Algorithmica (2026) 88:58
https://doi.org/10.1007/s00453-026-01400-9

MUL-Tree Pruning for Consistency and Compatibility

Christopher Hampson[1] · Daniel J. Harvey[2] · Costas S. Iliopoulos[1] · Jesper Jansson[2] · Zara Lim[1] · Wing-Kin Sung[3,4]

Received: 30 September 2024 / Accepted: 13 June 2026

Abstract

A multi-labelled tree (or MUL-tree) is a rooted tree leaf-labelled by a set of labels, where each label may appear more than once in the tree. We consider the MUL-tree Set Pruning for Consistency problem (MULSETPC), which takes as input a set of MUL-trees and asks whether there exists a perfect pruning of each MUL-tree that results in a consistent set of single-labelled trees. MULSETPC was proven to be NP-complete by Gascon et al. when the MUL-trees are binary, each leaf label is used at most three times, and the number of MUL-trees is unbounded. To determine the computational complexity of the problem when the number of MUL-trees is constant was left as an open problem. Here, we resolve this question by proving a much stronger result, namely that MULSETPC is NP-complete even when there are only two MUL-trees, every leaf label is used at most twice, and either every MUL-tree is binary or every MUL-tree has constant height. Furthermore, we introduce an extension of

A preliminary version of this article was presented at CPM 2023 [18]. Partially funded by JSPS KAKENHI grant 22H03550/23K24807.

✉ Daniel J. Harvey
Daniel.Harvey87@gmail.com

Christopher Hampson
christopher.hampson@kcl.ac.uk

Costas S. Iliopoulos
costas.iliopoulos@kcl.ac.uk

Jesper Jansson
jj@i.kyoto-u.ac.jp

Zara Lim
zara.lim@kcl.ac.uk

Wing-Kin Sung
kwksung@cuhk.edu.hk

1 Department of Informatics, King's College London, London, UK

2 Graduate School of Informatics, Kyoto University, Kyoto, Japan

3 Department of Chemical Pathology, The Chinese University of Hong Kong, Hong Kong, China

4 Laboratory of Computational Genomics, Li Ka Shing Institute of Health Sciences, Hong Kong, China

Published online: 08 July 2026

MULSETPC that we call MULSETPComp, which replaces the notion of consistency with compatibility. We prove that MULSETPComp is NP-complete even in two highly restricted cases – when there are only two MUL-trees, every MUL-tree has constant height and either every leaf label is used at most thrice or the instance contains a single-labelled tree containing all leaf labels. Finally, we present a polynomial-time algorithm for a special case of MULSETPC with a constant number of binary MUL-trees where every leaf label appears exactly once in at least one MUL-tree.

1 Introduction

In evolutionary biology, leaf-labelled (phylogenetic) trees are commonly employed to describe the evolution of species using leaf labels to represent different species [11]. Comparisons of these structures are used particularly in phylogenetic inferences – similarities may indicate evolutionary patterns, whereas differences may highlight genetic mutations. The measure of similarity between phylogenetic trees has been defined by multiple alternate metrics, such as the Robinson-Foulds distance [24], subtree pruning and regraft (SPR) distances [5, 30], and maximum agreement subtrees [2, 7, 12]. Other problems related to phylogenetic trees include constructing supertrees [1, 3, 4, 29] or consensus trees [6, 11, 16] which can determine relations or interactions between smaller phylogenetic trees.

Phylogenetic trees are classically described as single-labelled trees, where no label appears on the leaves of the tree more than once. Typically, construction or comparison algorithms of such phylogenetic trees make use of this property to reduce computational costs. Multi-labelled trees (or MUL-trees) are a generalisation of single-labelled trees in which multiple leaves may be labelled by the same label. MUL-trees can be useful to depict genome duplication, lineage sorting, or lateral gene transfer [19]. Other applications include the construction of phylogenetic networks by folding operations [19] [14, 15], biogeography [13][17,23], the study of host-parasite cospeciation [21], and gene evolution studies [19, 22, 25].

MUL-trees have been far less investigated than their single-labelled counterparts and many computational problems become NP-hard when extended to MUL-trees. For example, the majority rule consensus tree for a set of k single-labelled trees with n leaf labels each can be computed in $O(nk)$ time [16], but is NP-hard to compute for MUL-trees [8]. Another example is the problem of determining if a given a set $\mathcal{R}$ of so-called rooted triplets (single-labelled binary phylogenetic trees with exactly three leaves each) is consistent. Informally, a set of phylogenetic trees is consistent if it can be combined into a single-labelled phylogenetic tree that contains all of its members as embedded subtrees; see Sect. 2 below for the formal definition of "consistent". This problem can be solved in polynomial time by Aho *et al.*'s BUILD algorithm [1], which also outputs a tree that contains all of $\mathcal{R}$ when such a tree exists. In contrast, relaxing the requirement on $\mathcal{R}$ having to be able to be combined into a single-labelled tree and instead asking if $\mathcal{R}$ can be combined into a MUL-tree where one of the leaf labels appears twice and all the other leaf labels appear just once already makes the problem NP-hard [17].

A few polynomial-time algorithms do exist for MUL-trees – Cui et al.[8] presented a $O(n^2k + nk^2)$-time algorithm for building a majority rule consensus MUL-tree in which each leaf label appears at most twice, based on a reduction to the Perfect Phylogeny Haplotyping problem [10]. Furthermore, the maximum agreement subtree (MAST) distance between two MUL-trees can be computed in quadratic time, though it also becomes NP-complete when generalised to more than two MUL-trees [13, 18]. Other approaches convert MUL-trees into single-labelled trees which can be input to existing algorithms [25, 28].

This paper further investigates the computational complexity of MUL-tree-related problems by considering the *MUL-tree Set Pruning for Consistency* problem (MULSETPC), which takes as input a set of MUL-trees, and outputs whether or not there exists a pruning of the MUL-trees which gives a consistent set of single-labelled trees. Gascon et al. showed that, in general, MULSETPC is NP-complete via a polynomial reduction from 3-SAT [14,16]. However, their reduction from an instance of 3-SAT with m variables and z clauses gives an instance of MULSETPC containing $m + z + 1$ MUL-trees; moreover these MUL-trees may have labels occurring three times. Here we prove that MULSETPC is still NP-complete, even when restricted to instances involving only two MUL-trees, or in which every leaf label appears at most twice within a MUL-tree. This totally resolves the open question of Gascon et al. [14,16] regarding the parameterised complexity of MULSETPC when the parameter is the number of input trees. Also, we identify tractable fragments of MULSETPC which can be solved in polynomial time, in short, instances in which each label appears exactly once in at least one MUL-tree.

We also present a generalisation of MULSETPC called MULSETPComp, which asks for a *compatible* set of trees instead of a *consistent* set of trees. Tree compatibility is a generalisation of tree consistency where we allow the supertree displaying the set to instead display a refinement of each tree rather than the tree itself – again, a more rigorous definition is given later. Tree compatibility is relevant when determining the existence of a supertree for a given set of phylogenetic trees [1, 26]. This is because experimental data often contains uncertainty, which can be expressed by non-binary nodes in the tree; this necessitates the use of compatibility. Compatibility is also relevant to other questions such as the incomplete directed perfect phylogeny problem [29]. Our interest in tree compatibility was partially motivated by recent improvements in compatibility testing; in particular, by the efficient BUILDST algorithm of Deng and Fernández-Baca [9] which can determine if a set of single-labelled trees is compatible, and if so, construct a corresponding supertree, in polynomial time. Here, we prove that MULSETPComp is NP-complete, even when restricted to instances involving only two MUL-trees, or in which every leaf label appears at most thrice within a MUL-tree. We also prove that MULSETPComp is NP-complete, even when restricted to instances containing only two MUL-trees where one MUL-tree is a single-labelled tree containing all leaf labels. This second case is interesting as it suggests that the aforementioned tractable fragments of MULSETPC which can be solved in polynomial time are not tractable when converted to equivalent MULSETPComp instances.

The rest of the paper is organized as follows. In Sect. 2, we introduce the preliminary notation and definitions. In Sect. 3 we present the improved NP-completeness proof for MULSETPC by reduction from the Boolean 3-SAT problem. In Sect. 4 we give a

NP-completeness proof for MULSETPComp by reduction from the Exact 3-cover with Multiplicity 3 problem. Section 5 contains our polynomial-time results for tractable instances of MULSETPC. Finally, in Sect. 6 we present our conclusions and a few open problems.

2 Preliminaries

We shall use the following standard definitions on trees.

Definition 1 *(Basic tree definitions)* All trees we consider are rooted and unordered. If x, y are nodes in a tree T, then y is an *ancestor* of x (and x a *descendant* of y) if y lies on the unique path from x to the root of T. We denote this by $x \leq y$. Additionally, if $y \neq x$ then y is a *proper ancestor* of x, which we denote by $x < y$. If $x < y$ and y is adjacent to x then y is the *parent* of x and x a *child* of y. If x and x' are both children of y then x and x' are *siblings*. The *lowest common ancestor* of nodes x and y, denoted $\mathrm{lca}_T(x, y)$, is the node z such that $x \leq z$, $y \leq z$, and no proper descendant of z also satisfies these properties. The empty tree, denoted by $T_\emptyset$, is the unique tree which contains no nodes.

We use the following definition of leaf-labelled trees, which takes the definitions of Gascon et. al.[14] and generalises them to the case where the tree may not be binary.

Definition 2 *(Leaf-labelled trees)* A *leaf-labelled tree* $(T, \mathcal{X})$ is a (rooted, unordered) tree T where no node has exactly one child and where each leaf has been assigned a label from a set of labels $\mathcal{X}$. (We will sometimes refer to T as a leaf-labelled tree *on* $\mathcal{X}$, and omit $\mathcal{X}$ if it is clear from context.) A leaf-labelled tree is a *single-labelled tree* if every label in $\mathcal{X}$ is used at most once. Alternatively, a *multi-labelled tree* or *MUL-tree* is a leaf-labelled tree where we allow each label in $\mathcal{X}$ to label multiple leaves. We say a MUL-tree has *multiplicity* k if each leaf label appears at most k times. Let $L(T) \subseteq \mathcal{X}$ denote the set of leaf labels appearing in T and let $D(T) \subseteq L(T)$ denote the set of leaf labels appearing only once in T.

Recall a *complete binary tree* is a binary tree in which all levels of the tree but the last are completely filled, and the nodes on the last level fill from the left. (Note that such a tree may not satisfy the requirements for a phylogenetic tree, since an even number of nodes will force a node with one child in the penultimate layer.) Also recall that a *caterpillar* is a tree in which every node has at most one non-leaf child.

Note that in a single-labelled tree we sometimes abuse notation and identify the leaf and the leaf label. We do the same in a MUL-tree only if the context is clear and there is no possibility of confusion.

If u is a node of T then T^u denotes the subtree rooted at u containing u and all its descendants, maintaining the same leaf-labelling as T on the remaining leaves. Let $D^u := D(T^u)$.

Definition 3 *(Pruning and Perfect Pruning)* Given a leaf-labelled tree T, let y denote a leaf node of T and x the parent of y. We *prune* the leaf y in the following manner:

- Delete the leaf y.

- If x still has at least two children, do nothing else.
- Alternatively, if x now has only one child and is not the root, suppress the node x.
- Finally, if x has only one child z and is the root, delete x and make z the new root.

A *perfect pruning of* T is a single-labelled tree T' such that $L(T') = L(T)$, created by (possibly repeated) prunings of T.

In other words, in a perfect pruning of a MUL-tree T, for every label that appears more than once in T we prune away all but exactly one copy of the label to obtain a single-labelled tree. If T is a single-labelled tree, then its only perfect pruning is itself.

Given a leaf-labelled tree T and a set of leaf labels $L' \subseteq L(T)$, let $T\restriction_{L'}$ denote the leaf-labelled tree constructed by pruning from T every leaf labelled by a label from $L(T) - L'$. (That is, only leaves labelled by L' remain.) Two leaf-labelled trees T_1 and T_2 are *leaf-label isomorphic* if there is an isomorphism between T_1 and T_2 which preserves the labelling of the leaves. We say that a leaf-labelled tree T on L *displays* a single-labelled tree T' on $L' \subseteq L$ if there exists a perfect pruning T^* of T such that $T^*\restriction_{L'}$ is leaf-label isomorphic to T'. Note T does not display T' if $L(T')$ is not a subset of $L(T)$.

Definition 4 *(Refinement)* Given single-labelled trees T and T^*, we say T^* is a refinement of T if T can be obtained from T^* by (possibly repeated) contractions of non-leaf edges, where we treat a contraction as merging the child node into the parent node. We write $T \leq T^*$.

What follows is the definition of a *consistent set*, and the very similar definition of a *compatible set*.

Definition 5 *(Consistent Set)* Consider a set of single-labelled trees $T_1, \ldots, T_k$ with corresponding label sets $L_1, \ldots, L_k$. We say this set is *consistent* if there exists a single-labelled tree T on label set $L = \bigcup_{i=1}^{k} L_i$ such that for every $i = 1, \ldots, k$, T displays T_i.

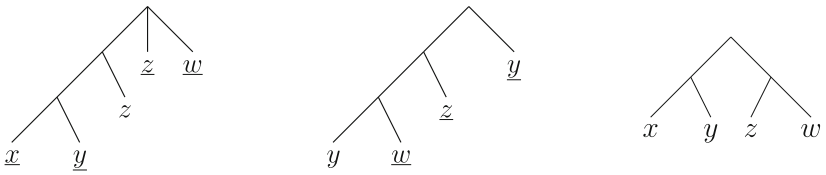
Note in the above definition that if $L_1 = \cdots = L_k$ then $L = L_1$ and so the set $T_1, \ldots, T_k$ is consistent if and only if the trees are pairwise leaf-label isomorphic.

Definition 6 *(Compatible Set)* Consider a set of single-labelled trees $T_1, \ldots, T_k$ with corresponding label sets $L_1, \ldots, L_k$. We say this set is *compatible* if there exists a single-labelled tree T on label set $L = \bigcup_{i=1}^{k} L_i$ such that for every $i = 1, \ldots, k$, $T\restriction_{L_i}$ is a refinement of T_i.

