiset. Note that, for every $x \in \underline{M}$, $\mathcal{C}(\mathcal{T})$ contains the trivial cluster $\{x\}$ with multiplicity $M(x)$. In the following we will mostly consider collections of clusters that have this property and we will say that a collection $\mathcal{C}$ of clusters on a multiset M is *compatible* if there exists a MUL-tree $\mathcal{T}$ on M with $\mathcal{C} \subseteq \mathcal{C}(\mathcal{T})$. Otherwise we say that $\mathcal{C}$ is *incompatible*.

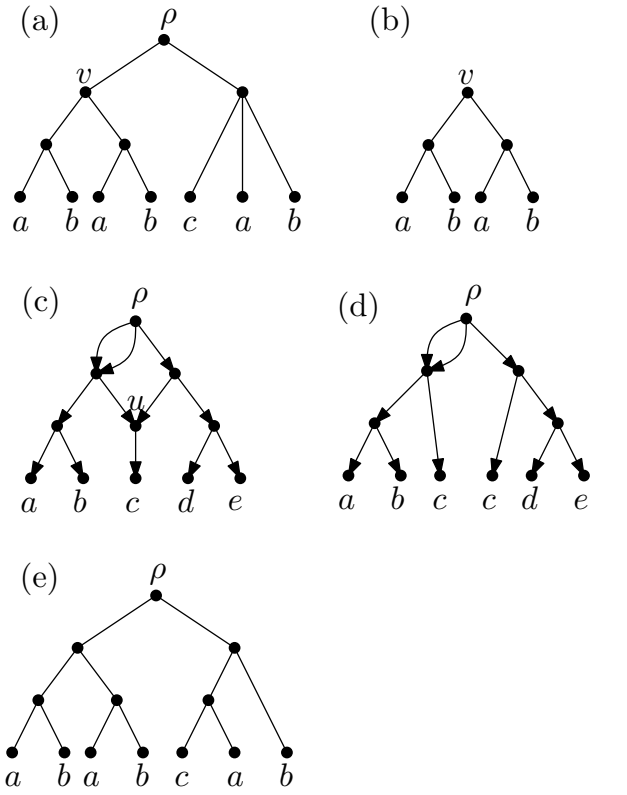

Figure 1: (a) A MUL-tree $\mathcal{T}$ on $\{a, a, a, b, b, b, c\}$. (b) The subtree $\mathcal{T}_v$. The cluster induced by v is $C_v = \{a, a, b, b\}$ and the multiplicity of cluster $\{a, b\}$ in $\mathcal{C}(\mathcal{T}_v)$ (as well as in $\mathcal{C}(\mathcal{T})$) is 2. (c) A phylogenetic network $\mathcal{N}$ on $\{a, b, c, d, e\}$ with $ret(\mathcal{N}) = 2$. (d) The network $\mathcal{N}_i$ resulting from processing vertex $v_i := u$ in the network $\mathcal{N}_{i-1} = \mathcal{N}$ in (c). (e) A refinement of the MUL-tree in (a).

A *phylogenetic network* $\mathcal{N} = (D, \varphi)$ on a set X consists of a directed acyclic graph $D = (V, E)$ (parallel edges allowed) and a labeling map $\varphi : L(D) \to X$ from the set $L(D)$ of leaves of D, that is, those vertices with no outgoing edge, onto X (cf. Figure 1(c)). We require that D has precisely one vertex ρ with no incoming edges, called the *root* of $\mathcal{N}$. Moreover, every vertex in D has either (i) no outgoing edge and precisely one incoming edge, or (ii) at least two outgoing edges, or (iii) at least two incoming edges. Note that there are several other definitions of phylogenetic networks in the literature, see e.g. [6]. The *reticulation number* of $\mathcal{N}$ is defined as $ret(\mathcal{N}) = \sum_{v \in V - \{\rho\}}(deg_{in}(v) - 1)$ where $deg_{in}(v)$ denotes the number of incoming edges at vertex v. Note that this number can be viewed as an estimate of how many genome duplication or hybridization events

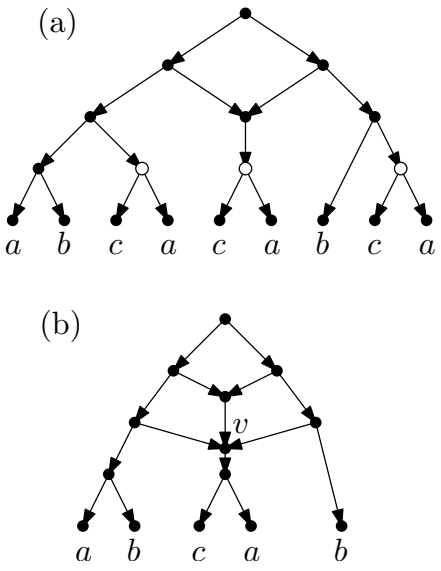

Figure 2: An Example illustrating one step in the algorithm for constructing the network $\mathcal{N}(\mathcal{T})$ for a given MUL-tree $\mathcal{T}$. (a) The network $\mathcal{N}_{i-1}$ with the vertices in a suitable set U marked by empty circles. (b) The resulting network $\mathcal{N}_i$.

have occurred in the evolutionary history of the set of species X represented by $\mathcal{N}$. Therefore, as described next, we use the reticulation number to score a given MUL-tree.

