Listing Conflicting Triples in Optimal Time

Mathias Weller

Listing Conflicting Triples in Optimal Time

Mathias Weller

CNRS, LIGM, Université Paris Est, Marne-la-Vallée, France

Abstract

Different sources of information might tell different stories about the evolutionary history of a given set of species. This leads to (rooted) phylogenetic trees that "disagree" on triples of species, which we call "conflict triples". An important subtask of computing consensus trees which is interesting in its own regard is the enumeration of all conflicts exhibited by a pair of phylogenetic trees (on the same set of n taxa). As it is possible that a significant part of the $\binom{n}{3}$ triples are in conflict, the trivial $\theta(n^3)$-time algorithm that checks for each triple whether it constitutes a conflict, was considered optimal. It turns out, however, that we can do way better in the case that there are only few conflicts. In particular, we show that we can enumerate all d conflict triples between a pair of phylogenetic trees in $O(n+d)$ time. Since any deterministic algorithm has to spend $\Theta(n)$ time reading the input and $\Theta(d)$ time writing the output, no deterministic algorithm can solve this task faster than we do (up to constant factors).

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

Keywords and phrases parameterized algorithms, phylogenetic trees, triplet enumeration, enumeration algorithms, polynomial time

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

In bioinformatics – more precisely, phylogenetics – evolutionary trees ("phylogenetic trees") are one of the fundamental types of data representation and, thus, among the most important objects being algorithmically analyzed and manipulated. A phylogenetic tree visualizes the evolutionary history of a set of taxa (e.g. a family of genes, a collection of species, etc.). However, different sources of information might imply different evolutionary histories of the same taxa. Such contradictions manifest themselves as "conflict triples" (sometimes also "conflict triplets"), that is, three taxa, say a, b, and c such that one phylogenetic tree P implies that a common ancestor of a and b split off the common lineage of a, b and c before splitting into a and b while another tree Q implies that a common ancestor of b and c split off the common lineage before splitting into b and c. More formally, $\mathrm{LCA}_P(ab) \neq \mathrm{LCA}_P(abc)$ and $\mathrm{LCA}_Q(bc) \neq \mathrm{LCA}_Q(ab) = \mathrm{LCA}_Q(abc)$. See Figure 1 for an example.

Conflict triples are essential ingredients to algorithms building so-called "supertrees", that is, phylogenetic trees that merge evolutionary histories into one that is "most consistent" [3, 9]. Conflict triples can also be used to reconcile gene trees into a single phylogeny by building a so-called "triplet-based median supertree" [11]. The problem of *counting* conflict triples has been used to measure the distance between phylogenetic trees. Brodal et al. [2] show how to compute this number in $O(n \log n)$ time. A recent study of the problem of finding a consensus tree given a set of disagreeing phylogenetic trees [4] makes heavy use of the list of all conflict triples between any two of the input trees, but does not detail how to enumerating them efficiently. Here, we address this problem, showing how to enumerate all d conflict triples of a pair (P, Q) of phylogenetic trees on n taxa in $O(n+d)$ time. Since

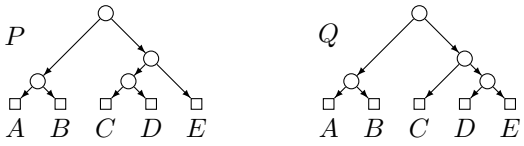

Figure 1 Two phylogenetic trees P and Q with conflict CDE (boxes = leaves, circles = inner vertices). In particular, $CD|_P E$ and $DE|_Q C$.

all algorithms solving this problem need to read the input (size $\Theta(n)$) and write the output (size $\Theta(d)$), this is asymptotically "best possible".

While *counting* the number of conflicts has received some attention in the past [2], not much work has been done on *enumerating* them. Such development might have been discouraged by the fact that a significant portion of the $\binom{n}{3}$ triples of taxa might be in conflict, in which case the trivial algorithm that tests each triple of taxa for being a conflict would be optimal. This work emerged from the question whether we can do better if only few triples are actually in conflict. In this sense, our work is in the context of "FPT in P", a research direction that brings ideas of parameterized complexity theory to the world of polynomial-time solvable problems. Indeed, parameterized complexity theory aims at providing algorithms for *hard* problems that run fast *in practice*, assuming that some measure of difficulty (the "parameter") is small in the instances that a particular application produces. Previously, "hard" most often meant "NP-hard", but there is no reason not to widen ones view to include polynomial-time solvable problems with *impractical* running time. While preliminary works in this direction focussed on decision problems [6, 8, 10], we consider an enumeration-type problem here.