Note that every consistent set of trees is also a compatible set, but the converse does not hold in general.

The MUL-set pruning for consistency problem can then be defined as follows:

MUL-tree Set Pruning for Consistency (MULSETPC) Problem:
Input: $(\mathcal{M}, \mathcal{X})$ where $\mathcal{M}$ is a set of MUL-trees on $\mathcal{X}$.
Output: $\exists$? a perfect pruning of each tree of $\mathcal{M}$ resulting in a consistent set of trees.



(a) T_1 (b) T_2 (c) T^*

Fig. 1 Let T_1, T_2, and T^* be the three trees with $\mathcal{X} = \{x, y, z, w\}$ shown above. If we prune away the non-underlined labels in T_1 and T_2 then $T^*{\restriction}_{L(T_i)}$ is a refinement of the pruned T_i for $i \in \{1, 2\}$, which shows that there exists a perfect pruning of $\{T_1, T_2\}$ giving a compatible set of trees. In contrast, there is no perfect pruning of $\{T_1, T_2\}$ giving a consistent set of trees because neither of the two single-labelled subtrees with leaf labels y, z, and w displayed by T_1 is also displayed by T_2. Note, however, that if the label w in T_1 is changed to x then $\{T_1, T_2\}$ becomes consistent since this label can be pruned along with one leaf labelled by z to obtain a perfect pruning of T_1 which is displayed by T^*

We introduce the following problem, which substitutes *compatible* for *consistent* sets:

MUL-tree Set Pruning for Compatibility (MULSETPComp) Problem:
Input: $(\mathcal{M}, \mathcal{X})$ where $\mathcal{M}$ is a set of MUL-trees on $\mathcal{X}$.
Output: ∃? a perfect pruning of each tree of $\mathcal{M}$ resulting in a compatible set of trees.

See Fig. 1 for an example that illustrates the difference between MULSETPC and MULSETPComp.

3 NP-completeness for MULSETPC instances with two MUL-trees and multiplicity 2

In this section we consider the MULSETPC problem, and show that it is NP-complete even when considering a heavily restricted set of instances. Specifically, we consider instances with at most two MUL-trees, where the multiplicity is 2, and where either the MUL-trees are binary or have height at most 5.

Proving these results requires two major cases – the binary case and the restricted height case. The methods for proving these two cases are very similar to each other, hence we shall prove both simultaneously. The core of our proof will be a reduction from 3-SAT [19].

3-satisfiability (3-SAT) Problem:
Input: A Boolean formula $C = (C_1 \wedge C_2 \wedge \ldots \wedge C_z)$ on a finite set of literals $\{l_1, l_2, \ldots, l_m, \bar{l}_1, \bar{l}_2, \ldots, \bar{l}_m\}$ where the formula is in conjunctive normal form and each clause contains 3 literals.
Output: ∃? a satisfying valuation $\mathcal{V}$ of C.

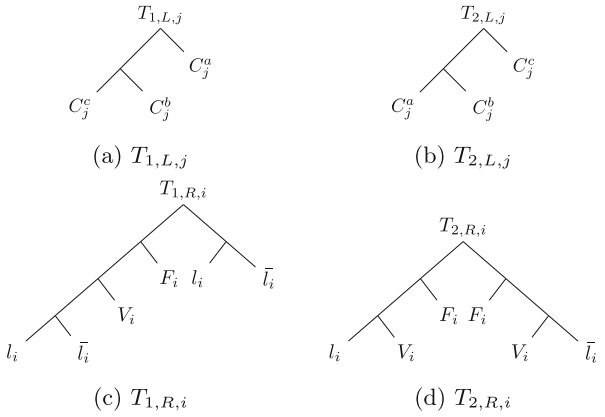


Fig. 2 Subtrees $T_{1,L,j}$, $T_{2,L,j}$, $T_{1,R,i}$, $T_{2,R,i}$ for MULSETPC

Our first goal is to construct, given an instance of 3-SAT, four MUL-trees $T_1(B)$, $T_2(B)$, $T_1(S)$ and $T_2(S)$ which we will use in our corresponding instances of MULSETPC. (Note that the B here stands for *binary*, and the S for *star* – essentially the difference between these trees is that we will switch out binary structures with unbounded height for stars which have bounded height but unbounded degree.) Our set of leaf labels $\mathcal{X}$ consists of the following:

- $\{l_i, \bar{l}_i \mid i = 1, \ldots, m\}$, the set of literals,
- $\{F_i, V_i \mid i = 1, \ldots, m\}$, a pair of "dummy" labels for each variable and
- $\mathcal{P} := \{C_j^1, C_j^2, C_j^3 \mid j = 1, \ldots, z\}$, a triple of "position" labels representing the three places of each clause of C.

Define $h := \mathcal{P} \to \{l_i, \bar{l}_i \mid i = 1, \ldots, m\}$ as the function which maps a position label C_j^x to the literal found in that position. For example, if $C_1 = (l_1 \vee l_4 \vee \bar{l}_6)$ then $h(C_1^1) = l_1$, $h(C_1^2) = l_4$ and $h(C_1^3) = \bar{l}_6$. We treat $\mathcal{P}$ as being ordered first by the index of the clause and then by the position. Let $H(l_i) := \{C_j^x \mid h(C_j^x) = l_i\}$, and define $H(\bar{l}_i)$ similarly.

Our trees $T_1(B)$, $T_2(B)$, $T_1(S)$, $T_2(S)$ will be constructed from six types of subtrees $T_{1,L,j}$, $T_{2,L,j}$, $T_{1,R,i}(B)$, $T_{2,R,i}(B)$, $T_{1,R,i}(S)$ and $T_{2,R,i}(S)$, where $i = 1, \ldots, m$ and $j = 1, \ldots, z$. See Fig. 2 for the subtrees $T_{1,L,j}$, $T_{2,L,j}$, $T_{1,R,i}$ and $T_{2,R,i}$; we now explain how to construct $T_{1,R,i}(B)$, $T_{2,R,i}(B)$, $T_{1,R,i}(S)$ and $T_{2,R,i}(S)$ from $T_{1,R,i}$ and $T_{2,R,i}$.

Let v_i denote the leaf labelled V_i in $T_{1,R,i}$ and let u_i denote the parent of v_i. Let u_i^+ and u_i^- denote the parents of leaves labelled l_i and $\bar{l}_i$ respectively in $T_{2,R,i}$. Let v_i^+ denote the child of u_i^+ labelled by V_i, and v_i^- denote the child of u_i^- labelled by V_i.

Initialise $T_{1,R,i}(B)$ and $T_{1,R,i}(S)$ as copies of $T_{1,R,i}$, then do the following:

- If $H(l_i) \cup H(\bar{l}_i) = \emptyset$, make no further changes in either case.
- Otherwise, in $T_{1,R,i}(B)$ subdivide the edge $u_i v_i$ $|H(l_i) \cup H(\bar{l}_i)|$ times, and add to each new node an adjacent leaf. Label these leaves with $H(l_i) \cup H(\bar{l}_i)$, respecting the ordering such that the first label is closest to u_i.

- In $T_{1,R,i}(S)$, instead subdivide the edge $u_i v_i$ exactly once, and for each label $C_j^x \in H(l_i) \cup H(\bar{l}_i)$ add a leaf adjacent to the new node labelled by C_j^x.

Initialise $T_{2,R,i}(B)$ and $T_{2,R,i}(S)$ as copies of $T_{2,R,i}$, and then:

- If $H(l_i) \neq \emptyset$, then in $T_{2,R,i}(B)$ subdivide $u_i^+ v_i^+$ $|H(l_i)|$ times and add an leaf adjacent to each new node. Label these leaves with $H(l_i)$, respecting the ordering so that the first label is closest to u_i^+. In $T_{2,R,i}(S)$ subdivide $u_i^+ v_i^+$ once and add $|H(l_i)|$ leaves adjacent to the new node, labelled by $H(l_i)$.
- If $H(\bar{l}_i) \neq \emptyset$, repeat the previous step, substituting $H(\bar{l}_i)$ for $H(l_i)$ and $u_i^- v_i^-$ for $u_i^+ v_i^+$.

Construct $T_{1,L}(B)$ by taking a complete binary tree on z leaves (recall that z is the number of clauses), suppressing any nodes with exactly one child, and then identifying the root of each $T_{1,L,j}$ (ordered by j) with exactly one of the leaves (ordered left-to-right). Construct $T_{2,L}(B)$ in the same fashion, substituting $T_{2,L,j}$ for $T_{1,L,j}$. Construct $T_{1,R}(B)$ by taking a complete binary tree on m leaves, suppressing any nodes with exactly one child, and then identifying the root of each $T_{1,R,i}(B)$ (ordered by i) with exactly one of the leaves (ordered left-to-right). Again, construct $T_{2,R}(B)$ in the same fashion, substituting $T_{2,R,i}(B)$ for $T_{1,R,i}(B)$. Finally, construct $T_1(B)$ by taking a root node $r(T_1(B))$ and adding an edge to the roots of $T_{1,L}(B)$ and $T_{1,R}(B)$; construct $T_2(B)$ in the obvious equivalent fashion. To create $T_1(S)$ take a root node $r(T_1(S))$ and add an edge to the roots of each $T_{1,L,j}$ and $T_{1,R,i}(S)$; create $T_2(S)$ in the obvious equivalent fashion.

Given an instance C of 3-SAT, we create two corresponding instances of MULSETPC to deal with our two separate cases: $(\{T_1(B), T_2(B)\}, \mathcal{X})$ and $(\{T_1(S), T_2(S)\}, \mathcal{X})$. Note that $T_1(B)$, $T_2(B)$, $T_1(S)$ and $T_2(S)$ have multiplicity 2; each leaf label $C_j^x \in \mathcal{P}$ appears once in $T_{1,L,j}$ or $T_{2,L,j}$ and once in $T_{1,R,i}(B)$, $T_{1,R,i}(S)$, $T_{2,R,i}(B)$ or $T_{2,R,i}(S)$, since $C_j^x \in H(l_i) \cup H(\bar{l}_i)$ for a single value of i. By inspection, the labels of $\mathcal{X} - \mathcal{P}$ also appear at most twice. Hence the instance $(\{T_1(B), T_2(B)\}, \mathcal{X})$ contains two binary MUL-trees with multiplicity 2, and the instance $(\{T_1(S), T_2(S)\}, \mathcal{X})$ contains two MUL-trees with height 5 and multiplicity 2. It suffices to now show the reduction, in two parts.

Lemma 7 *If C is a satisfiable instance of 3-SAT then the corresponding instances $(\{T_1(B), T_2(B)\}, \mathcal{X})$ and $(\{T_1(S), T_2(S)\}, \mathcal{X})$ of* MULSETPC *admit a perfect pruning giving a consistent set of trees.*

Proof Suppose that C is satisfiable. Then there exists a valuation of every variable which satisfies every clause of C; fix one such valuation and label it $\mathcal{V}$. For each clause C_j, *mark* one of the position labels C_j^x for $x \in \{1, 2, 3\}$ such that $h(C_j^x)$ is valued true by $\mathcal{V}$. Since $\mathcal{V}$ satisfies every clause, we will always be able to choose a label to mark; if there are multiple legitimate choices choose arbitrarily. Refer to any label C_j^x we have not marked as *unmarked*.

By our construction of $T_1(B)$ and $T_2(B)$, if we show that after pruning each $T_{1,L,j}$ and $T_{1,R,i}(B)$ is leaf-label isomorphic to $T_{2,L,j}$ and $T_{2,R,i}(B)$ respectively, then $T_1(B)$ is leaf-label isomorphic to $T_2(B)$. A similar argument holds for $T_1(S)$ and $T_2(S)$.

Consider first the labels of $\mathcal{P}$. Prune from each $T_{1,L,j}$ and $T_{2,L,j}$ the one marked label C_j^x, and leave the two unmarked labels C_j^x. We must keep the other copy of

the marked C_j^x and prune away the other copies of the unmarked C_j^x in whichever $T_{1,R,i}(B)$ and $T_{2,R,i}(B)$ they appear. (A similar argument holds for $T_{1,R,i}(S)$ and $T_{2,R,i}(S)$.) Consider the sets $H(l_i)$ and $H(\bar{l}_i)$. If C_j^x is marked, then $h(C_j^x)$ is true, and so at most one of $H(l_i)$, $H(\bar{l}_i)$ contains a marked label. Keeping this information in mind, we can now simply look at the subtrees themselves.

- After this pruning each $T_{1,L,j}$ will be leaf label isomorphic to the corresponding $T_{2,L,j}$, by inspection.
- Consider $T_{1,R,i}(B)$ and $T_{2,R,i}(B)$. We have already pruned away unmarked labels of $\mathcal{P}$. If l_i is true, prune the copy of l_i in $T_{1,R,i}(B)$ closest to the root of $T_{1,R,i}(B)$ and the copy of $\bar{l}_i$ furthest from the root; if $\bar{l}_i$ is true do the opposite. In $T_{2,R,i}(B)$ prune the copies of F_i, V_i closer to the literal l_i or $\bar{l}_i$ which evaluates as false. Hence we have pruned away the extra copies of each leaf label. It is clear, mostly by inspection, that $T_{1,R,i}(B)$ is leaf-label isomorphic to $T_{2,R,i}(B)$; the most important point is that since at most one of $H(l_i)$ and $H(\bar{l}_i)$ contains a marked label, at least one of these sets has been pruned away entirely in $T_{2,R,i}(B)$ (specifically the set closer to the false literal).
- Consider $T_{1,R,i}(S)$ and $T_{2,R,i}(S)$. This case is identical to the previous except that the labels of $\mathcal{P}$ are now arranged as the leaves of a star instead of the leaves of a caterpillar, but this does not alter the argument. □

See Fig. 3 for an illustration of the reduction in Lemma 7.