In the remainder of the paper we will use the following constructions to go from a phylogenetic network to a MUL-tree and vice versa. First note that with every phylogenetic network $\mathcal{N}$ on X one can associate a canonical MUL-tree $\mathcal{T}(\mathcal{N})$ on a multiset M with $\underline{M} = X$ as follows. Let $v_1, v_2, \ldots, v_\ell$ be a topological ordering of the vertices of $\mathcal{N}$. We construct a sequence of phylogenetic networks $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_\ell$ with $\mathcal{N}_0 = \mathcal{N}$. Assume we have already constructed $\mathcal{N}_{i-1}$ for some $i \in \{1, 2, \ldots, \ell\}$. If v_i has at most one incoming edge we put $\mathcal{N}_i = \mathcal{N}_{i-1}$. Otherwise we make $deg_{in}(v_i)$ copies of the subgraph induced by the set of vertices in $\mathcal{N}_{i-1}$ that can be reached on a directed path from v_i and attach one separate copy with each incoming edge at v_i, suppressing any resulting vertices with only one incoming and one outgoing edge (cf. Figure 1(d)). It is not hard to see that, discarding the directions of the edges, $\mathcal{N}_\ell$ is indeed a MUL-tree on some multiset M with $\underline{M} = X$ and it does not depend on the chosen topological ordering. In this case we will say that the phylogenetic network $\mathcal{N}$ *displays* the MUL-tree $\mathcal{T}(\mathcal{N})$.

In [5] a polynomial time algorithm is presented that can be viewed as reversing the construction of $\mathcal{T}(\mathcal{N})$ yielding, for any MUL-tree $\mathcal{T}$ on some multiset M, a phylogenetic network $\mathcal{N}(\mathcal{T})$ on $\underline{M}$ with $\mathcal{T} = \mathcal{T}(\mathcal{N}(\mathcal{T}))$. It should be noted, however, that there exist phylogenetic networks $\mathcal{N}$ with $\mathcal{N} \neq \mathcal{N}(\mathcal{T}(\mathcal{N}))$. The algorithm in [5] again constructs a sequence $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_\ell$ of phylogenetic networks, where $\mathcal{N}_0$ is the given MUL-tree $\mathcal{T}$ with all edges directed away from the root and $\mathcal{N}_\ell$ is the resulting network $\mathcal{N}(\mathcal{T})$. In [5] it is shown that, under relatively mild assumptions, $\mathcal{N}(\mathcal{T})$ has the minimum reticulation number among all phylogenetic networks that display $\mathcal{T}$. As mentioned above, this suggests using this number to score $\mathcal{T}$.

We close this section by recalling some facts about the construction of $\mathcal{N}_i$ from $\mathcal{N}_{i-1}$, $i \in \{1, 2, \ldots, \ell\}$, that will be used in the description of our algorithm for Problem (2) later on. First a set U of at least two vertices in $\mathcal{N}_{i-1}$ is selected such that (i) for every vertex $u \in U$, the subnetwork of $\mathcal{N}_{i-1}$ formed by those vertices that can be reached from u along directed paths does not contain any vertices with more than one incoming edge, implying that this subnetwork is actually a MUL-tree $\mathcal{T}_u$, (ii) the MUL-trees $\mathcal{T}_u$, $u \in U$, are pairwise isomorphic, (iii) every vertex w of $\mathcal{N}_{i-1}$ with $\mathcal{T}_w$ isomorphic to $\mathcal{T}_u$ for some $u \in U$ is contained in U, and (iv) the size of the cluster C_u induced by the vertices $u \in U$ is maximum among all U satisfying (i)-(iii). The algorithm then replaces the collection of trees $\mathcal{T}_u$, $u \in U$, by a single copy of such a tree that is attached to a vertex v with $|U|$ incoming edges, each representing one of the original trees (cf. Figure 2).

2.2 Random MUL-tree generator and used hardware/software To generate simulated input for evaluating our algorithms, we used the following approach. For given values m, m^* and δ, random MUL-trees are generated as follows: First a random binary rooted tree T with m leaves is generated according to the Yule-Harding model [3, 13]. Then a random multiset M with $|M| = m$, $|M^*| = m^*$ and $\Delta(M) = \delta$ is generated and, from all bijections between M and $L(T)$, one is selected uniformly at random to obtain a labeling map φ. The multiset M is generated by first creating m^* elements, each with multiplicity one, that constitute M^*. Then, putting $m' := m - m^* - \delta$, additional elements $x_1, x_2, \ldots, x_{m'}$ are generated in that order, assigning x_i, $i = 1, 2, \ldots, m' - 1$, a random multiplicity $M(x_i)$ that is selected uniformly at random in the range from 2 to $2 + m - m^* - 2m' - \sum_{j=1}^{i-1}(M(x_j) - 2)$. Finally, element $x_{m'}$ is assigned multiplicity $2 + m - m^* - 2m' - \sum_{j=1}^{m'-1}(M(x_j) - 2)$. Note that the values $|M|$, $|M^*|$ and $\Delta(M)$ are not completely independent. For example, as $|M^*|$ tends to $|M|$, $\Delta(M)$ tends to 0. Therefore, in our experiments we will usually mention $|M|$ and $\Delta(M)$ only. All computational experiments presented in this

paper were performed on an Intel Harpertown dual quad core system with Scientific Linux 5 operating system using 2GB of main memory.

2.3 Brief description of the overall approach

We briefly recall the key facts about the method described in [8]: We are given a collection $\mathbf{T}$ of MUL-trees, all on the same multiset M. These MUL-trees are turned into a list $\mathcal{L}$ of clusters on M sorted according to how often a particular cluster is induced by the trees in the input collection. We only take clusters into account that are induced by at least a certain percentage of the MUL-trees in $\mathbf{T}$. This percentage can be chosen by the user. In our computational experiments we used 50%.