Indeed, the concept of measuring the complexity in the size of the input *and* the output is fairly well known as *output sensitivity* in the context of enumeration algorithms. Running in $O(n+d)$ time where n is the size of the input and d is the size of the output, our algorithm can be called *total linear*.

2 Preliminaries

A *(phylogenetic) tree* is a rooted, binary[1] outbranching whose leaves are bijectively labeled by a set X (of taxa) and we refer to its root by $r(T)$. Since the labeling is bijective, we use leaves and labels interchangeably. If some vertex v of T is a strict ancestor of a vertex u in T, we write $u <_T v$ and we abbreviate $\forall_{v \in Z} v <_T u$ to $Z <_T u$. We also abbreviate sets of leaves (or labels) by the concatenation of their names, that is, abc refers to $\{a, b, c\}$. The *least common ancestor* of two leaves (or labels) a and b in T is the minimum among all u with $ab <_T u$ and we write $\mathrm{LCA}_T(ab) = u$. In this work a *triple* abc in T is a set of three labels $abc \subseteq X$. We say that abc *touches* $\mathrm{LCA}_T(abc)$ and omit the mention of T if it is clear from context. We say a triple abc is *ab-biased* in T if $\mathrm{LCA}_T(ab) \neq \mathrm{LCA}_T(abc)$ and we write $ab|_T c$ to indicate this fact. A triple abc is called a *conflict* of a pair (P, Q) of trees if, for some $xy \subseteq abc$, we have that abc is xy-biased in exactly one of P and Q (see Figure 1). Recall that abc and cab refers to the same conflict, so when claiming that abc is not listed twice, this also means that no two permutations of abc are listed.

For two vertices $u \in V(P)$ and $v \in V(Q)$, we define $u \sqcap v := \mathcal{L}(P_u) \cap \mathcal{L}(Q_v)$ and $u \wr v := \mathcal{L}(P_u) \setminus \mathcal{L}(Q_v)$. Note that $\sqcap$ is symmetrical while $\wr$ is not.

[1] While we only consider binary phylogenetic trees in this work, I conjecture that it easily generalizes.

▶ Observation 1. Let P and Q be phylogenetic trees on the same leaf-set. Let r_p and r_q be the roots of P and Q, respectively, and let u_p, v_p and u_q, v_q be their respective children. Then, $u_p \wr u_q = u_p \sqcap v_q = v_q \sqcap u_p = v_q \wr v_p$.

In the following, we call a tree T *LCA-enabled* if the LCA of any two vertices in T can be found in constant time. Note that we can LCA-enable any tree in linear time [1, 7].

In the algorithm, we will want to compute the subtree T' of a tree T that is induced by a set Z of leaves. If Z is ordered by an in-order or post-order traversal of T, then this can be done in $O(|Z|)$ time [5, Section 8]. The idea is that the inner vertices of T' are exactly the LCAs of consecutive (wrt. the order) leaves in Z and the arcs between them can be computed by looking at the nearest, lower vertex on the left and right of each inner vertex of T' according to the order.

▶ Observation 2 ([5, Section 8]). Let T be an LCA-enabled tree and let $Z \subseteq \mathcal{L}(T)$ be in post-order. Then, $T|_Z$ can be computed in $O(|Z|)$ time.

Furthermore, for leaf-labelled trees P and Q and vertices u and v of P and Q, respectively, we will want to detect whether $\mathcal{L}(P_u) = \mathcal{L}(Q_v)$ in constant time. To this end, we construct a mapping m that maps each vertex x of P to the unique vertex y of Q that is lowest among all vertices of Q satisfying $\mathcal{L}(P_x) \subseteq \mathcal{L}(Q_y)$. Note that, $m(x) = \mathrm{LCA}_Q(m(x'), m(x''))$ where x' and x'' are the children of x in P and, thus, m can be computed in $O(|P| + |Q|)$ time if Q is LCA-enabled. Finally, we only need to know the number of leaves reachable from each vertex of P and Q, which can easily be computed in $O(|P| + |Q|)$ time.

▶ Observation 3. Let P and Q be phylogenetic trees on the same leaf-set and let Q be LCA-enabled. Then, there is a linear-time preprocessing that allows answering if $\mathcal{L}(P_u) = \mathcal{L}(Q_v)$ in constant time for each u and v.