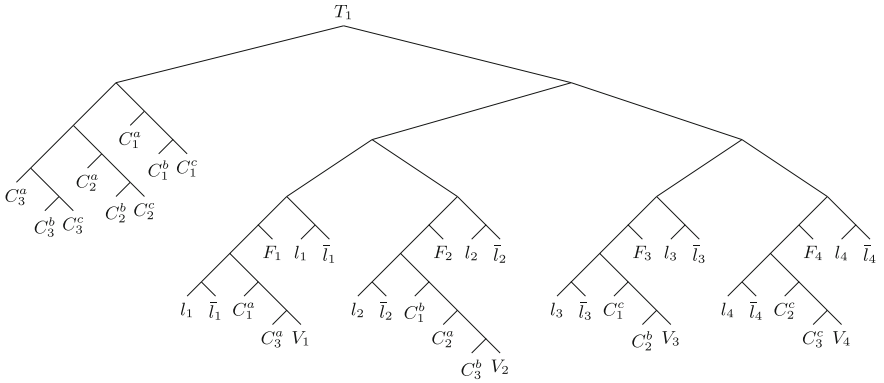
Lemma 8 *If C is not a satisfiable instance of 3-SAT then the corresponding instances $(\{T_1(B), T_2(B)\}, \mathcal{X})$ and $(\{T_1(S), T_2(S)\}, \mathcal{X})$ of* MULSETPC *do not admit perfect prunings giving a consistent set of trees.*

Proof We prove the contrapositive; that is, suppose either instance $(\{T_1(B), T_2(B)\}, \mathcal{X})$ or $(\{T_1(S), T_2(S)\}, \mathcal{X})$ does admit a perfect pruning giving a consistent set of trees. We shall show that C can be satisfied. Let T_1, T_2 denote either $T_1(B), T_2(B)$ or $T_1(S), T_2(S)$ and T_1', T_2' the MUL-trees after pruning. Recall that since T_1 and T_2 have identical sets of leaf-labels, so do T_1' and T_2', and so T_1' and T_2' must be leaf-label isomorphic to give a consistent set of trees.

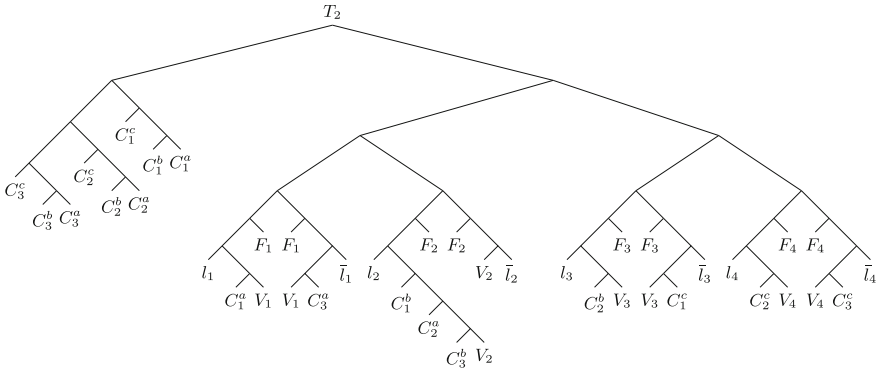
Let $T'_{1,R,i}$ denote the subtree of T_1' corresponding to $T_{1,R,i}(B)$ or $T_{1,R,i}(S)$ after we have pruned T_1 down to T_1'. Define $T'_{2,R,i}$ analogously. Note that since $T_{1,R,i}$ and $T_{2,R,i}$ are the only subtrees of their respective trees containing the leaf labels V_i, F_i, l_i and $\bar{l}_i$ neither will be pruned away entirely when constructing T_1' and T_2', so these subtrees are well-defined. For the same reason, the subtrees $T'_{1,R,i}$ and $T'_{2,R,i}$ are leaf-label isomorphic for all $i = 1, \ldots, m$. Finally, since the leaf labels V_i, l_i and $\bar{l}_i$ are not pruned, the nodes u_i, u_i^+ and u_i^- still exist in these subtrees.

Consider $T'_{2,R,i}$, and suppose that both $(T'_{2,R,i})^{u_i^-}$ and $(T'_{2,R,i})^{u_i^+}$ both contain leaf labels from $\mathcal{P}$. By inspection, $T'_{2,R,i}$ cannot be isomorphic to $T'_{1,R,i}$ as all leaf labels from $\mathcal{P}$ will be contained in $(T'_{1,R,i})^{u_i}$. Hence at most one of $(T'_{2,R,i})^{u_i^-}$ and $(T'_{2,R,i})^{u_i^+}$ contain leaf labels from $\mathcal{P}$; in the first case set $\bar{l}_i$ as true, in the second case set l_i as true. If neither case holds set the truth value of l_i arbitrarily.

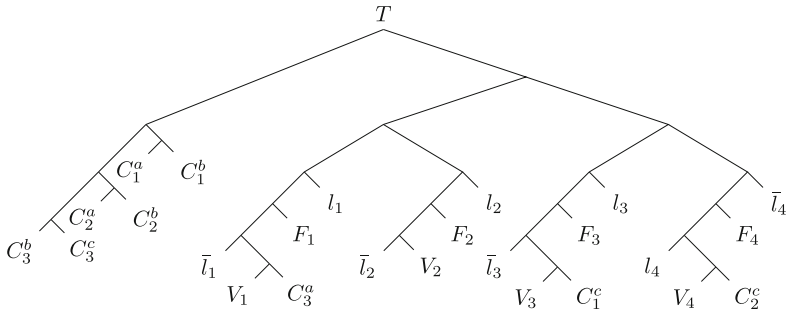
We now have a valuation (which we denote $\mathcal{V}$), it suffices to show $\mathcal{V}$ satisfies C. If C_j^x is contained in some $T'_{2,R,i}$, then $C_j^x \in H(l_i) \cup H(\bar{l}_i)$. If $C_j^x \in H(l_i)$ and C_j^x



(a) The MUL-tree T_1



(b) The MUL-tree T_2



(c) Tree T displays the two MUL-trees T_1 and T_2 after they have been perfectly pruned.

Fig. 3 Illustrating the reduction from 3-SAT in Lemma 7. The two MUL-trees T_1 and T_2 are constructed from $C = (C_1 \wedge C_2 \wedge C_3)$ where $C_1 = (l_1 \vee l_2 \vee \bar{l}_3)$, $C_2 = (l_2 \vee l_3 \vee l_4)$, and $C_3 = (\bar{l}_1 \vee l_2 \vee \bar{l}_4)$. Figure 3c shows the corresponding tree T which displays the pruned T_1 and T_2 corresponding to a satisfiable assignment $\bar{l}_1 = \bar{l}_2 = \bar{l}_3 = l_4 = true$ with marked labels C_1^c, C_2^c and C_3^a

is contained in $T'_{2,R,i}$ then C^x_j is contained in $(T'_{2,R,i})^{u_i^+}$, and so $h(C^x_j) = l_i$ which is true, and thus C_j is satisfied. A similar argument holds if $C^x_j \in H(\bar{l}_i)$.

Thus, if C_j is not satisfied, then C^x_j does not appear in any $T'_{2,R,i}$ for any $i = 1, \ldots, m$, $x = 1, 2, 3$, and because they are leaf-label isomorphic, C^x_j does not appear in $T_{1,R,i}$ either. Thus C^1_j, C^2_j, C^3_j all appear in $T'_{1,L,j}$ and $T'_{2,L,j}$. However, then $T'_{1,L,j}$ and $T'_{2,L,j}$ are not isomorphic, a contradiction. Hence we have a valuation that satisfies every C_j and hence satisfies C. □

Theorem 9 *The* MULSETPC *problem is NP-complete, even restricted to instances containing at most two binary MUL-trees with multiplicity 2.*

Theorem 10 *The* MULSETPC *problem is NP-complete, even restricted to instances containing at most two MUL-trees with multiplicity 2 and height at most 5.*

Proof of Theorems 9 and 10 Note first that MULSETPC is in NP, since given a set of pruned leaves, a perfect pruning can be constructed in polynomial time, and the consistency of the resulting set of trees determined in polynomial time using the BUILD algorithm [1], [14].

The result then follows directly from Lemma 7 and Lemma 8, using $(\{T_1(B), T_2(B)\}, \mathcal{X})$ to prove Theorem 9 or $(\{T_1(S), T_2(S)\}, \mathcal{X})$ to prove Theorem 10. □

4 NP-completeness for MULSETPComp

In this section, we extend our previous results regarding the problem MULSETPC to the similar problem MULSETPComp. We present the following two theorems.

Theorem 11 MULSETPComp *is NP-complete, even when restricted to instances containing at most two MUL-trees with multiplicity at most 3 and where the MUL-trees have height at most 4.*

Theorem 12 MULSETPComp *is NP-complete, even when restricted to instances containing at most two MUL-trees with height at most 3 and where one MUL-tree is a single-labelled tree containing all leaf labels.*

As was the case in Sect. 3, these two results have very similar proofs, and hence we shall prove both results simultaneously as we did previously. Here, we reduce from X3C3, also known to be NP-complete [19]. Recall that an *exact cover* of a set is a cover in which every element appears exactly once.

Exact 3-cover with multiplicity 3 (X3C3) Problem:
Input: A set $X = \{x_1, \ldots, x_{3q}\}$ and a collection $C = \{S_1, \ldots, S_k\}$ of 3-element subsets of X, such that any element of X appears in at most three sets in C.
Output: ∃? an exact cover for X.

As before, our first goal is to construct, given an instance (X, C) of X3C3, a set of two MUL-trees we shall use to construct our corresponding instance of MULSETPComp.

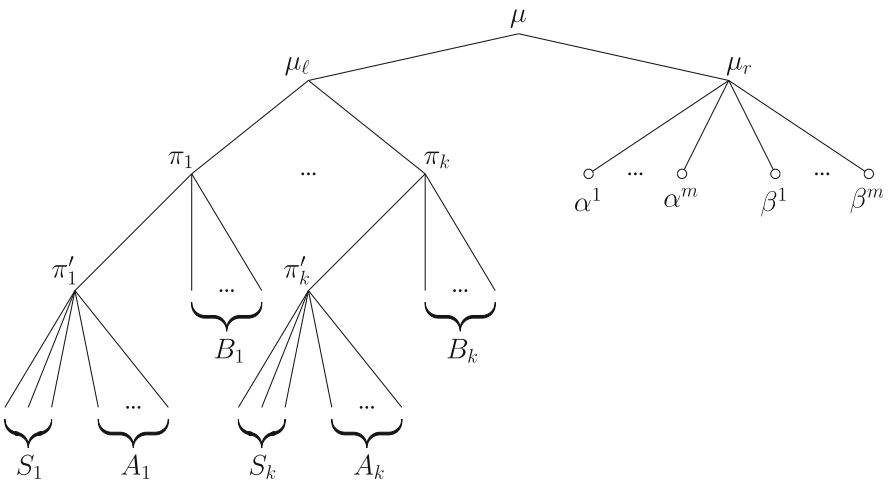


Fig. 4 The tree $T_1(M)$ for MULSETPComp

Recall that $|X| = 3q$ and that $k := |C|$. Let $m := k - q$, the number of sets of C *not* chosen to be part of our exact 3 cover. Define $A := \{a^i_j | i = 1, \ldots, m,\ j = 1, \ldots, k\}$ and $B = \{b^i_j | i = 1, \ldots, m,\ j = 1, \ldots, k\}$. Let $A^i = \{a^i_j | j = 1 \ldots, k\}$ and $A_j = \{a^i_j | i = 1, \ldots, m\}$, and define B^i, B_j similarly. The labels in $A \cup B$ are another set of "dummy" labels we use for technical reasons. Let $Y = X \cup A \cup B$, and $Y_1 = X \cup A_1 \cup B_1$. We may assume that $q \geq 3$. We may also assume that k is even; if not, add to X three additional elements $\{x_{3q+1}, x_{3q+2}, x_{3q+3}\}$ (which increases q by 1) and add to C a additional set $S' = \{x_{3q+1}, x_{3q+2}, x_{3q+3}\}$ (which increases $k = |C|$ by 1). It is clear this modified instance contains an exact 3-cover if and only if the original instance did. (If the original instance has an exact 3-cover simply add S' to get an exact 3-cover of the modified instance. Any exact 3-cover of the modified instance must contain S' to cover $\{x_{3q+1}, x_{3q+2}, x_{3q+3}\}$, so removing S' gives an exact 3-cover of the original instance.)

We denote our four trees $T_1(M), T_2(M), T_1(U)$ and $T_2(U)$. Our corresponding instance of MULSETPComp for the low *multiplicity* result Theorem 11 will be $(\{T_1(M), T_2(M)\}, Y)$, and our corresponding instance for Theorem 12, in which there exists a tree containing each label *uniquely*, will be $(\{T_1(U), T_2(U)\}, Y_1)$.

See Figs. 4 and 6 for the construction of $T_1(M)$, and Figs. 5 and 7 for the construction of $T_2(M)$. Note that a node labelled α^i in Fig. 4 is the root of the appropriate subtree from Fig. 6, not a leaf labelled by α^i. An equivalent statement holds for $\bar{\alpha}^i$, β^i and $\bar{\beta}^i$, where the subtree rooted at β^i is found by taking the subtree of Fig. 6 and replacing each a^i_j with b^i_j. (An equivalent statement holds for $\bar{\beta}^i$ and Fig. 7). See Fig. 8 for $T_1(U)$ and Fig. 9 for $T_2(U)$. Note the following other useful facts about our construction:

- Trees $T_1(M)$ and $T_2(M)$ have the same set of leaf labels. Hence a perfect pruning of $T_1(M)$ and $T_2(M)$ is a compatible set of trees if and only if there exists a tree $T^*(M)$ on the same leaf label set which is a refinement of both perfect prunings. An equivalent result holds for $T_1(U)$ and $T_2(U)$.