From $\mathcal{L}$ we want to select a compatible collection of clusters by running through $\mathcal{L}$. This is Problem (1) mentioned in the introduction. At step i we have a collection $\mathcal{C}_i$ of compatible clusters already selected from $\mathcal{L}$. Then, for the next cluster C in $\mathcal{L}$, we check whether $\mathcal{C}_i \cup C$ is compatible. If this is the case we add C to $\mathcal{C}_i$ to obtain $\mathcal{C}_{i+1}$. Otherwise we put $\mathcal{C}_{i+1} = \mathcal{C}_i$. The resulting compatible collection of clusters is denoted by $\mathcal{C}$.

Next we search for a suitable MUL-tree $\mathcal{T}$ such that $\mathcal{C} \subseteq \mathcal{C}(\mathcal{T})$ holds. This is Problem (2) mentioned in the introduction. Note that there can be a superpolynomial number of such trees. To select a biologically meaningful candidate from these trees, a *score* $\sigma(\mathcal{T})$ is associated with each MUL-tree $\mathcal{T}$. This score is defined as the minimum of the reticulation number over all networks $\mathcal{N}(\mathcal{T}')$ where $\mathcal{T}'$ is a binary MUL-tree that *refines* $\mathcal{T}$ (cf. Figure 1(e)), that is, there exists a sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ of MUL-trees such that:

(i) $\mathcal{T}_1 = \mathcal{T}$ and $\mathcal{T}_k = \mathcal{T}'$,

(ii) For all $i \in \{1, 2, \ldots, k-1\}$, the MUL-tree $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by adding a new vertex v, connecting v by a new edge to some vertex u of $\mathcal{T}_i$ with at least three children, deleting the edges between u and at least two (but not all!) of its children, and then connecting these children, each by a new edge, with vertex v so that they become children of v in $\mathcal{T}_{i+1}$.

In the following we will refer to the process described in (ii) also as a *single refinement step*, or, in case we need to be more precise, the *refinement of vertex* u by vertex v.

3 Problem (1) — Selecting a compatible collection of clusters

Our approach is based on the fixed-parameter algorithm described in [4] to decide whether a given collection of clusters on a multiset is compatible. We use this algorithm to check, at each step i, whether for the next cluster C in the list $\mathcal{L}$ the collection $\mathcal{C}_i \cup C$ is compatible. In this process, two dynamic programming tables are used. In the first one, denoted by $\mathbf{DP}_1$, the rows are indexed by the submultisets $M' \subseteq M$ and the columns are indexed by the subcollections $\mathcal{C}' \subseteq \mathcal{C}$. The entry in $\mathbf{DP}_1$ for some M' and $\mathcal{C}'$ is either 0 or 1, where a 1 means that there exists a MUL-tree $\mathcal{T}'$ on M' with $\mathcal{C}' \subseteq \mathcal{C}(\mathcal{T}')$ and 0 means that there is no such MUL-tree $\mathcal{T}'$. The computation of an entry in $\mathbf{DP}_1$ involves the look-up of other entries in $\mathbf{DP}_1$ but does not use the second table we describe next.

The second table, denoted by $\mathbf{DP}_2$, has a row for certain collections $\mathbf{M} = \{M_1, M_2, \ldots, M_s\}$ of submultisets of M, and a column for each subcollection $\mathcal{C}' \subseteq \mathcal{C}$. Again, the entry in $\mathbf{DP}_2$ for given $\mathbf{M}$ and $\mathcal{C}'$ is either 0 or 1, where 1 means that $\mathcal{C}'$ can be partitioned into s, possibly empty, subcollections $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s$ such that, for every $j \in \{1, 2, \ldots, s\}$, there exists a MUL-tree $\mathcal{T}_j$ on M_j such that $\mathcal{C}_j \subseteq \mathcal{C}(\mathcal{T}_j)$ holds, and the entry is 0 if $\mathcal{C}'$ cannot be partitioned in such a way. The collections $\mathbf{M}$ considered in $\mathbf{DP}_2$ result from a preprocessing step described in [4] that partitions the given multiset M into independent submultisets. Checking compatibility of $\mathcal{C}$ then essentially amounts to checking whether there exists an assignment of the clusters in $\mathcal{C}$ to the submultisets $M' \subseteq M$ resulting from the partitioning such that, for each such M', the collection of clusters assigned to M' is compatible when viewed as clusters on M'. The computation of an entry of $\mathbf{DP}_2$ involves the look-up of other entries in $\mathbf{DP}_2$ as well as the look-up of entries in $\mathbf{DP}_1$.

It follows from a key observation in [4] that it suffices to consider only such submultisets of M in the dynamic programming tables that have an empty intersection with the thin part M^* of M. As a consequence the size of the tables can be bounded by $c^{\Delta(M)}$ for some constant $c > 1$. Still, for practically relevant values of $\Delta(M)$, the size of the tables is too large to be used in a straight forward way. However, through computational experiments (cf. Figure 3) we found that in both tables usually only a small fraction of the entries is actually accessed during a run of our algorithm. Therefore, using a sparse matrix implementation of the tables that only stores those entries that are accessed at least once, we were able to overcome this problem.