3 The Algorithm

Given two phylogenetic trees P and Q on the label-set X, our algorithm will first list all conflict triples abc that touch $r(P)$ or $r(Q)$ and then recurse into specific induced subtrees of P and Q such that, the conflicts in these subtrees are exactly the conflicts between P and Q that do not touch $r(P)$ and $r(Q)$. The observation that being a conflict triple is invariant under deletion of unrelated leaves implies the correctness of this approach.

▶ Observation 1. Let $Y \subseteq X$, and let $abc \subseteq Y$. Then, abc is a conflict triple of (P, Q) if and only if abc is a conflict triple of $(P|_Y, Q|_Y)$.

▶ Observation 2. Let abc be a conflict triple of (P, Q) that touches neither $r(P)$ nor $r(Q)$. Let u_p and v_p be the children of $r(P)$ and let u_q and v_q be the children of $r(Q)$. Then, abc is completely contained in $u_p \sqcap u_q$, $u_p \sqcap v_q$, $v_p \sqcap u_q$, or $v_p \sqcap v_q$.

Note that the four sets mentioned in Observation 2 are disjoint, and so, no conflict can be contained in any two of them. Then, our algorithm can be described as the following recursion (see Algorithm 1 for a detailed description):

Base Case: If $r(P)$ and $r(Q)$ are leaves, then return without listing anything.

Recursion: First, choose an arbitrary pairing $\{(u_p, u_q), (v_p, v_q)\}$ of the children of $r(P)$ and $r(Q)$. Second, list all conflict triples abc touching $r(P)$ or $r(Q)$. Third, recursively list all conflict triples of

1. $(P|_{u_p \sqcap u_q}, Q|_{u_p \sqcap u_q})$,
2. $(P|_{v_p \sqcap v_q}, Q|_{v_p \sqcap v_q})$,
3. $(P|_{u_p \sqcap v_q}, Q|_{u_p \sqcap v_q})$ and
4. $(P|_{v_p \sqcap u_q}, Q|_{v_p \sqcap u_q})$.

Procedure ListCommonRootConflicts
Input: Trees P & Q on X, a child x_p of $r(P)$, a child x_q of $r(Q)$
Output: Conflict triples abc with $ab \leq x_p$ touching $r(P)$ and $r(Q)$
1 **foreach** $a \in x_p \sqcap x_q$ *and* $b \in x_p \wr x_q$ *and* $c \in X \setminus \mathcal{L}(x_p)$ **do list** abc;

Procedure ListUncommonRootConflicts
Input: Trees P & Q on X, a child x_p of $r(P)$, a child x_q of $r(Q)$
Output: Conflict triples $abc \leq x_p$ touching $r(Q)$ (but not $r(P)$)
1 **foreach** $a, b \in x_p \sqcap x_q$ *and* $c \in x_p \wr x_q$ *with* $ab \not|_P c$ **do list** abc;
2 **foreach** $a, b \in x_p \wr x_q$ *and* $c \in x_p \sqcap x_q$ *with* $ab \not|_P c$ **do list** abc;

Procedure ListAllConflicts
Input: Trees P & Q
Output: Conflict triples of (P, Q)
1 **if** $|\mathcal{L}(P)| > 1$ **then**
2 $(u_p, u_q), (v_p, v_q) \leftarrow$ arbitrary pairing of children of $r(P)$ & $r(Q)$;
3 **foreach** $(x_p, x_q) \in \{(u_p, u_q), (v_p, v_q)\}$ **do**
4 compute $x_p \sqcap x_q$, $x_p \wr x_q$ and $x_q \wr x_p$;
5 `ListCommonRootConflicts`(P, Q, x_p, x_q);
6 `ListUncommonRootConflicts`(P, Q, x_p, x_q);
7 `ListUncommonRootConflicts`(Q, P, x_q, x_p);
8 `ListAllConflicts`$(P|_{x_p \sqcap x_q}, Q|_{x_p \sqcap x_q})$;
9 `ListAllConflicts`$(P|_{u_p \wr u_q}, Q|_{v_q \wr v_p})$;
10 `ListAllConflicts`$(P|_{v_p \wr v_q}, Q|_{u_q \wr u_p})$;

1 First shot at triplet enumeration. Note that, although theoretically unnecessary, we provide x_q to the calls to `ListCommonRootConflicts` and `ListUncommonRootConflicts`, since this lets us use the pre-computed sets $x_p \sqcap x_q$ and $x_p \wr x_q$ and $x_q \wr x_p$.