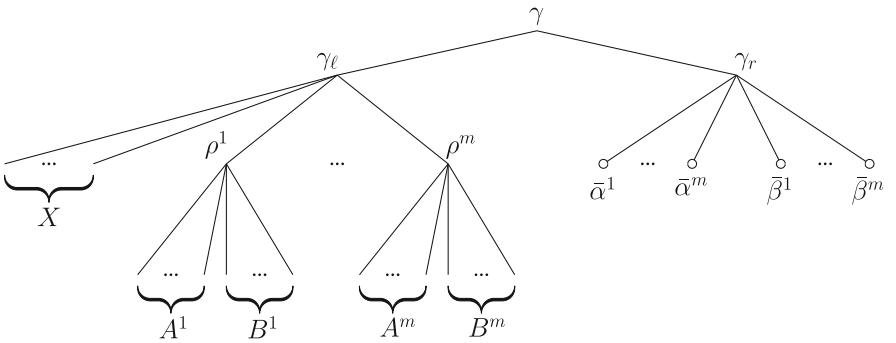


Fig. 5 The tree $T_2(M)$ for MULSETPComp

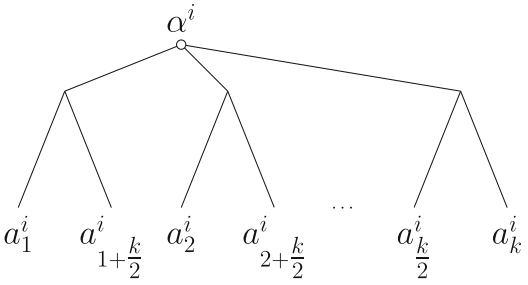


Fig. 6 The subtree of $T_1(M)$ rooted by α^i

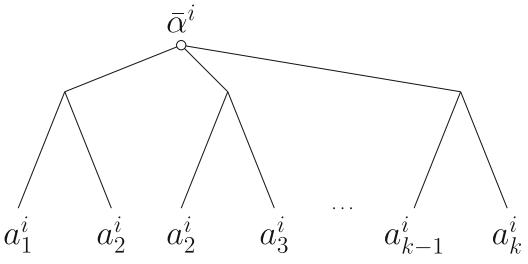


Fig. 7 The subtree of $T_2(M)$ rooted by $\bar{\alpha}^i$

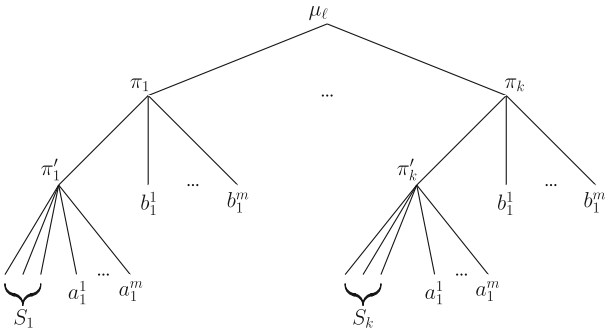


Fig. 8 The tree $T_1(U)$ for MULSETPComp

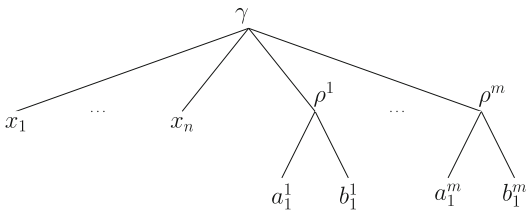


Fig. 9 The tree $T_2(U)$ for MULSETPComp

- In $T_1(M)$ any label of X may appear at most three times, since any x_i may appear in at most three sets of C. Every label of A and B appears once in $T_1(M)^{\mu_\ell}$ and once in $T_1(M)^{\mu_r}$. Hence $T_1(M)$ has multiplicity 3. In $T_2(M)$ any label of X appears only once, and the labels of A and B appear at most thrice, once in $T_2(M)^{\gamma_\ell}$ and once or twice in $T_2(M)^{\gamma_r}$. Hence $T_2(M)$ has multiplicity 3.
- $T_2(U)$ contains every leaf label of Y_1 exactly once. Note that $T_1(U)$ is created from $T_1(M)^{\mu_l}$, replacing each A_i, B_i with A_1, B_1 respectively.
- The heights of all MUL-trees can be determined by inspection.

We will need the following two technical lemmas.

Lemma 13 *Consider a set of elements $X = \{x_1, \ldots x_k\}$ together with a subset $X' \subset X$ such that $|X'| = k-1$ and a collection of sets $\mathcal{X} = \{x_i, x_{i+i}\}_{i\in[k-1]}$. Then it is possible to construct a set equal to X' by choosing one element from each set in $\mathcal{X}$.*

Proof Since $|X'| = k - 1$, let x_j denote the one element of $X' - X$. For every i such that $i < j$, choose the element x_i from $\{x_i, x_{i+i}\}$. For every i such that $i \geq j$, instead choose the element x_{i+1}. Clearly this constructed set will be equal to X', as required. □

Lemma 14 *Let T, T^* be single-labelled trees such that $T \leq T^*$, and let u, x, y be leaf nodes in T (and hence also in T^*). If $\mathrm{lca}_T(u, x) < \mathrm{lca}_T(u, y)$ then $\mathrm{lca}_{T^*}(u, x) < \mathrm{lca}_{T^*}(u, y)$.*

Proof Suppose for the sake of obtaining a contradiction that $\mathrm{lca}_T(u, x) < \mathrm{lca}_T(u, y)$ but $\mathrm{lca}_{T^*}(u, x) \not< \mathrm{lca}_{T^*}(u, y)$. Since $\mathrm{lca}_{T^*}(u, x)$ and $\mathrm{lca}_{T^*}(u, y)$ both lie on the unique path from u to the root of T^*, it follows that $\mathrm{lca}_{T^*}(u, y) \leq \mathrm{lca}_{T^*}(u, x)$. This property is maintained even after (repeated) edge contractions, and hence $\mathrm{lca}_T(u, y) \leq \mathrm{lca}_T(u, x)$, contradicting $\mathrm{lca}_T(u, x) < \mathrm{lca}_T(u, y)$. □

The following two lemmas form the core of our main result.

Lemma 15 *If (X, C) is an instance of X3C3 that allows an exact 3-cover C' then the corresponding instances $(\{T_1(M), T_2(M)\}, Y)$ and $(\{T_1(U), T_2(U)\}, Y_1)$ of* MULSETPComp *admit a perfect pruning giving a compatible set of trees.*

Proof Let $\mathcal{I} \subset [k]$ denote the set of m indices i of those S_i we did not choose as part of our exact 3-cover, that is the sets $S_i \in C - C'$. Let $\psi : \mathcal{I} \to [m]$ be an arbitrary bijection. Prune the trees $T_1(M)$ and $T_1(U)$ as follows:

- For each S_i (or equivalently, each subtree rooted at π_i):

  - If $S_i \in C'$ then keep the labels of S_i as the children of π_i' but prune away all other leaf labels in the subtrees $T_1(M)^{\pi_i}$ and $T_1(U)^{\pi_i}$. After pruning there are three leaves (with labels corresponding to the elements of S_i) as children of π_i', which is itself a child of μ_ℓ.
  - If $S_i \notin C'$ then prune away all leaf labels in $T_1(M)^{\pi_i}$ except $a_i^{\psi(i)}$ and $b_i^{\psi(i)}$, and all leaf labels in $T_1(U)^{\pi_i}$ except $a_1^{\psi(i)}$ and $b_1^{\psi(i)}$. After pruning, the remaining leaves will be children of π_i.

- This completes the pruning of $T_1(U)$; it only remains to complete the pruning of $T_1(M)$. In each $T_1(M)^{\alpha^j}$, prune away a_i^j if the leaf label appears in $T_1(M)^{\mu_\ell}$. Since m sets of C are not in C', there are m leaf labels of the form a_i^j appearing in $T_1(M)^{\mu_\ell}$, specifically $a_i^{\psi(i)}$ for the m values of i for which $\psi(i)$ is defined, or equivalently for all $\psi(i) = 1, \ldots, m$. Hence we must prune away one leaf label in each $T_1(M)^{\alpha^j}$.
- Repeat the previous step for each $T_1(M)^{\beta^j}$ – the argument is identical.

Denote the pruned versions of $T_1(M)$, $T_1(U)$ by $T_1(M)'$ and $T_1(U)'$. Every leaf label of X appears once in each pruned tree; this follows directly from C' being an exact 3 cover. All other leaf labels appear only once by inspection; hence this is a perfect pruning.

Since $T_2(U)$ is already a single-labelled tree, $T_2(U)' = T_2(U)$. We now prune $T_2(M)$ as follows:

- For each $T_2(M)^{\rho^j}$, prune away all leaf labels except $a^j_{\psi^{-1}(j)}$ and $b^j_{\psi^{-1}(j)}$; after pruning ρ^j has two children, both leaves.
- For each $T_2(M)^{\bar{\alpha}^j}$, we wish to prune this subtree to create a $(k-1)$ leaf star with all labels a_i^j for our fixed j except $a^j_{\psi^{-1}(j)}$. This is possible due to Lemma 13; from each pair of leaf labels rooted by a child of $\bar{\alpha}^j$ we pick one leaf to keep and one to prune away such that we keep one copy of everything except $a^j_{\psi^{-1}(j)}$.
- Repeat the previous step for each $T_2(M)^{\bar{\beta}^j}$ – as before the argument is identical.

There is one copy of each leaf label of X in $T_2(M)$, which we do not prune. We prune the leaf labels of $A \cup B$ in $T_2(M)$ so that the leaf labels of $A \cup B$ in $T_2(M)^{\gamma_r}$ are exactly those that do not appear in $T_2(M)^{\gamma_\ell}$. Hence this is a perfect pruning, which we denote $T_2(M)'$.

We now show that $T_2(M)'$ can be constructed from $T_1(M)'$ by repeated non-leaf edge contractions, which will show $\{T_1(M)', T_2(M)'\}$ form a compatible set (with $T(M)^* := T_1(M)'$). We also perform the equivalent construction for $T_2(U)'$.

In $T_1(M)'$ and $T_1(U)'$ contract every remaining π_i' (one for each of the q sets $S_i \in C'$) into its parent μ_ℓ. Furthermore in $T_1(M)'$, in each $T_1(M)^{\alpha^i}$ and $T_1(M)^{\beta^i}$, contract every non-leaf edge to create a star rooted at α_i or β_i. By inspection, we can see $T_2(M)' \leq T_1(M)'$ and $T_2(U)' \leq T_1(U)'$, giving our required compatible sets of trees. □

See Figs. 10, 11, 12, and 13 for an example of a pruning as in Lemma 15.

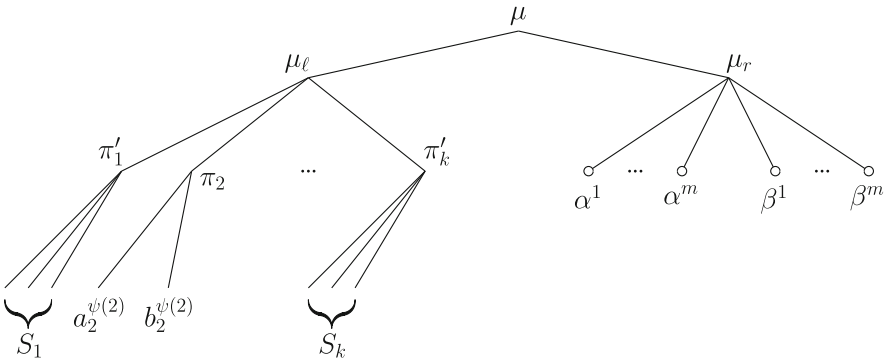


Fig. 10 As an illustrated example, a possible perfect pruning $T_1(M)'$ as in Lemma 15. In this example, $S_1, S_k \in C'$, but $S_2 \notin C'$

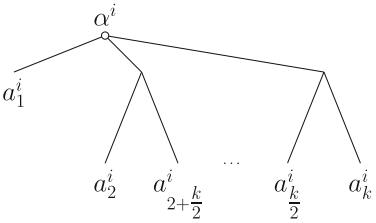


Fig. 11 A subtree of $T_1(M)'$ in the pruning from Fig 10. In this example, $\psi(1+\frac{k}{2}) = i$

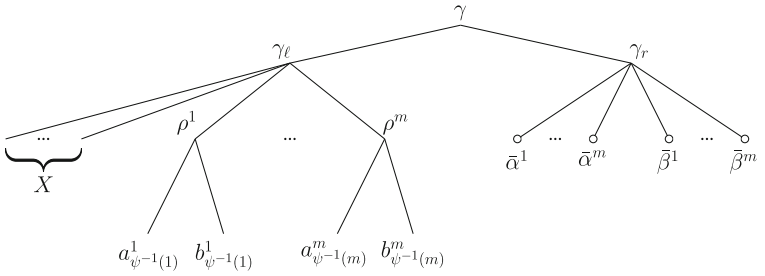


Fig. 12 A possible perfect pruning $T_2(M)'$, corresponding to the perfect pruning of Fig. 10

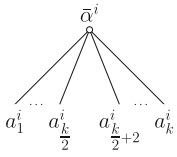


Fig. 13 A subtree of $T_2(M)'$ in the pruning from Fig. 12. Again, here $\psi^{-1}(i) = \frac{k}{2} + 1$

Lemma 16 *If (X, C) is an instance of X3C3 that does not allow an exact 3-cover C' then the corresponding instances $(\{T_1(M), T_2(M)\}, Y)$ and $(\{T_1(U), T_2(U)\}, Y_1)$ of* MULSETPComp *do not admit perfect prunings giving a compatible set of trees.*

Proof We prove the contrapositive: suppose that either $(\{T_1(M), T_2(M)\}, Y)$ or $(\{T_1(U), T_2(U)\}, Y_1)$ admits a perfect pruning giving a compatible set of trees. We

shall construct an exact 3-cover $C' \subset C$. Let T_1', T_2' denote our perfect pruning of either $T_1(M), T_2(M)$ or $T_1(U), T_2(U)$. Let T^* denote the tree that is a refinement of both T_1', T_2'.

Claim 17 Suppose that T_1', T_2' are perfect prunings of $T_1(M), T_2(M)$ respectively. For each i, at least one label in A^i and one label in B^i is in $(T_2')^{\gamma_\ell}$.