To further speed-up the computation of $\mathcal{C}$, we developed the following simple, yet remarkably effective rule that helps to reduce the number of accessed entries in $\mathbf{DP}_2$ which, in turn, also reduces the number of accessed entries in $\mathbf{DP}_1$. To describe this rule, consider

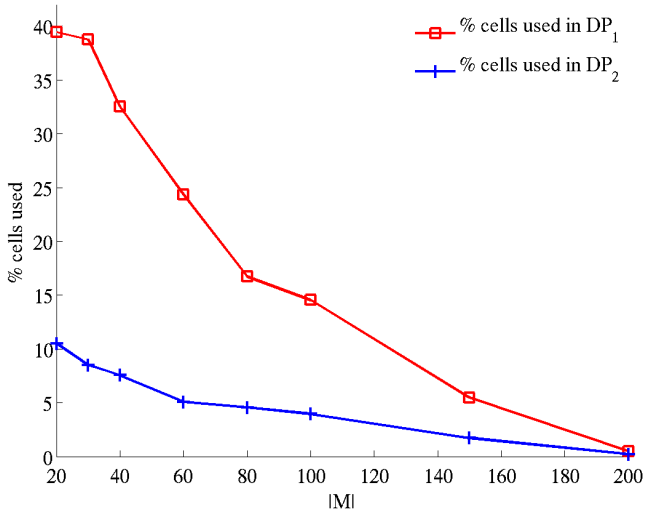

Figure 3: Percentage of accessed entries in $\mathbf{DP}_1$ and $\mathbf{DP}_2$ plotted against the size $|M|$ of the multiset M. Each data point shown is the average over 1000 experiments with varying values of $\Delta(M)$.

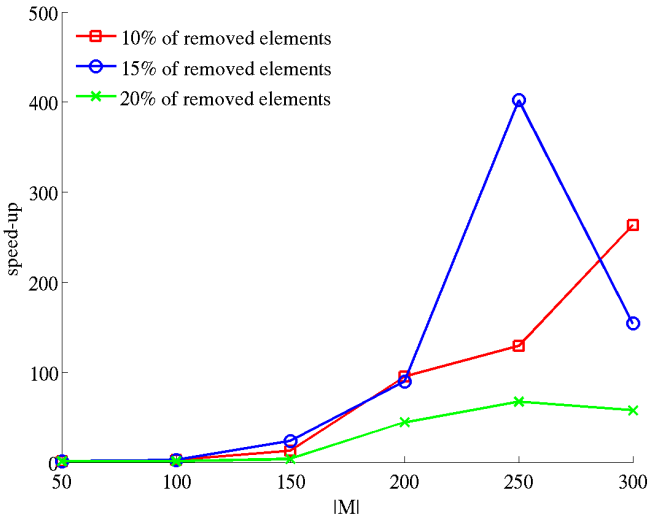

Figure 4: Illustration of the impact of the rule (*) described in the text for reducing the number of entries accessed in $\mathbf{DP}_2$ expressed as the speed-up achieved using the rule in comparison to not using it. Each data point is averaged over 100 datasets with $|M|$ as depicted and $\Delta(M) = 20\%|M|$.

the entry in $\mathbf{DP}_2$ for $\mathbf{M} = \{M_1, M_2, \ldots, M_s\}$ and $\mathcal{C}'$.

(*) If there exists some $j \in \{1, 2, \ldots, s\}$ such that $\bigcup_{C \in \widetilde{\mathcal{C}}} C \subseteq M_j$ holds for the collection $\widetilde{\mathcal{C}}$ of those clusters $C \in \mathcal{C}'$ with $C \subseteq M_j$ then we need not consider this entry of $\mathbf{DP}_2$. Put differently, it is sufficient to consider the entry for $\mathbf{M}'$ and $\mathcal{C}''$, where $\mathbf{M}'$ is obtained by removing M_j from $\mathbf{M}$ and $\mathcal{C}''$ is obtained by removing the clusters in $\widetilde{\mathcal{C}}$ from $\mathcal{C}'$.

The impact of (*) is illustrated in Figure 4. In this experiment, simulated input lists $\mathcal{L}$ were obtained by first generating a random MUL-tree $\mathcal{T}$ as described in Section 2.2 and then using $\mathcal{T}$ to generate a collection $\mathbf{T}$ of 100 MUL-trees by restricting $\mathcal{T}$ to random submultisets of M that are obtained by randomly removing a fixed percentage of the elements in M.

4 Problem (2) — Assembling and scoring MUL-trees

Recall that the input is a compatible collection $\mathcal{C}$ of clusters on the multiset M. Essentially, we aim to systematically generate all binary MUL-trees $\mathcal{T}$ with $\mathcal{C} \subseteq \mathcal{C}(\mathcal{T})$ and keep track of those for which the reticulation number of the phylogenetic network $\mathcal{N}(\mathcal{T})$ is minimum. In the following, we outline first how this can be done and then present some ideas that help to speed-up the whole process. As mentioned above, this approach involves as a subproblem the problem of computing the score $\sigma(\mathcal{T})$ of a MUL-tree $\mathcal{T}$ and this problem can be shown to be NP-hard using similar ideas as those employed in [4].

4.1 The basic setup — depth first search

To systematically generate all relevant binary MUL-trees on M, we perform a depth first search which consists of two phases. In the first phase we process the non-trivial clusters in $\mathcal{C}$ according to a suitably chosen ordering described below. In the second phase the possibly non-binary MUL-trees found that induce $\mathcal{C}$ are refined to binary trees.