We defer showing correctness in favor of introducing some modifications that allow achieving our running-time goal. In order to see why this is necessary, let us analyze `ListAllConflicts`. This requires a closer look at how many triples are listed in each recursive step. `ListCommonRootConflicts` unconditionally lists $|x_p \sqcap x_q| \cdot |x_p \wr x_q| \cdot |X \setminus \mathcal{L}(x_p)|$ conflicts for each pair (x_p, x_q) of the chosen pairing. However, `ListUncommonRootConflicts` has to perform numerous checks of the type "$ab|c$?". Since it is possible that none of these triples is a conflict, we cannot bound these operations in the number of listed conflicts. Instead, we use `ListSubtreeConflicts` to list all the triples abc with $a, b \in x_p \sqcap x_q$ and $c \in x_p \wr x_q$ (or vice versa), and $ab \not|_P c$ in constant time per listed triple (see Figure 2 for an illustration). The idea is (i) to focus on the subtree P' of P that is rooted at $\mathrm{LCA}_P(x_p \sqcap x_q)$, (ii) to pick any leaf $c \in x_p \wr x_q$ and, (iii) for each y on the unique path from c to $r(P')$, listing all triples abc for which a and c are "below y" and b is not, thereby ensuring $\mathrm{LCA}_T(ac) \neq \mathrm{LCA}_T(abc)$. We will thus replace the first for-loop of `ListUncommonRootConflicts` by a call to `ListSubtreeConflicts`$(P, x_p \sqcap x_q)$ and the second for-loop with a call to `ListSubtreeConflicts`$(P, x_p \wr x_q)$.

▶ **Lemma 1.** `ListSubtreeConflicts` *is correct, that is, it outputs a triple abc if and only if $a, b \in Z$, $c \notin Z$, and $ab \not|_T c$. Further, the procedure takes $O(d)$ time (where d is the total number of listed triples) and no triple is listed twice.*

Procedure ListSubtreeConflicts

Input: Tree T, leaf subset $Z \subseteq \mathcal{L}(T)$ in post-order
Output: Triples abc with $a, b \in Z$, and $c \in \mathcal{L}(T) \setminus Z$, and $ab \not\parallel_T c$

```
1 if Z ≠ ∅ then
2     foreach c ∈ L(T) \ Z do
3         T' ← T|_{Z∪{c}};
4         y ← parent of c;
5         while y ≠ r(T') do
6             y' ← sibling of y in T';
7             foreach a ∈ L(T'_y) \ {c} and b ∈ L(T'_{y'}) do list abc;
8             y ← parent of y in T';
```

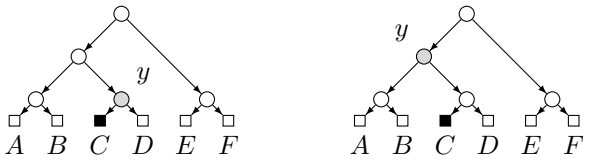

Figure 2 An example illustrating the tree T' in two steps of `ListSubtreeConflicts` (gray = vertex y, black = leaf c with label C). Left: first step (y is the parent of c), listing DAC and DBC. Right: second step, listing all abC, with $a \in \{A, B, D\}$ and $b \in \{E, F\}$.

Proof. We first show the first equivalence.

"⇒": Let abc be a listed triple. Then, there is some y with $c < y < r(T')$ with sibling y' such that $a \in \mathcal{L}(T'_y) \setminus \{c\}$ and $b \in \mathcal{L}(T'_{y'})$ (by symmetry among ab). But then, $a, b \in Z$, and $c \notin Z$ and $ac <_{T'} y$ and $b \leq_{T'} y'$, implying $ac|_{T'}b$ and, thus, $ac|_T b$.

"⇐": Let abc be a triple with $a, b \in Z$, $c \notin Z$ and $ab \not\parallel_T c$. Then, $|Z \neq \varnothing$, and $c \in \mathcal{L}(T) \setminus Z$. Since $ab \not\parallel_T c$, we have $\mathrm{LCA}_T(ab) = \mathrm{LCA}_T(abc)$ and, by symmetry among ab, we suppose $\mathrm{LCA}_T(ac) < \mathrm{LCA}_T(abc)$. Let y and y' be the children of $\mathrm{LCA}_{T'}(abc)$ with $a, c <_{T'} y$ and note that y will be reached by the while-loop. Clearly, $a \in \mathcal{L}(T'_y)$, and $b \in \mathcal{L}(T'_{y'})$, and, thus, abc is listed.