Proof We prove this result for A^i; the result follows equivalently for B^i. Suppose otherwise there is no label of A^i in $(T_2')^{\gamma_\ell}$, that is, all of A^i is contained in $(T_2')^{\gamma_r}$, and hence $(T_2')^{\bar{\alpha}^i}$. Since $k \geq q \geq 3$, any perfect pruning (T_2') which has k leaves in $(T_2')^{\bar{\alpha}^i}$ must have at least two leaves which are siblings and distance 2 from $\bar{\alpha}^i$. Denote these labels a^i_j and a^i_{j+1}. Define $j' = j + \frac{k}{2}$ if $j \leq \frac{k}{2}$ and $j' = j - \frac{k}{2}$ if $j > \frac{k}{2}$. Since $k \geq 3$, it follows $j' \neq j + 1$. Since a^i_j and $a^i_{j'}$ are not siblings, it follows that $\text{lca}_{T_2'}(a^i_j, a^i_{j+1}) < \text{lca}_{T_2'}(a^i_j, a^i_{j'})$

Since T^* is a refinement of both T_1' and T_2', and $(T_1')^{\mu_\ell}$ and $(T_2')^{\gamma_\ell}$ must contain all leaf labels from X, it follows that some subtree of T^* is a refinement of both $(T_1')^{\mu_\ell}$ and $(T_2')^{\gamma_\ell}$. As a result, there must be another subtree of T^* which is a refinement of both $(T_1')^{\mu_r}$ and $(T_2')^{\gamma_r}$. Since of A^i is contained in $(T_2')^{\gamma_r}$, it follows all of A^i is contained in $(T_1')^{\mu_r}$ and hence in $(T_1')^{\alpha^i}$. Since $T_1(M)^{\alpha^i}$ contains exactly one copy of each label in A^i, it follows we cannot have pruned any leaves away, that is, $(T_1')^{\alpha^i} = T_1(M)^{\alpha^i}$. By inspection, $\text{lca}_{T_1'}(a^i_j, a^i_{j'}) < \text{lca}_{T_1'}(a^i_j, a^i_{j+1})$, since a^i_j and $a^i_{j'}$ are siblings but a^i_j and a^i_{j+1} are not.

By Lemma 14, both of these equations must also hold T^*, giving us a contradiction. □

Suppose that T_1', T_2' are perfect prunings of $T_1(M), T_2(M)$ respectively, and consider the subtree $(T_2')^{\rho^i}$. By Claim 17 both A^i and B^i have at least one leaf label in $(T_2')^{\gamma_\ell}$, these leaf labels must exist in $(T_2')^{\rho^i}$. If a^i_j is a leaf label in $(T_2')^{\rho^i}$, let $\theta(a^i_j)$ denote a leaf label of B^i in $(T_2')^{\rho^i}$. (At least one such label must exist; if there are multiple possible labels simply choose arbitrarily. The function θ may not be a bijection.) Alternatively suppose that T_1', T_2' are perfect prunings of $T_1(U), T_2(U)$. If a^i_j is a leaf label in $(T_2')^{\rho^i}$, let $\theta(a^i_j)$ denote a leaf label of B^i in $(T_2')^{\rho^i}$. (This exists by inspection, and note here $j = 1$ and $\theta(a^i_j) = b^i_j$.) Hence no matter which case we are dealing with, θ is well-defined on a^i_j in $(T_2')^{\rho^i}$.

Claim 18 No subtree $(T_1')^{\pi_j'}$ contains both a leaf label from A_j and a different leaf label from $S_j \cup A_j$.

Proof Suppose for the sake of obtaining a contradiction that $(T_1')^{\pi_j'}$ contains one leaf label from A_j, denoted a^i_j, and another label $z \in S_j \cup A_j$ such that $z \neq a^i_j$. (Note that as two leaves of π_j' exist, the node π_j' will not have been suppressed, so this subtree is well-defined.) Clearly a^i_j must also be contained in $(T_2')^{\rho^i}$; if $T_2 = T_2(M)$ this is

due to the labels of X forcing one subtree of T^* to be a refinement of both $(T_1')^{\mu_\ell}$ and $(T_2')^{\gamma_\ell}$, if $T_2 = T_2(U)$ it follows by inspection. Hence $\theta(a_j^i)$ is well-defined. Since z and a_j^i are siblings in T_1' but a_j^i and $\theta(a_j^i)$ are not, $\text{lca}_{T_1'}(a_j^i, z) < \text{lca}_{T_1'}(a_j^i, \theta(a_j^i))$.

By inspection, $\text{lca}_{T_2'}(a_j^i, \theta(a_j^i)) = \rho^i$, which has not been suppressed as it has a_j^i and $\theta(a_j^i)$ as children. If z appears in $(T_2')^{\rho^i}$, then $z \in A^i$, since $z \notin B^i$. Hence $z \in A_j \cap A^i$, and so $z = a_j^i$, contradicting our choice of z which is distinct from a_j^i. Hence z does not appear in $(T_2')^{\rho^i}$ and so $\text{lca}_{T_2'}(a_j^i, \theta(a_j^i)) < \text{lca}_{T_2'}(a_j^i, z)$. By Lemma 14, these equations give a contradiction. □

Let $R_1, \ldots, R_k$ denote the k subtrees of T_1' rooted at children of μ_ℓ. Note that the equivalent subtrees in $T_1(M)$ or $T_1(U)$ are rooted at the nodes π_j but this may no longer be true after pruning, and instead such a subtree may be rooted at π_j', may consist of a single leaf label, or may be entirely empty. (Should a subtree be entirely pruned away, we abuse notation and say that $R_j = \emptyset$ to ensure that we have exactly k subtrees.) If a subtree R_j contains at least one leaf label from S_j we say it has Type 1, otherwise it has Type 2.

There are at least m labels from A in $(T_1')^{\mu_\ell}$; this follows by Claim 17 when $T_1' = T_1(M)$ and by inspection when $T_1' = T_1(U)$. By Claim 18 each of these labels must appear in a different R_j. Also by Claim 18, no R_j containing a label from A contains a label from S_j, and so there are at least m Type 2 subtrees. Hence there are at most $k - m = q$ Type 1 subtrees.

Every Type 1 subtree R_j contains at most three labels from S_j (and thus from X) since $|S_j| = 3$. Hence the Type 1 subtrees contain in total at most $3q$ leaf labels from X. However, $|X| = 3q$, and since T_1' contains all leaf labels from X exactly once, the Type 1 subtrees must contain exactly $3q$ leaf labels from X. Thus, each Type 1 subtree must contain exactly three labels from S_j. If R_j is a Type 1 subtree, add S_j to C'. The collection C' is an exact 3-cover; if two sets of C' share an element then two R_j in T_1' have a leaf label in common, which is impossible in a perfect pruning, and the order of C' shows C' covers X, as required. □

We now prove Theorem 11 and Theorem 12.

Proof of Theorem 11 and Theorem 12 Note first that MULSETPComp is in NP, since given a set of pruned leaves, a perfect pruning can be constructed in polynomial time. The compatibility of this set of trees can be determined in polynomial time using the BUILDST algorithm [9]. The result then follows directly from Lemma 15 and Lemma 16, using either $(\{T_1(M), T_2(M)\}, Y)$ or $(\{T_1(U), T_2(U)\}, Y_1)$ for Theorem 11 and 12 respectively. □

We close this section with the following related result.

Remark MULSETPComp is NP-complete when restricted to instances with at most two MUL-trees with multiplicity 2, as long as all trees are binary.

Recall that in general any consistent set of trees is also a compatible set; in the case where all trees are binary the inverse also holds, as a binary tree cannot be refined further. This proves the remark.

5 An algorithm for MULSETPC instances with *k* binary MUL-trees where every label is unique in at least one tree

5.1 Preliminaries

In this section, we consider the instances of MULSETPC in which we are given a set of k binary MUL-trees $\mathcal{M} = T_1, \ldots, T_k$ such that every label appears uniquely in at least one tree, that is, $\bigcup_{i=1}^{k} D(T_i) = \mathcal{X}$. (Recall that $D(T_i)$ denotes the set of leaf labels that appear exactly once in T_i.) We observe that this special condition on the input makes the problem computationally tractable, as it leads to a new recursive formula for MULSETPC that can be evaluated efficiently by adapting the classic technique of dynamic programming over k-tuples of nodes previously used for the MAST problem [13, 18, 27].

While the trees in a given instance of MULSETPC are unordered, for technical reasons we treat them as having a fixed left-right ordering defined on the children of each node. This left-right ordering is defined arbitrarily on each tree. Note, however, that when we consider isomorphism between trees, we ignore this ordering; that is, we treat T_1 and T_2 as leaf-label isomorphic even if the isomorphism doesn't maintain the ordering, and given two isomorphic trees it is possible that the isomorphism switches the ordering of some of the children.

To solve MULSETPC, we need to determine if there exists a perfect pruning of each MUL-tree $T_1, \ldots, T_k$, together with a single-labelled binary tree T on $\mathcal{X}$, such that T displays the pruned T_i for every $i = 1, \ldots, k$. This is equivalent to finding a single-labelled binary tree T on $\mathcal{X}$ such that T_i displays $T\restriction_{L(T_i)}$ for every $i = 1, \ldots, k$.

Let ϵ denote a "dummy" or "empty" node, and set $T_i^{\epsilon} = T_{\emptyset}$, the empty tree. For all k-tuples of nodes $\vec{a} = (a_1, \ldots, a_k) \in \prod_{i=1}^{k} \Big(V(T_i) \cup \{\epsilon\}\Big)$, define $S(\vec{a}) = \bigcup_{i=1}^{k} \Big(D(T_i) \cap L(T_i^{a_i})\Big)$. We aim to find a binary tree T that is leaf-labelled by $S(\vec{a})$ such that $T_i^{a_i}$ displays $T\restriction_{L(T_i)}$ for $i = 1, \ldots, k$. To do this we define the following boolean function.

Definition 19 $(W(\vec{a}))$

$$W(\vec{a}) = \begin{cases} \texttt{true} & \text{if there exists a single-labelled binary tree } T \text{ leaf-labelled} \\ & \text{by } S(\vec{a}) \text{ such that } T_i^{a_i} \text{ displays } T\restriction_{L(T_i)} \text{ for all } i = 1, \ldots, k, \\ \texttt{false} & \text{otherwise.} \end{cases}$$

The instance of MULSETPC will be satisfied if and only if such a tree T exists for the k-tuple consisting of the root nodes $r_1, \ldots, r_k$, and so it is sufficient to compute $W(r_1, \ldots, r_k)$. Lemma 20 shows that the necessary condition of the existence of T is that $S(\vec{a}) \cap L(T_i) \subseteq L(T_i^{a_i})$ for $i = 1, \ldots, k$.

Lemma 20 *Let T be a binary tree leaf-labelled by $S(\vec{a})$. If $S(\vec{a}) \cap L(T_i) \nsubseteq L(T_i^{a_i})$ for some i, $T_i^{a_i}$ does not display $T\restriction_{L(T_i)}$.*

Proof Suppose that $S(\vec{a}) \cap L(T_i) \not\subseteq L(T_i^{a_i})$, then there exists $u \in S(\vec{a}) \cap L(T_i)$ such that $u \notin L(T_i^{a_i})$. Since $T\restriction_{L(T_i)}$ is labelled by $S(\vec{a}) \cap L(T_i)$, we have that $T_i^{a_i}$ cannot display $T\restriction_{L(T_i)}$. □

Given a non-leaf node $a_i \in V(T_i)$, let a_i^ℓ and a_i^r denote the left and right children of a_i respectively. For each $i = 1, \ldots, k$ and $a_i \in V(T_i)$, we define a set $P(a_i)$ by $P(a_i) = \{\epsilon, a_i\}$ when a_i is a leaf and $P(a_i) = \{\epsilon, a_i, a_i^\ell, a_i^r\}$ otherwise. Finally, if $a_i = \epsilon$ set $P(a_i) = \{\epsilon\}$. For each $a_i \in V(T_i)$ define a function c^{a_i} as follows:

$$c^{a_i}(\epsilon) = a_i, \qquad \text{and} \qquad c^{a_i}(a_i) = \epsilon.$$

When a_i is a non-leaf (and thus a_i^ℓ and a_i^r are defined), also let

$$c^{a_i}(a_i^r) = a_i^\ell, \qquad \text{and} \qquad c^{a_i}(a_i^\ell) = a_i^r.$$

Alternatively, if $a_i = \epsilon$, let

$$c^{\epsilon}(\epsilon) = \epsilon.$$

Hence c^{a_i} is an involution on $P(a_i)$ which associates each $x \in P(a_i)$ with a complement. Let $\Pi(\vec{a}) = \prod_{i=1}^k P(a_i) - \{(\epsilon, \ldots, \epsilon), (a_1, \ldots, a_k)\}$, and note that $|\Pi(\vec{a})| < 4^k$.

We define a partial ordering $\leq$ on $\prod_{i=1}^k (V(T_i) \cup \{\epsilon\})$ by taking $(a_1', \ldots, a_k') \leq (a_1, \ldots, a_k)$ if and only if $a_i' \leq a_i$ for all $i = 1, \ldots, k$. Also, $(a_1', \ldots, a_k') < (a_1, \ldots, a_k)$ if and only if $(a_1', \ldots, a_k') \leq (a_1, \ldots, a_k)$ and $a_i' < a_i$ for some i. We treat ϵ as being a minimal node, that is, $\epsilon < a_i$ for all $a_i \in V(T_i)$. Hence the unique $\leq$-minimum element is $(\epsilon, \ldots, \epsilon)$.

Example 21 Let T_1, T_2, and T_3 in Fig. 14 form an instance of MULSETPC.