To start with, we sort the non-trivial clusters in $\mathcal{C}$ into some ordering $C_1, C_2, \ldots, C_t$ such that cluster C_i is maximal in $\{C_i, C_{i+1}, \ldots, C_t\}$ with respect to multiset inclusion. Then, during the first phase of the depth first search, assume that, at stage i of the search process, we have some MUL-tree $\mathcal{T}$ on M such that the non-trivial clusters induced by $\mathcal{T}$ are precisely the clusters $C_1, C_2, \ldots, C_{i-1}$. We need to check whether there exists a MUL-tree $\mathcal{T}'$ that results from refining an interior vertex u of $\mathcal{T}$ by a new vertex v in such a way that $C_v = C_i$ holds. Note that, in view of the chosen ordering in which we process the non-trivial clusters in $\mathcal{C}$, we only need to check interior vertices u of $\mathcal{T}$ for which all children are leaves. Therefore, this check can be performed very efficiently. There can be, however, more than one such tree $\mathcal{T}'$ and for each of them the next cluster C_{i+1} is processed in a separate branch in the

depth first search. If there is no such tree $\mathcal{T}'$ we back track immediately.

When in our search we reach the point where the current MUL-tree $\mathcal{T}$ is such that the non-trivial clusters induced by $\mathcal{T}$ are precisely the clusters in $\mathcal{C}$, we continue refining $\mathcal{T}$ further, but now it does not matter any longer which clusters are induced by the new vertices added during the refining. A single step in this phase of the depth first search consists in selecting an interior vertex u in $\mathcal{T}$ with at least three children and then considering all possible ways to turn it into a vertex with precisely two children. This is done by first refining u by a new vertex v_1 and then, in case u has still more than two children, by refining u further by another vertex v_2 so that in the resulting MUL-tree $\mathcal{T}'$ vertex u has precisely two children, namely v_1 and either one of its original children or v_2.

During the entire search process we maintain an initially empty list $\mathcal{B}$ of pairwise non-isomorphic binary MUL-trees $\mathcal{T}$ with $\mathcal{C} \subseteq \mathcal{C}(\mathcal{T})$ and for which the score $\sigma(\mathcal{T})$ is minimum among the binary MUL-trees encountered so far. We denote this score by $\sigma_{\mathcal{B}}$. So, whenever the current MUL-tree $\mathcal{T}$ considered in the depth first search is binary, we compute $\sigma(\mathcal{T})$. If $\sigma(\mathcal{T}) < \sigma_{\mathcal{B}}$ we remove all MUL-trees from $\mathcal{B}$ and then add $\mathcal{T}$ to it. If $\sigma(\mathcal{T}) = \sigma_{\mathcal{B}}$ we check if $\mathcal{T}$ is isomorphic to some MUL-tree in $\mathcal{B}$ and, in case it is not, add it to $\mathcal{B}$. And if $\sigma(\mathcal{T}) > \sigma_{\mathcal{B}}$ we keep $\mathcal{B}$ as is and discard $\mathcal{T}$.

In computational experiments we found that the second phase of the depth first search is by far the more time consuming one accounting for up to 95% of the time. Therefore, in the next section, we focus on ideas for how to speed-up the computation during this phase.

4.2 Speeding-up the second phase The key observation we rely on in the following is that it only makes sense to refine a vertex u by a new vertex v if there is a chance that in the resulting binary MUL-tree there is at least one other vertex v' with $C_v = C_{v'}$. Note that this is an immediate consequence of the way the phylogenetic network $\mathcal{N}(\mathcal{T})$ is defined. Also note that a necessary condition for the existence of such a vertex v' is that $C_v \cap M^* = \emptyset$ must hold. Before we explain how this observation can be employed, we introduce some more notation. In a MUL-tree $\mathcal{T}$ on a multiset M we denote the set of children of a vertex v by $ch(v)$ and, in addition, define the set $ch^+(v) = \{u \in ch(v) : C_u \cap M^* = \emptyset\}$.

4.2.1 Preprocessing Before starting to systematically refine vertices of degree larger than three, it is useful to first apply some preprocessing that further simplifies the task of refining the given MUL-tree in the second phase.

First note that for any vertex u with $2 \leq |ch^+(u)| < |ch(u)|$ we can safely refine u by a new vertex v with $C_v = \bigcup_{w \in ch^+(u)} C_w$. Thus, we can assume from now on that in the MUL-tree $\mathcal{T}$ we want to refine we have either $ch^+(v) = ch(v)$ or $|ch^+(v)| \leq 1$ for every vertex v. In particular, to compute the score $\sigma(\mathcal{T})$ it suffices to refine only those vertices u with $ch^+(u) = ch(u)$, that is, we need not refine $\mathcal{T}$ until it is binary but can derive $\sigma(\mathcal{T})$ already from some intermediate non-binary MUL-tree that refines $\mathcal{T}$ just enough.

Next note that sometimes we can partition the given MUL-tree $\mathcal{T}$ into subtrees and then score these subtrees independently of one another. To outline how this can be done, define V_1 as the set of those interior vertices v of $\mathcal{T}$ other than the root for which $C_v \cap (M - C_v) = \emptyset$. Let $\leq_{\mathcal{T}}$ denote the partial order on the vertex set of $\mathcal{T}$ with $a \leq_{\mathcal{T}} b$ if vertex b lies on the path from a to the root of $\mathcal{T}$. Consider the set V_1^* of maximal elements in V_1 with respect to the restriction of $\leq_{\mathcal{T}}$ to V_1. Then the MUL-trees $\mathcal{T}_v$, $v \in V_1^*$, together with the tree $\mathcal{T}^*$ that is obtained from $\mathcal{T}$ by replacing each $\mathcal{T}_v$ by a distinguished leaf ℓ_v, $v \in V_1^*$, form subtrees that can be refined independently. Note that, while $\mathcal{T}^*$ cannot be further partitioned, it is possible that the subtrees $\mathcal{T}_v$, $v \in V_1^*$, can be further partitioned. If the latter is the case then we partition them recursively.