Second, suppose that any triple abc is listed twice. As y and y' are siblings in each iteration of the while-loop, abc is listed for two different values of y. However, there is a single vertex (namely $\mathrm{LCA}(ab)$) for which neither $ab \subseteq \mathcal{L}(T'_y)$ nor $ab \subseteq \mathcal{L}(T'_{y'})$. Thus, there is a single iteration for which abc can be output.

Finally, we show the claimed running time. We start by showing that, each time the while-loop is run, it outputs at least $|Z|-1$ triples. To this end, consider y' and its sibling y in any last iteration of the while-loop (that is, the parent of y and y' is $r(T')$). Then, the number of triples that are listed is $|\mathcal{L}(T'_y)-1| \cdot |\mathcal{L}(T'_{y'})| \geq |\mathcal{L}(T'_y)|-1+|\mathcal{L}(T'_{y'}|-1 = |\mathcal{L}(T')|-1 = |Z|-1$. Since, by Observation 2, T' can be computed in $O(|Z|)$ time (line 3), we conclude that `ListSubtreeConflicts` runs in $O(d)$ time. ◀

With Lemma 1, we can finally list all d_r conflict triples abc with $\mathrm{LCA}_P(abc) = r(P)$ or $\mathrm{LCA}_Q(abc) = r(Q)$ in $O(d_r)$ time. Thus, `ListAllConflicts` completes the following tasks in the mentioned times.
(Task a) list all conflict triples touching $r(P)$ or $r(Q)$: $O(d_r)$ time;
(Task b) compute common and uncommon leaves: $O(|X|)$ time;

(Task c) compute the subtrees induced by these leaf-sets: $O(|X|)$ time;
(Task d) preprocess these subtrees for the recursive calls: $O(|X|)$ time;
(Task e) make recursive calls

The algorithm in its current form has a worst-case running time of $O(|X|^2)$. In the following, we show how to avoid the costly computations of (b), (c), and (d) if they are unnecessary and bound their running-time in $O(d_r)$ if they cannot be avoided. To this end, note that, when called with u_p and u_q, `ListCommonRootConflicts` outputs

$$|u_p \sqcap u_q| \cdot |u_p \wr u_q| \cdot (|v_p \sqcap v_q| + |v_p \wr v_q|) \leq d_r$$

unique conflicts. Thus, if $u_p \sqcap u_q \neq \varnothing$ and $u_p \wr u_q \neq \varnothing$, then

$$\begin{aligned} |X| &= (|u_p \sqcap u_q| + |u_p \wr u_q|) + (|v_p \sqcap v_q| + |v_p \wr v_q|) \\ &\leq |u_p \sqcap u_q| \cdot |u_p \wr u_q| \cdot (|v_p \sqcap v_q| + |v_p \wr v_q|) + 2 \leq d_r + 2 \end{aligned}$$

and we can thus bound the time spent for (b), (c), and (d) in $O(d_r)$. By symmetry, the same holds if $v_p \sqcap v_q \neq \varnothing$ and $v_p \wr v_q \neq \varnothing$. It remains to explore the cases that one of $u_p \sqcap u_q$ and $u_p \wr u_q$ and one of $v_p \sqcap v_q$ and $v_p \wr v_q$ is empty.

First, $u_p \wr u_q = v_p \sqcap v_q = \varnothing$. Then all leaves of P_{u_p} are leaves of Q_{u_q} and all leaves of P_{v_p} are not leaves of Q_{v_q}. Thus, Q_{v_q} does not have any leaves, contradicting the fact that P and Q are binary trees. Symmetrically, $u_p \sqcap u_q = v_p \wr v_q = \varnothing$ cannot happen.

Second, $u_p \wr u_q = v_p \wr v_q = \varnothing$. Then, $\mathcal{L}(u_p) = \mathcal{L}(u_q)$ and $\mathcal{L}(v_p) = \mathcal{L}(v_q)$. This situation can be detected in constant time, given a linear-time preprocessing of P and Q that links a node x_p of P to a node x_q of Q if and only if P_{x_p} and Q_{x_q} have the same leaf-set (see Observation 3). In this case, there are no root-conflicts and none of the costly steps (b)–(d) are necessary.