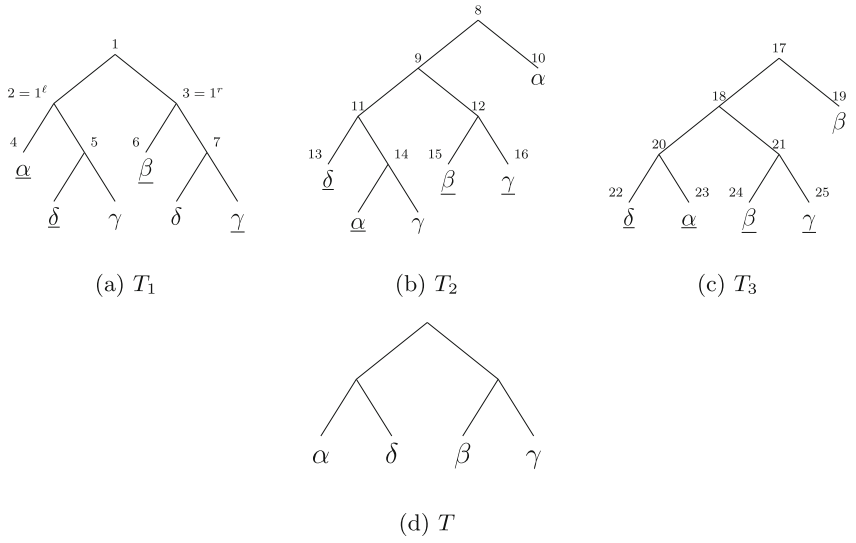
1. Consider the 3-tuple $(1, 8, 17)$, consisting of the roots of the three trees. Note that $S(1, 8, 17) = \{\alpha, \beta, \gamma, \delta\}$ and that $T\restriction_{L(T_1)} = T\restriction_{L(T_2)} = T\restriction_{L(T_3)} = T$ as $L(T_1) = L(T_2) = L(T_3) = L(T)$. Thus $W(1, 8, 17) = \texttt{true}$ since (by inspection) T_1^1, T_2^8 and T_3^{17} all display T.
2. Consider the 3-tuple $(2, 11, 18)$. By inspection (specifically of T_3^{18}) we see that $S(2, 11, 18) = \{\alpha, \gamma, \delta\}$. For T_1^2 to display some single-labelled T labelled by $S(2, 11, 18)$, it must be that $T = T_1^2$, since T_1^2 and T will be single-labelled trees on the same set of labels. By the same argument, $T_2^{11} = T$. However, T cannot be isomorphic to both trees. Hence no such T exists and so $W(2, 11, 18) = \texttt{false}$.

□

Our main result in this section is an algorithm that efficiently determines $W(\vec{a})$. Fundamental to these efforts is the following recursive formula $Z(\vec{a})$, which gives a convenient way of evaluating the function $W(\vec{a})$.

Definition 22 $(Z(\vec{a}))$ $Z(\vec{a})$ is defined as follows:

- If $S(\vec{a}) \cap L(T_i) \not\subseteq L(T_i^{a_i})$ for some i, $Z(\vec{a}) = \texttt{false}$.
- Otherwise, if $|S(\vec{a})| \leq 1$, $Z(\vec{a}) = \texttt{true}$.



(a) T_1 (b) T_2 (c) T_3

(d) T

Fig. 14 Figures for Examples 21 and 23. The underlined labels are those retained after pruning T_1, T_2 and T_3 down to obtain T. Note that $D(T_1) = \{\alpha, \beta\}$, $D(T_2) = \{\beta, \delta\}$ and $D(T_3) = \{\alpha, \gamma, \delta\}$, so every label appears uniquely in at least one tree

- Otherwise,

$$Z(\vec{a}) = \bigvee_{(x_1,\ldots,x_k)\in\Pi(\vec{a})} \Big(Z(x_1,\ldots,x_k) \wedge Z(c^{a_1}(x_1),\ldots,c^{a_k}(x_k)) \Big)$$

Example 23 Consider again the trees T_1, T_2, and T_3 in Fig. 14.

1. We show $Z(1, 8, 17) = \texttt{true}$ as follows. First we note that $S(1, 8, 17) = \{\alpha, \beta, \gamma, \delta\}$ and that $L(T_i) = L(T_i^{r_i})$ for $i = 1, 2, 3$, where r_i is the root of T_i. Hence $Z(1, 8, 17) = \texttt{true}$ if and only if there exists a 3-tuple in $\Pi(1, 8, 17)$ such that Z evaluates to $\texttt{true}$ on both the 3-tuple and its complement. Consider the 3-tuple $(1, 9, 18)$ and complement $(\epsilon, 10, 19)$. Since $S(\epsilon, 10, 19) = \emptyset$, it follows that $Z(\epsilon, 10, 19) = \texttt{true}$. To show that $Z(1, 9, 18) = \texttt{true}$, determine $Z(\vec{x})$ where $\vec{x} \in \{(4, 14, 23), (5, 13, 22), (6, 15, 24), (7, 16, 25), (2, 11, 20), (3, 12, 21)\}$. For each of these 3-tuples $Z(\vec{x}) = \texttt{true}$. (We omit the calculation, but note for each $\vec{x}$ either $Z(\vec{x}) = \texttt{true}$ trivially or $Z(\vec{x}) = \texttt{true}$ due to two 3-tuples of $\Pi(\vec{x})$ which appear earlier in the sequence.) Hence $Z(1, 9, 18) = Z(2, 11, 20) \wedge Z(3, 12, 21) = \texttt{true}$ and so $Z(1, 8, 17) = \texttt{true}$.
2. We now show $Z(2, 11, 18) = \texttt{false}$. As $S(2, 11, 18) = \{\alpha, \gamma, \delta\}$, it follows $S(2, 11, 18) \subseteq L(T_1^2), L(T_2^{11}), L(T_3^{18})$, and so if we wish to show $Z(2, 11, 18) = \texttt{false}$ we must show that for every $\vec{x} \in \Pi(2, 11, 18)$ either $Z(\vec{x}) = \texttt{false}$ or $Z(\vec{y}) = \texttt{false}$ (where $\vec{y}$ is the complement of $\vec{x}$). Let $(x_1, x_2, x_3) := \vec{x}$. Without loss of generality suppose that $x_3 = 18$ or 20. (If not, simply swap $\vec{x}$ and $\vec{y}$.) Hence $\delta, \alpha \in S(\vec{x}) \cap L(T_1)$. If $x_1 \neq 2$ then either δ or α is not an

element of $L(T_1^{x_1})$ and thus $S(\vec{x}) \cap L(T_1) \not\subseteq L(T_1^{x_1})$ and $Z(\vec{x}) = \texttt{false}$. Hence we may suppose that $x_1 = 2$ and, by an equivalent argument, that $x_2 = 11$. Since $(2, 11, 18) \notin \Pi(2, 11, 18)$, we may assume $\vec{x} = (2, 11, 20)$. However, then $\vec{y} = (\epsilon, \epsilon, 21)$, and $Z(\epsilon, \epsilon, 21) = \texttt{false}$. Hence for all choices of $\vec{x}$, either $Z(\vec{x}) = \texttt{false}$ or $Z(\vec{y}) = \texttt{false}$, and so $Z(2, 11, 18) = \texttt{false}$.

Finally, note $Z(1, 8, 17) = W(1, 8, 17)$ and $Z(2, 11, 18) = W(2, 11, 18)$. □

5.2 Proof that $W(\vec{a}) = Z(\vec{a})$

The following two lemmas prove that indeed $W(\vec{a}) = Z(\vec{a})$ for every $\vec{a} = (a_1, \ldots, a_k) \in \prod_{i=1}^{k}(V(T_i) \cup \{\epsilon\})$.

Lemma 24 *Let $T_1, T_2, \ldots, T_k$ be a collection of k binary MUL-trees such that $\bigcup_{i=1}^{k} D(T_i) = \mathcal{X}$. If $Z(\vec{a}) = \texttt{true}$ then $W(\vec{a}) = \texttt{true}$.*

Proof We prove this by strong induction on the partial ordering $\leq$ of the k-tuples. Consider the base case, the unique minimal k-tuple $(\epsilon, \ldots, \epsilon)$. As $S(\epsilon, \ldots, \epsilon) = \emptyset$, we set $T = T_\emptyset$ and note $T_i^\epsilon = T_\emptyset$ displays $T \restriction_{L(T_i)}$ trivially for all $i = 1, \ldots, k$. Thus $W(\epsilon, \ldots, \epsilon) = \texttt{true}$.

Now suppose that $Z(\vec{a}) = \texttt{true}$. Then $S(\vec{a}) \cap L(T_i) \subseteq L(T_i^{a_i})$ for all $i = 1, \ldots, k$, and either *(i)* $|S(\vec{a})| \leq 1$ or *(ii)* there is some $\vec{x} \in \Pi(\vec{a})$ such that $Z(\vec{x}) \wedge Z(\vec{y}) = \texttt{true}$ (where $\vec{y}$ is the complement of $\vec{x}$).

Case (i): If $S(\vec{a}) = \emptyset$ then set $T = T_\emptyset$, which is trivially displayed by $T_i^{a_i}$ for all i. Otherwise, if $S(\vec{a}) = \{u\}$, let T to be the singleton tree with its single leaf labelled by u. Now, if $u \in L(T_i^{a_i})$ then $T_i^{a_i}$ displays $T \restriction_{L(T_i)} = T$. Alternatively, if $u \notin L(T_i^{a_i})$ then $u \notin S(\vec{a}) \cap L(T_i)$ and so $u \notin L(T_i)$. Hence $T_i^{a_i}$ displays $T \restriction_{L(T_i)} = T_\emptyset$ trivially.

Case (ii): As $\vec{x}, \vec{y} < \vec{a}$, by induction it follows that $W(\vec{x}) \wedge W(\vec{y}) = \texttt{true}$ and hence that there exist T_x and T_y such that $T_i^{x_i}$ displays $T_x \restriction_{L(T_i)}$ and $T_i^{y_i}$ displays $T_y \restriction_{L(T_i)}$ for all i. We will need the following claim.

Claim 25 *$S(\vec{a})$ is a disjoint union of $S(\vec{x})$ and $S(\vec{y})$.*

Proof Since $L(T_i^{x_i}), L(T_i^{y_i}) \subseteq L(T_i^{a_i})$ for all possible x_i, y_i, it follows that $S(\vec{x}), S(\vec{y}) \subseteq S(\vec{a})$. Similarly, since $L(T_i^{a_i}) = L(T_i^{x_i}) \cup L(T_i^{y_i})$ it follows that $S(\vec{a}) \subseteq S(\vec{x}) \cup S(\vec{y})$. Now suppose for the sake of obtaining a contradiction that there exists $u \in S(\vec{x}) \cap S(\vec{y})$. Hence there exist i, j such that $u \in D(T_i) \cap L(T_i^{x_i})$ and $u \in D(T_j) \cap L(T_j^{y_j})$. Since $Z(\vec{x}) \wedge Z(\vec{y}) = \texttt{true}$, it follows by induction that $S(\vec{x}) \cap L(T_j) \subseteq L(T_j^{x_j})$ and thus $u \in L(T_j^{x_j})$. As $u \in L(T_j^{y_j})$, u is not unique in $L(T_j)$ and so $u \notin D(T_j)$, a contradiction. Hence $S(\vec{x}) \cap S(\vec{y}) = \emptyset$ and so $S(\vec{a})$ is a disjoint union of $S(\vec{x})$ and $S(\vec{y})$. □

We construct T as follows. If $|L(T_x)|, |L(T_y)| \neq 0$, then construct T by taking the trees T_x and T_y and adding a new root node adjacent to the roots of T_x and T_y. If $T_y = T_\emptyset$, let $T = T_x$. Finally, if $T_x = T_\emptyset$, let $T = T_y$. (Note that since $|S(\vec{a})| \geq 2$, it is not possible that $T_x = T_y = T_\emptyset$.) Since $S(\vec{a})$ is a disjoint union of $S(\vec{x})$ and $S(\vec{y})$, T

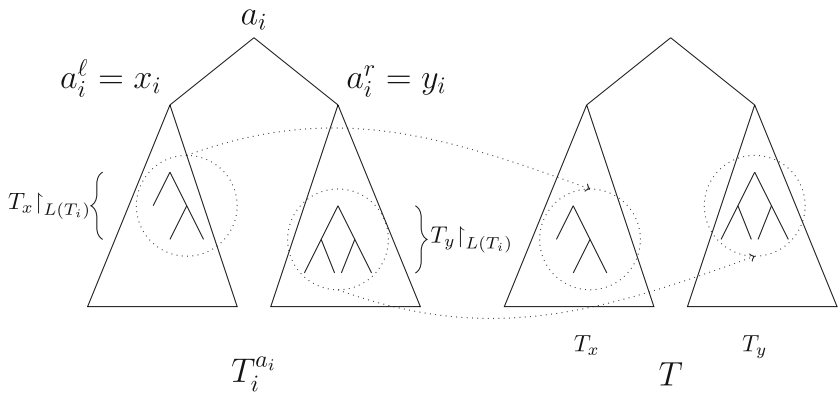


Fig. 15 Illustrating Lemma 24, Case (ii), Subcase (i). Since $T_i^{x_i}$ displays $T_x\restriction_{L(T_i)}$ and $T_i^{y_i}$ displays $T_y\restriction_{L(T_i)}$, then $T_i^{a_i}$ displays $T_x\restriction_{L(T_i)}$ and $T_y\restriction_{L(T_i)}$ connected by a root node (assuming all these trees are non-empty). Thus constructing T from T_x and T_y and a new root node connecting them is sufficient

is a single labelled tree, and is leaf-labelled by $S(\vec{a})$, as required. It remains to show that $T_i^{a_i}$ displays $T\restriction_{L(T_i)}$ for all $i = 1, \ldots, k$.

Subcase (i) – $a_i \neq \epsilon$, and either $x_i = a_i^\ell$ or $x_i = a_i^r$: Without loss of generality, let $x_i = a_i^\ell$, and so $y_i = a_i^r$. Since $x_i = a_i^\ell$, it follows that $T_i^{x_i}$ (and $T_i^{y_i}$) are both non-empty. By induction $T_i^{x_i}$ displays $T_x\restriction_{L(T_i)}$ and $T_i^{y_i}$ displays $T_y\restriction_{L(T_i)}$. If both $T_x\restriction_{L(T_i)}$ and $T_y\restriction_{L(T_i)}$ are non-empty, then $T_i^{a_i}$ displays $T\restriction_{L(T_i)}$ by inheriting the mapping from $T_i^{x_i}$ and $T_i^{y_i}$, and mapping a_i to the root of T (Fig. 15). Alternatively, if (without loss of generality) $T_y\restriction_{L(T_i)}$ is empty but $T_x\restriction_{L(T_i)}$ is not, then $T_i^{a_i}$ displays $T\restriction_{L(T_i)} = T_x\restriction_{L(T_i)}$ simply by inheriting the mapping from $T_i^{x_i}$. Finally, if both $T_x\restriction_{L(T_i)}$ and $T_y\restriction_{L(T_i)}$ are empty, then $T\restriction_{L(T_i)} = T_\emptyset$ which is displayed by $T_i^{a_i}$ trivially.