4.2.2 The dependency graph In this section, we describe how to speed-up the actual refining of a MUL-tree $\mathcal{T}$ on a multiset M. Let V denote the vertex set of $\mathcal{T}$ and define the collection of *potential clusters* associated with a vertex $v \in V$ with $ch^+(v) = ch(v)$ as

$$\mathcal{P}_v = \{ \bigcup_{u \in U} C_u : U \subseteq ch^+(v), 2 \leq |U| < |ch^+(v)| \},$$

that is, the collection of those clusters that can be formed by refining vertex v and which are relevant for the score of the tree.

Now, let N be a subset of V. The *dependency graph* $D_{(\mathcal{T},N)} = (V, A)$ is a directed graph with vertex set V that has a directed edge (u, v) from u to v if $u \in N$ and $C_u \in (\mathcal{P}_v \cup \{C_v\})$. Intuitively, N consists of those vertices that have already been added to refine certain vertices and an edge $(u, v) \in A$ indicates that considering the refinement by vertex u might be relevant for computing the score $\sigma(\mathcal{T})$ because there is at least one other vertex in $\mathcal{T}$ that gives, or at least will potentially give, rise to a copy of cluster C_u.

Before we describe how the graph $D_{(\mathcal{T},N)} = (V, A)$ might be used to prune the depth first search we outline briefly how $D_{(\mathcal{T},N)}$ is updated when refining a vertex $w \in V$ with $|ch(w)| = |ch^+(w)| \geq 3$. Recall that a single step consists of selecting a subset $W \subseteq ch^+(w)$

with $2 \leq |W| < |ch^+(w)|$, adding a new vertex u as a child of w to $\mathcal{T}$ and then changing each $w' \in W$ from being a child of w to being a child of u. In addition, putting $W' = ch^+(w) - W$, we add another new vertex u' as a child of w to $\mathcal{T}$ if $|W'| > 1$ holds and then change each $w' \in W'$ from being a child of w to being a child of u'. In the following we only consider the case that two new vertices, u and u', have been added. The case that only u is added is completely analogous. Let $\mathcal{T}'$ denote the resulting MUL-tree with vertex set $V' = V \cup \{u, u'\}$ and edge set E'. Put $N' = N \cup \{u, u'\}$. Then $D' = D_{(\mathcal{T}',N')} = (V', A')$ is obtained from $D_{(\mathcal{T},N)}$ by:

(i) Adding vertices u and u'.

(ii) Adding edges (u, v) from u to all vertices $v \in V$ with $C_u \in (\mathcal{P}_v \cup \{C_v\})$ and adding edges (u', v) from u' to all vertices $v \in V$ with $C_{u'} \in (\mathcal{P}_v \cup \{C_v\})$

(iii) Adding edges (v, u) for all $v \in N$ with $C_v \in (\mathcal{P}_u \cup \{C_u\})$ and adding edges (v, u') for all $v \in N$ with $C_v \in (\mathcal{P}_{u'} \cup \{C_{u'}\})$

(iv) Removing all edges (v, w) for which we no longer have $C_v \in (\mathcal{P}_w \cup \{C_w\})$.

To speed-up the scoring using the dependency graph $D_{(\mathcal{T},N)} = (V, A)$, we apply the following rule: If there exists some $u \in N$ that has no outgoing edge $(u, v) \in A$ then the vertex u is not relevant for computing the score. So, if we get into this situation we can track-back immediately. We refer to this as the *no outgoing edge rule* or *NOE-rule*, for short.

4.2.3 Computational experiments In Figure 5 we illustrate the impact of the preprocessing and the application of the NOE-rule on the run-time for simulated input MUL-trees. In the plot, each data point is an average over 100 input MUL-trees on a multiset M with the specified size and deviation from its underlying set. As is clearly visible, as soon as the size of M increases above 60, our new algorithm for Problem (2) quickly becomes several magnitudes faster than the basic setup described in Section 4.1. This observation is independent of the value for $\Delta(M)$ but is most pronounced for $\Delta(M) = 10\%$.

To shed some light on the individual impact of the preprocessing and the NOE-rule, we also measured the speed-up achieved by applying them together in comparison to using the preprocessing alone (cf. Figure 6). The different curves in the plot highlight the impact of the deviation of the input MUL-tree $\mathcal{T}$ from being a binary tree, that is, the number $\kappa(\mathcal{T})$ of single refinement steps needed to turn $\mathcal{T}$ into a binary tree. This number

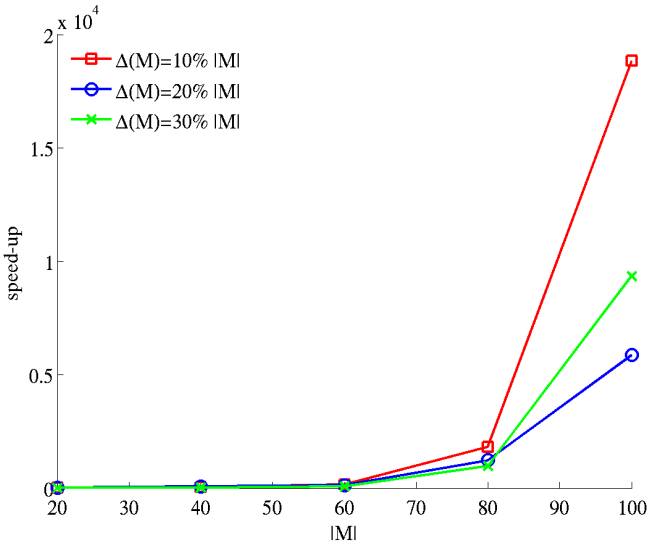

Figure 5: Results of an experiment (with $\kappa(\mathcal{T}) = 10\%$ and $\Delta(M)$ values as specified) in terms of the achieved speed-up in comparison to the basic setup described in Section 4.1 when executing the preprocessing step described in the text in conjunction with the NOE-rule.