Third, $u_p \sqcap u_q = v_p \sqcap v_q = \varnothing$. Then, changing the root-child pairing to (u_p, v_q) and (v_p, u_q) gives the previous case. The same preprocessing allows us to detect and deal with this case.

The final version of the algorithm is presented as Algorithm 2 and we can prove its running time and correctness.

▶ **Lemma 2.** *Algorithm 2 outputs a triple if and only if it is a conflict. Moreover, no conflict is listed twice and Algorithm 2 runs in $O(|X|+d)$ time, where X is the label set of the input trees and d is the total number of conflicts listed.*

Proof. Let line 2 of `ListAllConflicts` produce the pairs (u_p, u_q) and (v_p, v_q).

"⇒": Let abc be a triple that is listed by Algorithm 2. If `ListCommonRootConflicts` lists abc then, without loss of generality, $a \in u_p \sqcap u_q$, and $b \in u_p \wr u_q$, and $c \in X \setminus \mathcal{L}(u_p)$. Thus, $a \leq u_p, u_q$, and $b \leq u_p, v_q$, and $c \leq v_p$. Now, if $c \leq v_q$, then $ab|_P c$ and $a|_Q bc$, otherwise, $ab|_P c$ and $ac|_Q b$. In both cases, abc is a conflict. Otherwise, abc is listed by `ListSubtreeConflicts` and, without loss of generality, let the first argument be P (lines 12 and 13). Then, by construction of `ListSubtreeConflicts`, there is some $Z \in \{x_p \sqcap x_q, x_p \wr x_q\}$ and some y such that $a, c <_P y$, and $a, b \in Z$, and $c \notin Z$, and $y < \mathrm{LCA}_P(ab)$. Thus $ac|_P b$. Now, if $Z = x_p \sqcap x_q$ then, as $c < y < x_p$ and $c \notin Z$, we have $c \not\leq x_q$, but $a, b < x_q$, implying $ab|_Q c$. If $Z = x_p \wr x_q$ then, as $c < y < x_p$ and $c \notin Z$, we have $c \leq x_q$, but $a, b \not\leq x_q$, implying $ab|_Q c$. In both cases, abc is a conflict.

"⇐": Let abc be a conflict between P and Q and, by symmetry among abc, let $ab|_P c$ and $ac|_Q b$. Further, by symmetry among u_p and v_p, let $ab < u_p$. First, suppose that $\mathrm{LCA}_P(abc) = r(P)$, that is, $c \leq v_p$. If $abc < u_q$ (or $abc < v_q$), then there is $Z := u_p \sqcap u_q$

Procedure ListAllConflicts'

Input: Trees P & Q, preprocessed to answer leaf-set equivalence in $O(1)$
Output: Conflict triples of (P, Q)

```
1  (u_p, u_q), (v_p, v_q) ← arbitrary pairing of children of r(P) & r(Q);
2  if L(u_p) = L(v_q) then swap u_q and v_q;
3  if L(u_p) = L(u_q) then
4      ListAllConflicts'(P_{u_p}, Q_{u_q});
5      ListAllConflicts'(P_{v_p}, Q_{v_q});
6  else
7      foreach (x_p, x_q) ∈ {(u_p, u_q), (v_p, v_q)} do
8          compute & post-order the sets x_p ⊓ x_q, x_p ≀ x_q and x_q ≀ x_p;
9          compute P|_{x_p⊓x_q}, P|_{u_p≀u_q}, Q|_{x_q⊓x_p}, and Q|_{x_q≀x_p};
10         compute the leaf-set equivalence relation for corresponding tree-pairs;
11         ListCommonRootConflicts(P, Q, x_p, x_q);
12         ListSubtreeConflicts(P, x_p ⊓ x_q);
13         ListSubtreeConflicts(P, x_p ≀ x_q);
14         ListSubtreeConflicts(Q, x_q ⊓ x_p);
15         ListSubtreeConflicts(Q, x_q ≀ x_p);
16         ListAllConflicts'(P|_{x_p⊓x_q}, Q|_{x_q⊓x_p});
17     ListAllConflicts'(P|_{u_p≀u_q}, Q|_{v_q≀v_p});
18     ListAllConflicts'(P|_{v_p≀v_q}, Q|_{u_q≀u_p});
```

2 Refined algorithm to enumerate all conflict triples. Note that we do not have to update leaf-set equivalence relations for the recursions in lines 4 and 5 since the relation computed in the parent remains valid.