Subcase (ii) – $a_i \neq \epsilon$, and either $x_i = a_i$ or $x_i = \epsilon$: Without loss of generality, let $x_i = a_i$ and so $y_i = \epsilon$. Now $T_i^{a_i} = T_i^{x_i}$ displays $T_x\restriction_{L(T_i)} = T\restriction_{L(T_i)}$. (As $T_i^{y_i} = T_i^\epsilon = T_\emptyset$ displays $T_y\restriction_{L(T_i)}$, it follows that $T_y\restriction_{L(T_i)} = T_\emptyset$ also.)

Subcase (iii) – $a_i = \epsilon$: If $a_i = \epsilon$, it follows that $x_i = y_i = \epsilon$. Since $T_i^{x_i} = T_i^\epsilon = T_\emptyset$ displays $T_x\restriction_{L(T_i)}$ it follows that $T_x\restriction_{L(T_i)} = T_\emptyset$ and similarly $T_y\restriction_{L(T_i)} = T_\emptyset$. Thus $T_i^{a_i}$ displays $T\restriction_{L(T_i)} = T_\emptyset$.

In all subcases $T_i^{a_i}$ displays $T\restriction_{L(T_i)}$, where T is a single labelled tree leaf-labelled by $S(\vec{a})$. Thus $W(\vec{a}) = \texttt{true}$. □

Lemma 26 *Let $T_1, T_2, \ldots, T_k$ be a collection of k binary MUL-trees such that $\bigcup_{i=1}^k D(T_i) = \mathcal{X}$. If $W(\vec{a}) = \texttt{true}$ then $Z(\vec{a}) = \texttt{true}$.*

Proof We prove this by strong induction on the partial ordering $\leq$ of the k-tuples. For the base case, consider the k-tuple $(\epsilon, \ldots, \epsilon)$, the unique minimal element:

- $S(\epsilon, \ldots, \epsilon) \cap L(T_i) \subseteq L(T_i^\epsilon)$ for all i since $S(\epsilon, \ldots, \epsilon) = \emptyset$, and so the first case does not apply.

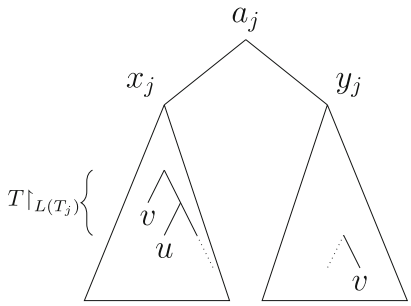


Fig. 16 The tree $T_j^{a_j}$ in Case (i) of Lemma 26, showing how $S(\vec{x}) = S(\vec{a})$ and $S(\vec{y}) = \emptyset$

- $|S(\epsilon, \ldots, \epsilon)| = 0$, and so $Z(\epsilon, \ldots, \epsilon) = \texttt{true}$ by the second case.

Hence our result holds in the base case, since $Z(\epsilon, \ldots, \epsilon)$ will always evaluate to $\texttt{true}$. Now suppose that the result holds for all $(a'_1, \ldots, a'_k) < \vec{a}$, and show that it holds also for $\vec{a}$.

As $W(\vec{a}) = \texttt{true}$, let T be the single labelled tree leaf-labelled by $S(\vec{a})$ such that $T_i^{a_i}$ displays $T\restriction_{L(T_i)}$ for each $i = 1, \ldots, k$. We show that $Z(\vec{a}) = \texttt{true}$. Consider the first two possibilities. If $S(\vec{a}) \cap L(T_j) \nsubseteq L(T_j^{a_i})$ for some j, then by Lemma 20, $T_j^{a_j}$ does not display $T\restriction_{L(T_j)}$, contradicting $W(\vec{a}) = \texttt{true}$. Also, if $|S(\vec{a})| \leq 1$, then $Z(\vec{a}) = \texttt{true}$.

Hence we may assume that $S(\vec{a}) \cap L(T_i) \subseteq L(T_i^{a_i})$ for all i and that $|S(\vec{a})| \geq 2$. It suffices to find k-tuples $\vec{x} = (x_1, \ldots, x_k)$ and $(c^{a_1}(x_1), \ldots, c^{a_k}(x_k))$ such that $\vec{x} \in \Pi(a_1, \ldots, a_k)$ and $Z(\vec{x}) \wedge Z(c^{a_1}(x_1), \ldots, c^{a_k}(x_k)) = \texttt{true}$. For simplicity, let $\vec{y} = (y_1, \ldots, y_k)$ denote the complement of $\vec{x}$, that is, let $y_i = c^{a_i}(x_i)$ for all i.

Case (i) – There exists $a_j \neq \epsilon$ such that either $T_j^{a_j^\ell}$ or $T_j^{a_j^r}$ displays $T\restriction_{L(T_j)}$: In this case define $\vec{x}$ by setting $x_i = a_i$ for all $i \neq j$, and setting $x_j = a_j^\ell$ or $x_j = a_j^r$, depending on which rooted subtree displays $T\restriction_{L(T_j)}$. Then note the complement $\vec{y}$ consists of $y_i = \epsilon$ for all $i \neq j$ and $y_j = a_j^r$ or $y_j = a_j^\ell$.

We now determine $S(\vec{x})$ and $S(\vec{y})$. It is clear that $S(\vec{x}), S(\vec{y}) \subseteq S(\vec{a})$. Suppose for the sake of obtaining a contradiction that there exists $u \in S(\vec{a})$ but $u \notin S(\vec{x})$. Since $x_i = a_i$ for all $i \neq j$, it follows that $u \in D(T_j) \cap L(T_j^{a_j})$ but $u \notin L(T_j^{x_j})$. But as $u \in S(\vec{a}) \cap L(T_j)$, u is a label of $T\restriction_{L(T_j)}$. Since $T_j^{x_j}$ displays $T\restriction_{L(T_j)}$, it follows that $u \in L(T_j^{x_j})$, a contradiction. (This is shown in Fig. 16.) Hence $S(\vec{x}) = S(\vec{a})$. Now suppose there exists $v \in S(\vec{y})$. Thus $v \in D(T_j) \cap L(T_j^{y_j})$ since $L(T_i^{y_i}) = \emptyset$ for all $i \neq j$; also $v \in S(\vec{a})$. The tree $T_j^{x_j}$ displays $T\restriction_{L(T_j)}$ which is labelled by $S(\vec{a}) \cap L(T_j)$, hence the tree $T_j^{x_j}$ must contain v. Thus v appears twice in $L(T_j^{a_j})$, once in $L(T_j^{x_j})$ and once in $L(T_j^{y_j})$. (See the label v Fig. 16.) This contradicts $v \in D(T_j)$. Hence $S(\vec{y}) = \emptyset$.

Now let $T_x := T$ which is a single labelled tree leaf-labelled by $S(\vec{x}) = S(\vec{a})$, and let $T_y := T_\emptyset$ which is a single labelled tree leaf-labelled by $S(\vec{y}) = \emptyset$. For each

$i = 1, \ldots, k$, $T_i^{x_i}$ displays $T\restriction_{L(T_i)} = T_x\restriction_{L(T_i)}$, and $T_i^{y_i}$ displays $T_\emptyset = T_y$. Hence $W(\vec{x}) \wedge W(\vec{y}) = \texttt{true}$. By induction, $Z(\vec{x}) \wedge Z(\vec{y}) = \texttt{true}$ and so $Z(\vec{a}) = \texttt{true}$.

Case (ii) – For all i either $a_i = \epsilon$, or a_i is a leaf, or a_i is a non-leaf but neither $T_j^{a_j^\ell}$ nor $T_j^{a_j^r}$ displays $T\restriction_{L(T_i)}$: As $T_i^{a_i}$ displays $T\restriction_{L(T_i)}$, it is possible to prune labels from $T_i^{a_i}$ to obtain a tree leaf-label isomorphic to $T\restriction_{L(T_i)}$. Denote the tree that remains after this pruning by Q_i for all $i = 1, \ldots, k$. As $|S(\vec{a})| \geq 2$ the root of T has two children, and so we label the disjoint, non-empty subtrees rooted at these two children T^ℓ and T^r.

Initially, let $a_i \neq \epsilon$. Suppose $a_i \notin V(Q_i)$ and a_i has two children in $T_i^{a_i}$. That is, a_i was removed from $T_i^{a_i}$ when creating Q_i – this can only occur if all of the labels from either $T_i^{a_i^\ell}$ or $T_i^{a_i^r}$ were pruned, and thus either $T_i^{a_i^r}$ or $T_i^{a_i^\ell}$ displays $T\restriction_{L(T_i)}$. This possibility is covered by the previous case. Alternatively, if $a_i \notin V(Q_i)$ and a_i does not have two children in $T_i^{a_i}$, then $T_i^{a_i}$ is a singleton tree consisting only of a_i and thus $Q_i = T_\emptyset$. Hence either $a_i \in V(Q_i)$ (and thus a_i is the root of Q_i) or $Q_i = T_\emptyset$.

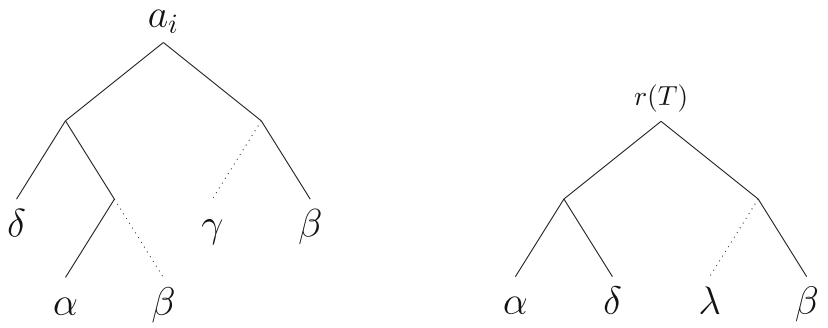
There is an isomorphic mapping from Q_i to $T\restriction_{L(T_i)}$, and under this mapping the node a_i is mapped to a node in T (as long as $Q_i \neq T_\emptyset$). This allows us to classify each $i \in \{1, \ldots, k\}$ such that $a_i \neq \epsilon$, as follows:

- If a_i maps to the root of T, let i have Type C.
- If a_i maps to a node of T^ℓ, let i have Type L.
- If a_i maps to a node of T^r, let i have Type R.
- If $Q_i = T_\emptyset$, let i have Type N.

When $a_i = \epsilon$, let i have Type E (Fig. 17, Fig. 18). It is clear that every $i \in \{1, \ldots, k\}$ has exactly one of these five types. Also note if Q_i is a singleton tree consisting of the node $a_i (\neq \epsilon)$, then since the root of T is not a leaf, i must have Type L or Type R.

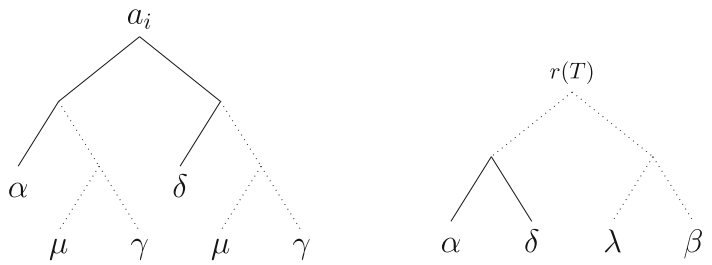
Recall we need to find k-tuple $\vec{x} \in \Pi(a_1, \ldots, a_k)$ such that $Z(\vec{x}) \wedge Z(\vec{y}) = \texttt{true}$, where $\vec{y}$ is the complement of $\vec{x}$. We use the type of each i to determine x_i (and thus also complement y_i). .

- Say i has Type C. Let u^ℓ and u^r denote the two children of a_i in Q_i. One of these is mapped to a node of T^ℓ and the other to a node of T^r. Note also the root of $T\restriction_{L(T_i)}$ also must have two children, due to the isomorphic mapping. If u^ℓ maps to a node of T^ℓ then $T_i^{a_i^\ell}$ displays $T^\ell\restriction_{L(T_i)}$. (This can be seen by inheriting the pruning of $T_i^{a_i}$ that gives Q_i, restricted only to $T_i^{a_i^\ell}$.) Similarly, $T_i^{a_i^r}$ displays $T^r\restriction_{L(T_i)}$; if instead u^ℓ maps to a node of T^r then a similar argument holds after switching the subtrees. Let $x_i = a_i^\ell$ if u^ℓ maps to T^ℓ or a_i^r otherwise. Thus $T_i^{x_i}$ displays $T^\ell\restriction_{L(T_i)}$ and $T_i^{y_i}$ displays $T^r\restriction_{L(T_i)}$.
- Say i has Type L. Let $x_i = a_i$. As a_i, and thus all of Q_i, maps to T^ℓ, pruning T to create $T\restriction_{L(T_i)}$ leads to all labels of T^r being pruned away. Hence $T_i^{x_i} = T_i^{a_i}$ displays $T\restriction_{L(T_i)} = T^\ell\restriction_{L(T_i)}$, and $T_i^{y_i} = T_i^\epsilon = T_\emptyset$ displays $T^r\restriction_{L(T_i)} = T_\emptyset$.
- Say i has Type R. Let $x_i = \epsilon$ and $y_i = a_i$. This follows equivalently to the Type L case.
- Say i has Type N. As $Q_i = T_\emptyset$, it follows that $T\restriction_{L(T_i)} = T_\emptyset$. Set $x_i = a_i$; it follows that $T_i^{x_i}$ and $T_i^{y_i}$ trivially display $T^\ell\restriction_{L(T_i)} = T^r\restriction_{L(T_i)} = T_\emptyset$.



(a) $T_i^{a_i}$ displaying Q_i. (b) T displaying $T\restriction_{L(T_i)}$.

Fig. 17 A possible $T_i^{a_i}$, Q_i, T and $T\restriction_{L(T_i)}$ for Case (ii) of Lemma 26 – the dotted leaf edges denote pruned labels. Here suppose $S(\vec{a}) = \{\alpha, \beta, \delta, \lambda\}$ but $\lambda \notin L(T_i)$. By pruning $T_i^{a_i}$ to create Q_i and T to create $T\restriction_{L(T_i)}$ we obtain our leaf-label isomorphic trees. Since a_i maps to $r(T)$, i will have Type C



(a) $T_i^{a_i}$ displaying Q_i. (b) T displaying $T\restriction_{L(T_i)}$.