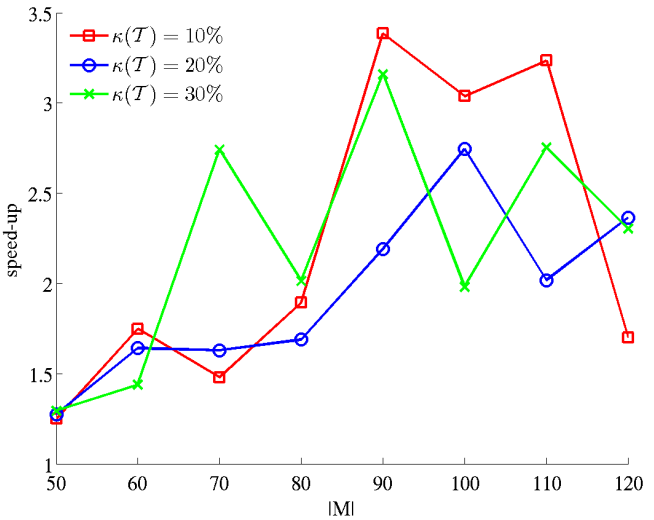

Figure 6: Speed-up due to applying the NOE-rule in addition to the preprocessing for different values of $\kappa(\mathcal{T})$ ($\Delta(M)$ was fixed to $20\%|M|$).

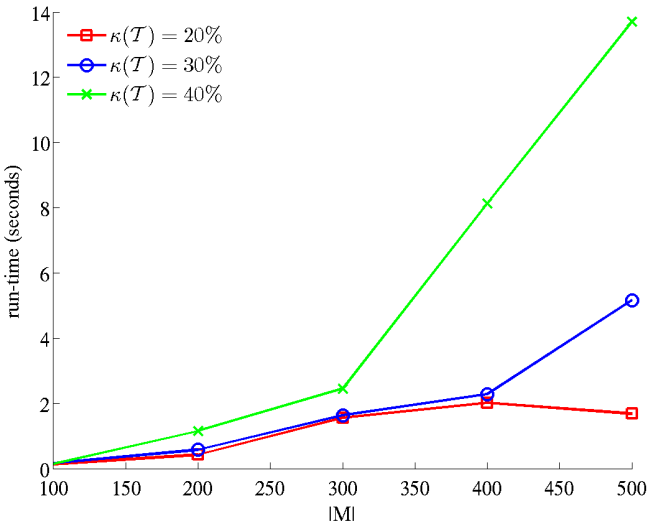

Figure 7: Run-times for the algorithm for Problem (2) when both the preprocessing and the NOE-rule are applied ($\Delta(M) = 10\%|M|$).

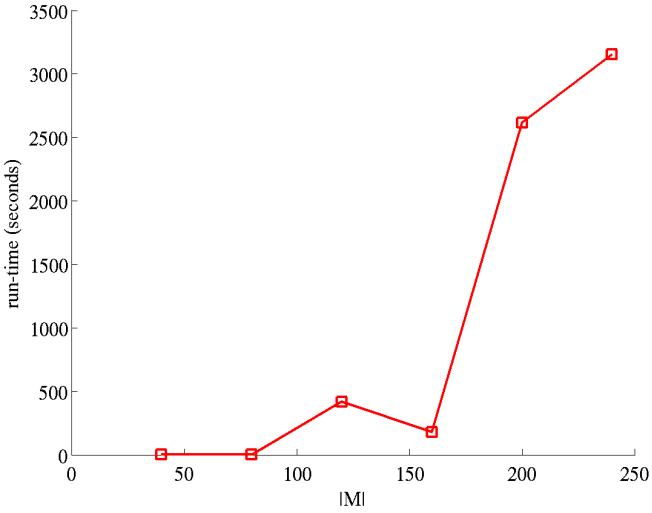

Figure 8: Run-times of the consensus method presented in [8] when using our new algorithms for Problems (1) and (2).

is given as percentage of the maximum possible number of refinement steps for the specified size of the multiset M. Overall, the picture is not so clear cut. It appears that there are specific combinations of $|M|$, $\Delta(M)$ and $\kappa(\mathcal{T})$ for which a significant additional speed-up can be achieved by using the NOE-rule.

Next, in Figure 7 the actual run-times for experiments illustrating the combined impact of the preprocessing and the NOE-rule are depicted. As is clearly visible, the run-time grows in the size $|M|$ of M and this growth is dependent on the value for $\kappa(\mathcal{T})$.

Finally, in Figure 8, we depict the run-time for the overall method outlined in Section 2.3 when using our new algorithms for Problems (1) and (2). The input collections $\mathbf{T}$ of MUL-trees were generated in the same way as in the experiments described at the end of Section 3 with both $\Delta(M)$ and the number of randomly removed elements fixed to $10\%|M|$. Note that, when using the basic version of the method (rule (*) for Problem (1) switched off, no preprocessing and no application of the NOE-rule for Problem (2)), even for $|M| = 40$ most of the simulated inputs could not be processed within 24 hours, the limit set for all experiments.