(or $Z := u_p \wr u_q$) with $a, b \in Z$ and $c \notin Z$ and $ab \not| _Q c$ and, by Lemma Lemma 1, abc is listed by `ListSubtreeConflicts` in line 14 (or line 15). Otherwise, $\mathrm{LCA}_Q(abc) = r(Q)$, that is, $ac < u_q$ and $b \leq v_q$ or vice versa (since $ac|_Q b$). But then, $ac < u_q$ (or $ac < v_p$) and $b \leq v_q$ (or $b \leq u_p$), implying $a \in u_p \sqcap u_q$, and $b \in u_p \wr u_q$ (or $b \in u_p \sqcap u_q$, and $a \in u_p \wr u_q$), and $c \not< u_p$ and, thus, abc is listed by `ListCommonRootConflicts` in line 11. Second, suppose that $\mathrm{LCA}_P(abc) < r(P)$, that is, $c \leq u_p$. If $\mathrm{LCA}_Q(abc) = r(Q)$, then $ac < u_q$ and $b < v_q$ or vice versa. But then, there is $Z := u_p \sqcap u_q$ (or $Z := u_p \wr u_q$) with $a, c \in Z$, and $b \notin Z$ and $ac \not| _P b$ and, by Lemma 1, acb is listed by `ListSubtreeConflicts` in line 12 (or line 13). Otherwise, $\mathrm{LCA}_Q(abc) < r(Q)$. If $abc < u_q$ then, by induction on the recursion depth, abc is listed by the recursive call on line 16 (or line 4 if $\mathcal{L}(u_p) = \mathcal{L}(u_q)$). Otherwise, $abc < v_q$ and, by induction on the recursion depth, abc is listed by the recursive call on line 17 (or line 4 if $\mathcal{L}(u_p) = \mathcal{L}(v_q)$, as u_q and v_q would have been swapped in line 2 in this case).

To show that no conflict abc is output twice, assume the contrary. Again, symmetry lets us suppose $ab|_P c$, and $ac|_Q b$, and $ab < u_p$. Note that the two occurrences of abc cannot be output by

- different recursive calls, since all tree-pairs in recursive calls have pairwise disjoint sets of leaf-labels,
- the same call to `ListCommonRootConflicts` since $x_p \sqcap x_q$, and $x_p \wr x_q$ and $X \setminus \mathcal{L}(x_p)$ are pairwise disjoint, or
- the same call to `ListSubtreeConflicts` by Lemma 1.

Thus, abc is listed by different calls in the same node of the recursion tree. If $\mathrm{LCA}_P(abc) = r(P)$ and $\mathrm{LCA}_Q(abc) = r(Q)$, then abc is listed by both calls to `ListCommonRootConflicts`, implying that abc intersects $u_p \sqcap u_q$ and $u_p \wr u_q$ as well as $v_p \sqcap v_q$ and $v_p \wr v_q$. However, as

these sets are disjoint, this cannot happen. If $\mathrm{LCA}_P(abc) = r(P)$ and $\mathrm{LCA}_Q(abc) \neq r(Q)$, then $abc <_Q u_q$ or $abc <_Q v_q$ and $c \leq_P v_p$. If $abc <_Q u_q$, then $ab \subseteq u_p \sqcap u_q$ and abc can be listed only in the call to `ListSubtreeConflicts` in line 14 for $(x_p, x_q) = (u_p, u_q)$. If $abc <_Q v_q$, then $ab \subseteq v_q \wr v_p$ and abc can be listed only in the call to `ListSubtreeConflicts` in line 15 for $(x_p, x_q) = (v_p, v_q)$. The case that $\mathrm{LCA}_P(abc) \neq r(P)$ and $\mathrm{LCA}_Q(abc) = r(Q)$ is completely analogous. Since the case that $\mathrm{LCA}_P(abc) \neq r(P)$ and $\mathrm{LCA}_Q(abc) \neq r(Q)$ is treated in a different recursive step, this case distinction is exhaustive and abc is indeed not listed twice.