Fig. 18 Another possible $T_i^{a_i}$, Q_i, T and $T\restriction_{L(T_i)}$ for Case (ii) of Lemma 26. Here suppose $S(\vec{a}) = \{\alpha, \beta, \delta, \lambda\}$ but $\beta, \lambda \notin L(T_i)$. In this case a_i maps to a node in T^ℓ, and so i will have Type L

- Say i has Type E. Set $x_i = y_i = \epsilon$; note these are the only valid choices from $P(\epsilon)$. Since $T_i^{a_i} = T_i^{\epsilon} = T_\emptyset$ displays $T\restriction_{L(T_i)}$, it follows that $T\restriction_{L(T_i)} = T_\emptyset$ and thus $T_i^{x_i}$ and $T_i^{y_i}$ display $T^\ell\restriction_{L(T_i)} = T_\emptyset$ and $T^r\restriction_{L(T_i)} = T_\emptyset$ respectively.

Thus by the above process we define $\vec{x}$ and its complement $\vec{y}$. In all cases $T_i^{x_i}$ displays $T^\ell\restriction_{L(T_i)}$ and $T_i^{y_i}$ displays $T^r\restriction_{L(T_i)}$. Two things remain to check.

Claim 27 T^ℓ is labelled by $S(\vec{x})$ and T^r is labelled by $S(\vec{y})$.

Proof Let $u \in S(\vec{x})$, and set j such that $u \in D(T_j) \cap L(T_j^{x_j})$. As $S(\vec{x}) \subseteq S(\vec{a})$, it follows that $u \in S(\vec{a})$ and thus $u \in L(T)$. Suppose for the sake of obtaining a contradiction that $u \notin L(T^\ell)$. Hence $u \in L(T^r)$ and also u is a label of $T^r\restriction_{L(T_j)}$. As $T_j^{y_j}$ displays $T^r\restriction_{L(T_j)}$, it follows that $u \in L(T_j^{y_j})$, and as $u \in L(T_j^{x_j})$, it follows that u

is not unique in T_j, contradicting $u \in D(T_j)$. Thus $u \in L(T^\ell)$, and so $S(\vec{x}) \subseteq L(T^\ell)$. Alternatively, let $v \in L(T^\ell)$. Hence $v \in L(T)$ and so $v \in S(\vec{a})$. Choose j such that $v \in D(T_j) \cap L(T_j^{a_j})$. As $v \in L(T_j)$ and $v \in L(T^\ell)$, v is a label of $T^\ell{\restriction}_{L(T_j)}$ which is displayed by $T_j^{x_j}$, and so $v \in L(T_j^{x_j})$. With $v \in D(T_j)$ this implies $v \in S(\vec{x})$ and thus $L(T^\ell) \subseteq S(\vec{x})$. A similar argument shows that T^r is labelled by $S(\vec{y})$. □

Claim 28 $\vec{x} \in \Pi(\vec{a})$. (That is, $\vec{x} \neq (\epsilon, \ldots, \epsilon), \vec{a}$.)

Proof Without loss of generality let $\vec{x} = \vec{a}$ and $\vec{y} = (\epsilon, \ldots, \epsilon)$. Hence every i has Type L or Type N (whenever $a_i \neq \epsilon$) or Type E (when $a_i = \epsilon$). Since $|S(\vec{a})| \geq 2$, we know T^r is non-empty, so let $u \in L(T^r)$. It follows from the types that $T^r{\restriction}_{L(T_i)} = T_\emptyset$ for all i, and so $u \notin L(T_i)$ for all i. However $u \in S(\vec{a})$, a contradiction. □

Set $T_x = T^\ell$, a single labelled tree leaf-labelled by $S(\vec{x})$. $T_i^{x_i}$ displays $T_x{\restriction}_{L(T_i)}$ for all i. Set $T_y = T^r$, a single labelled tree leaf-labelled by $S(\vec{y})$. $T_i^{y_i}$ displays $T_y{\restriction}_{L(T_i)}$ for all i. Thus $W(\vec{x}) \wedge W(\vec{y}) = \texttt{true}$, and so by induction $Z(\vec{x}) \wedge Z(\vec{y}) = \texttt{true}$, as required. □

5.3 A polynomial-time algorithm

Lemmas 24 and 26 provide us with a criterion for deciding the MULSETPC problem that can be computed in polynomial-time in the size of the trees, for any fixed number of trees.

Theorem 29 *Let $T_1, T_2, \ldots, T_k$ be a collection of k binary MUL-trees such that $\bigcup_i \mathcal{X}(T_i) = \bigcup_i D(T_i)$. Then* MULSETPC *for this instance can be solved in $O((nk + 4^k) \cdot \prod_{i=1}^k (|T_i| + 1)) = O^*(8^k \cdot \prod_{i=1}^k |T_i|)$ time.*

Proof Based on Lemmas 24 and 26, it is sufficient to compute $Z(r_1, \ldots, r_k)$, where r_i is the root of tree T_i, since $W(r_1, \ldots, r_k) = Z(r_1, \ldots, r_k)$ and $T_i^{r_i} = T_i$ for all $i = 1, \ldots, k$. We can compute $Z(r_1, \ldots, r_k)$ via dynamic programming with memoization as outlined in Algorithm 1. To simplify the presentation, the operations related to memoization have been omitted from the pseudocode. More precisely, we assume that whenever the algorithm has computed some $Z(a_1, \ldots, a_k)$-value and is about to return, it also stores it in a table, and that whenever the algorithm needs some Z-value on line 6, it can check if that value has previously been computed and if so, retrieve it, in $O(1)$ time.

Algorithm 1 Dynamic programming algorithm for $Z(a_1, \ldots, a_k)$

```
1: let S = S(a_1, ..., a_k)
2: if S ∩ L(T_i) ⊈ L(T_i^{a_i}) for some i = 1, ..., k then return false
3: else if |S| ≤ 1 then return true
4: else
5:     for (x_1, ..., x_k) ∈ Π(a_1, ..., a_k) do
6:         if Z(x_1, ..., x_k) ∧ Z(c^{a_1}(x_1), ..., c^{a_k}(x_k)) then return true
7:     return false
```

For the time complexity, we can use memoization to store the values of Z in a table with at most $O(\prod_{i=1}^{k} |V(T_i) \cup \{\epsilon\}|) = O(\prod_{i=1}^{k} (|T_i| + 1))$ entries. We preprocess each tree T_i to compute, for every node a_i, a bit vector of length n which indicates the labels appearing in $T_i^{a_i}$. The preprocessing takes $O(n \sum_{i=1}^{k} |T_i|)$ time using bottom-up traversals, and after preprocessing, any $S(a_1, \ldots, a_k)$-set can be obtained in $O(nk)$ time. Furthermore, we require at most $O(nk + 4^k)$ time to compute the value of each entry $Z(a_1, \ldots, a_k)$, since $|\Pi(a_1, \ldots, a_k)| < |P(a_i)| \times \cdots \times |P(a_k)| \leq 4^k$. Hence, the running time is $O((nk + 4^k) \cdot \prod_{i=1}^{k} (|T_i| + 1)) = O((nk + 4^k) \cdot \prod_{i=1}^{k} 2|T_i|) = O^*(8^k \cdot \prod_{i=1}^{k} |T_i|)$, as required. □

Given an instance of unrestricted MULSETPC, it is easy to create an equivalent instance where every label is unique in at least one tree. To do this, for every label that is not already unique in one tree, add a new singleton tree consisting only of that label. This will give an instance which can be solved by Algorithm 1 and which will have the same output as the original instance. However, since the running time of Algorithm 1 depends exponentially on the number of input trees, adding these extra trees could potentially give a running time exponential in n.

6 Conclusions

The above results resolve an open problem posed in [14,16] as to whether the MULSETPC problem remains NP-complete when the number of MUL-trees is constant. According to Theorems 9 and 10, two MUL-trees are sufficient for NP-completeness, even with each label appearing at most twice in any tree and either the height or the degree constant. Theorems 11 and 12 extend this result and show that the more general MULSETPComp problem also remains NP-complete even when the number of MUL-trees is constant. Theorem 9 is tight in the sense that, if we restrict our attention to MUL-trees in which each label appears uniquely in at least one tree, we obtain a polynomial-upper bound for a fixed number of trees (Theorem 29). However, Theorem 12 suggests the algorithm presented in Theorem 29 for MULSETPC cannot be directly generalised to solve equivalent MULSETPComp instances in polynomial time, unless $P = NP$.

The above results also suggest two new open problems. Firstly, is it possible to improve Theorem 11 to show that MULSETPComp is still NP-complete when restricted to MUL-trees with multiplicity 2? Secondly, what can be said about the complexity of MULSETPC and MULSETPComp for instances in which the multiplicity is not restricted but the number of leaf labels that may appear more than once is restricted? That is, for each MUL-tree, k leaf labels may appear an unbounded number of times in the tree, whereas all other labels appear at most once. For which values of k are these subproblems NP-complete? This question is interesting because of its connection to the instances investigated in Sect. 5.

Author Contributions All authors contributed equally to the manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D.: Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. SIAM J. Comput. **10**(3), 405–421 (1981)
2. Amir, A., Keselman, D.: Maximum Agreement Subtree in a Set of Evolutionary Trees: Metrics and Efficient Algorithms. SIAM J. Comput. **26**(6), 1656–1669 (1997)
3. Bansal, M.S.: Linear-time algorithms for some phylogenetic tree completion problems under Robinson-Foulds distance. In RECOMB International conference on Comparative Genomics, pages 209–226. Springer, 2018
4. Bansal, M.S., Burleigh, J.G., Eulenstein, O., Fernández-Baca, D.: Robinson-Foulds supertrees. Algorithms for Molecular Biology **5**(1), 1–12 (2010)
5. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Ann. Comb. **8**(4), 409–423 (2005)
6. Bryant, D.: A classification of consensus methods for phylogenetics. In: M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, B. Mirkin, and F. S. Roberts, (eds), Bioconsensus, Vol. 61 DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 163–184. American Mathematical Society (2003)
7. Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., Thorup, M.: An $O(n \log n)$ Algorithm for the maximum agreement subtree problem for binary trees. SIAM J. Comput. **30**(5), 1385–1404 (2000)
8. Cui, Y., Jansson, J., Sung, W.-K.: Polynomial-time algorithms for building a consensus MUL-Tree. J. Comput. Biol. **19**(9), 1073–1088 (2012)
9. Deng, Y., Fernández-Baca, D.: Fast compatibility testing for rooted phylogenetic trees. Algorithmica **80**(8), 2453–2477 (2018)
10. Ding, Z., Filkov, V., Gusfield, D.: A linear-time algorithm for the perfect phylogeny haplotyping (PPH) problem. J. Comput. Biol. **13**(2), 522–553 (2006)
11. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates Inc, Sunderland, Massachusetts (2004)
12. Finden, C.R., Gordon, A.D.: Obtaining common pruned trees. J. Classif. **2**(1), 255–276 (1985)
13. Ganapathy, G., Goodson, B., Jansen, R., Le, H., Ramachandran, V., Warnow, T.: Pattern identification in biogeography. IEEE/ACM Trans. Comput. Biol. Bioinf. **3**(4), 334–346 (2006)
14. Huber, K.T., Moulton, 871 V.: Phylogenetic networks from multi-labelled trees. J. Math. Biol. 52(5), 613–632 (2006)
15. Huber, K.T., Moulton, V., Steel, M., Wu, T.: Folding and unfolding phylogenetic trees and networks. J. Math. Biol. 73(6), 1761–1780 (2016)
16. Jansson, J., Shen, C., Sung, W.-K.: Improved algorithms for constructing consensus trees. J. ACM 63(3), (2016). (Article 28)
17. Nelson, G.J., Platnick, N.I.: Systematics and Biogeography: Cladistics and Vicariance, Columbia University Press (1981)
18. Lott, M., Spillner, A., Huber, K.T., Petri, A., Oxelman, B., Moulton, V.: Inferring polyploid phylogenies from multiply-labeled gene trees. BMC Evol. Biol. **9**(1), 1–11 (2009)
19. Gascon, M., Dondi, R., El-Mabrouk, N.: Complexity and Algorithms for MUL-Tree Pruning. In: Flocchini P., Moura L., (eds.) Combinatorial Algorithms: 32nd International Workshop on Combinatorial Algorithms (IWOCA 2021), pp. 324–339, Cham, (2021). Springer International Publishing
20. Minaka, N.: Cladograms and reticulated graphs: A proposal for graphic representation of cladistic structures. Bull. Biogeogr. Soc. Jpn **45**(1), 1–10 (1990)
21. Page, R.D.M.: Parasites, phylogeny and cospeciation. Int. J. Parasitol. 23(4), 499–506 (1993)
22. Page, R.D.M.: Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. Syst. Biol. 43(1), 58–77 (1994)
23. Pe'er, I., Pupko, T., Shamir, R., Sharan, R.: Incomplete directed perfect phylogeny. SIAM J. Comput. 33(3), 590–607 (2004)
24. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Math. Biosci. 53(1–2), 131–147 (1981)

25. Scornavacca, C., Berry, V., Ranwez, V.: Building species trees from larger parts of phylogenomic databases. Inf. Comput. **209**(3), 590–605 (2011)
26. Steel, M.: The complexity of reconstructing trees from qualitative characters and subtrees. J. Classification **9**(1), 91–116 (1992)
27. Steel, M., Warnow, T.: Kaikoura tree theorems: Computing the maximum agreement subtree. Inf. Process. Lett. **48**, 77–82 (1993)
28. van Iersel, L., Kelk, S., Lekić, N., Scornavacca, C.: A practical approximation algorithm for solving massive instances of hybridization number for binary and nonbinary trees. BMC Bioinformatics **15**(1), 1–12 (2014)
29. Whidden, C., Zeh, N., Beiko, R.G.: Supertrees based on the subtree prune-and-regraft distance. Syst. Biol. **63**(4), 566–581 (2014)
30. Wu, Y.: A practical method for exact computation of subtree prune and regraft distance. Bioinformatics **25**(2), 190–196 (2009)