5 Discussion and conclusions

Our new algorithms for Problems (1) and (2) make it possible to apply the consensus method described in [8] to collections of MUL-trees with more than 200 leaves, as illustrated in Figure 8. We also applied the method with the new algorithms to a biological data set presented in [7]. From an alignment of 71 DNA sequences from flowering plants for which there is evidence of hybridization, we constructed 1000 phylogenetic trees using PAUP* (version 4.0b10) with leaves labeled by the sequences in the alignment. To obtain MUL-trees, the sequences were replaced by the plant species they were sequenced from. By construction, these MUL-trees were all on the same multiset M of size 71 with $\Delta(M) = 28$ and $|M^*| = 25$. It took 30 seconds to process this collection of MUL-trees to form a consensus. In contrast, the basic version of the method was again not able to process this data set within 24 hours.

The computational experiments also suggest that the parameter $\Delta(M)$ that was used in [4] to develop a fixed-parameter algorithm is indeed of practical relevance. Another parameter that seems to have a significant impact in the context of computing the score $\sigma(\mathcal{T})$ for a MUL-tree $\mathcal{T}$ is the number $\kappa(\mathcal{T})$ of single refinement steps needed to turn $\mathcal{T}$ into a binary MUL-tree. When applying the preprocessing and the NOE-rule, however, the picture seems to get a little blurred. We think that this is due to the fact that when applying these we actually aim to *not* completely refine $\mathcal{T}$ to a binary tree.

In future work it could be interesting to explore ways to also speed-up the first phase of the depth first search described in Section 4.1, which currently follows a straightforward approach. Also, the computational complexity of the problem where the input is just a compatible collection $\mathcal{C}$ of clusters on some multiset

M and we want to find a binary MUL-tree $\mathcal{T}$ on M with minimum score $\sigma(\mathcal{T})$ such that $\mathcal{C} \subseteq \mathcal{C}(\mathcal{T})$ holds, is, to the best of our knowledge, open. In relation to this, the reduction establishing NP-hardness for the related problem of computing the score $\sigma(\mathcal{T})$ for a given MUL-tree $\mathcal{T}$ mentioned at the beginning of Section 4 employs MUL-trees whose maximum vertex degree is not bounded by a constant. So, is this problem fixed-parameter tractable with respect to the maximum degree? Similarly, is this problem fixed-parameter tractable with respect to the score of the tree?

Acknowledgments This work was supported by the British Council and the DAAD within the ARC Programme. We thank J. Bowman for providing us with the sequence alignment for the data set studied in [7]. The authors would like to acknowledge that the experiments discussed in this paper were carried out on the High Performance Computing Cluster supported by the Research Computing Service at the University of East Anglia. In addition, we thank the anonymous reviewers for their helpful comments on how to improve the presentation of our results.

References

[1] G. Ganapathy, B. Goodson, R. Jansen, H.-S. Le, V. Ramachandran, and T. Warnow. Pattern identifcation in biogeography. *IEEE Transactions on Computational Biology and Bioinformatics*, 3:334–346, 2006.

[2] S. Guillemot, J. Jansson, and W.-K. Sung. Computing a smallest multilabeled phylogenetic tree from rooted triplets. *IEEE Transactions on Computational Biology and Bioinformatics*, 8:1141–1147, 2011.

[3] E. Harding. The probabilities of rooted tree-shapes generated by random bifurcation. *Advances in Applied Probability*, 3:44–77, 1971.

[4] K. T. Huber, M. Lott, V. Moulton, and A. Spillner. The complexity of deriving multi-labeled trees from bipartitions. *Journal of Computational Biology*, 15:639–651, 2008.

[5] K. T. Huber and V. Moulton. Phylogenetic networks from multi-labelled trees. *Journal of Mathematical Biology*, 52:613–632, 2006.

[6] D. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, 23:254–267, 2006.

[7] J.-Y. Lee, K. Mummenhoff, and L. Bowman. Allopolyploidization and evolution of species with reduced floral structures in Lepidum L. (Brassicaceae). *PNAS*, 99:16835–16840, 2002.

[8] M. Lott, A. Spillner, K. Huber, A. Petri, B. Oxelman, and V. Moulton. Inferring polyploid phylogenies from multiply-labeled gene trees. *BMC Evolutionary Biology*, 9, 2009.

[9] M. Popp, P. Erixon, F. Eggens, and B. Oxelman. Origin and evolution of a circumpolar polyploid species complex in Silene (Caryophyllaceae) inferred from low copy nuclear RNA polymerase introns, rDNA, and chloroplast DNA. *Systematic Botany*, 30:302–313, 2005.

[10] M. Popp and B. Oxelman. Origin and evolution of north american polyploid silene (caryophyllaceae). *American Journal of Botany*, 94:330–349, 2007.

[11] C. Scornavacca, V. Berry, and V. Ranwez. From gene trees to species trees through a supertree approach. In *Proc. International Conference on Language and Automata Theory and Applications*, pages 702–714, 2009.

[12] J. Smedmark, T. Eriksson, and B. Bremer. Allopolyploid evolution in geinae (colurieae: Rosaceae) - building reticulate species trees from bifurcating gene trees. *Organisms Diversity and Evolution*, 5:275–283, 2005.

[13] G. Yule. A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S. *Philosophical Transactions of the Royal Society of London. Series B*, 213:21–87, 1925.