To show the running time, let $\mathcal{T}$ denote the recursion tree for input (P, Q) and, for each node v of $\mathcal{T}$, let δ_v and γ_v denote the time spent in lines 1–3 and in lines 8–15, respectively. Then, the algorithm finishes in $\sum_{v \in V(\mathcal{T})} (\delta_v + \gamma_v + O(1))$ time. First, using the leaf-set equivalence relation computed in line 10 in the parent of v (or pre-computed if v is the root), we execute lines 1–3 in constant time, that is, $\delta_v \in O(1)$. Second, by the consideration above, tasks (a)–(d) can be completed in $O(d_r)$ time, where d_r is the number of triples output by `ListCommonRootConflicts` and `ListSubtreeConflicts`, that is, in lines 11–15. Then, we can bound the total running time by

$$\sum_{v \in V(\mathcal{T})} \delta_v + \sum_{v \in V(\mathcal{T})} \gamma_v = O(|\mathcal{T}|) + O(\sum d_r) = O(|\mathcal{T}| + d)$$

where $\sum d_r = d$ because each conflict has a root and no conflict is listed twice (see Lemma 1). Finally, note that the leaf-sets of the recursive calls of `ListAllConflicts`' form a partition of X and, therefore, each leaf of $\mathcal{T}$ has a "private" element of X that occurs only in that leaf, implying $|\mathcal{T}| \in O(|X|)$. ◀

▶ **Theorem 3.** *Given phylogenetic trees P and Q on the same set of n taxa, Algorithm 2 enumerates all d conflict triples in $O(n + d)$ time.*

4 Conclusion

We have shown how to list all conflict triples between two phylogenetic trees in $O(n+d)$ time where n is the number of taxa and d is the number of listed conflicts. This improves the previously used, trivial $\Theta(n^3)$-time algorithm that tests for each leaf-triple abc for being a conflict. The presented algorithm is fastest-possible (up to constant factors), since all algorithms solving the problem must at read the input and write the output.

Our work is located in the field of output-sensitive enumeration algorithms as well as the rising field of "FPT in P", meaning the use of parameters to speed up polynomial-time algorithms.

A simple next step is to extend the algorithm to non-binary outbranchings. More challengingly, we want to reconsider other polynomial-time enumeration problems parameterized by the length of the output list in hope to produce more "fastest-possible" algorithms. We also plan to analyze real-world phylogenetic trees to see whether the parameter is sufficiently smaller than n^3 to make it worth implementing in practice.

Acknowledgments

I thank the *Institut de Biologie Computationelle* for funding my research, as well as my colleagues Krister Swenson and Celine Scornavacca for fruitful discussions.

References

1 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *LNCS*, pages 88–94. Springer, 2000.

2 G. S. Brodal, R. Fagerberg, T. Mailund, C. N. Pedersen, and A. Sand. Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1814–1832. Society for Industrial and Applied Mathematics, 2013.

3 J. Byrka, S. Guillemot, and J. Jansson. New results on optimizing rooted triplets consistency. *Discrete Applied Mathematics*, 158(11):1136 – 1147, 2010. ISSN 0166-218X. 10.1016/j.dam.2010.03.004.

4 C. Chauve, M. Jones, M. Lafond, C. Scornavacca, and M. Weller. Constructing a consensus phylogeny from a leaf-removal distance. under review, 2017.

5 R. Cole, M. Farach-Colton, R. Hariharan, T. Przytycka, and M. Thorup. An $o(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5): 1385–1404, 2000.

6 F. V. Fomin, D. Lokshtanov, M. Pilipczuk, S. Saurabh, and M. Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *CoRR*, abs/1511.01379, 2015. URL http://arxiv.org/abs/1511.01379.

7 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

8 J. M. Hochstein and K. Weihe. Maximum s-t-flow with k crossings in o(k3n log n) time. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 843–847, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. URL http://dl.acm.org/citation.cfm?id=1283383.1283473.

9 J. Jansson, N. B. Nguyen, and W.-K. Sung. Algorithms for combining rooted triplets into a galled phylogenetic network. *SIAM Journal on Computing*, 35(5):1098–1121, 2006. 10.1137/S0097539704446529.

10 G. B. Mertzios, A. Nichterlein, and R. Niedermeier. Linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *CoRR*, abs/1703.05598, 2017. URL http://arxiv.org/abs/1703.05598.

11 V. Ranwez, A. Criscuolo, and E. J. Douzery. Supertriplets: A triplet-based supertree approach to phylogenomics. *Bioinformatics*, (26):i115–i123, 2010.