

RESEARCH

A sample article title

Xilin Yu^{*}, Thien Le
, Sarah Christensen
, Erin Molloy
and Tandy Warnow

^{*}Correspondence:
yuxilin51@gmail.com
Department of Computer Science,
University of Illinois at
Urbana-Champaign, 201 N
Goodwin, Urbana, US
Full list of author information is
available at the end of the article

Abstract

Keywords:

1 Introduction

2 Terminology and Preliminary

Throughout the paper, we consider only unrooted trees. For any tree T , let $V(T)$, $E(T)$, and $L(T)$ denote the vertex set, the edge set, and the leaf set of T , respectively. For any $v \in V(T)$, let $N_T(v)$ denote the set of neighbors of v in T . A tree is *fully resolved* if every non-leaf node has degree 3. Let \mathcal{T}_S denote the set of all fully resolved trees on leaf set S . In any tree T , each edge e induces a bipartition $\pi_e := A|B$ of the leaf set, where A and B are the leaves in the two components of $T - e$, respectively. A bipartition $A|B$ is non-trivial if both sides have size at least 2. For a tree T , $C(T) := \{\pi_e \mid e \in E(T)\}$ denotes the set of all bipartitions of T . For a fully resolved tree with n leaves, $C(T)$ contains $2n - 3$ bipartitions, exactly $n - 3$ of which are non-trivial.

A tree T' is a *refinement* of T if T can be obtained from T' by contracting a set of edges. Equivalently, T' is a refinement of T if and only if $C(T) \subseteq C(T')$.

Two bipartitions π_1 and π_2 of the same leaf set are *compatible* if and only if there exists a tree T such that $\pi_1, \pi_2 \in C(T)$. The following theorem and corollary give other categorizations of compatibility.

Theorem 1 (Theorem 2.20 of [1]) *A pair of bipartitions $A|B$ and $A'|B'$ of the same set is compatible if and only if at least one of the four pairwise intersections $A \cap A'$, $A \cap B'$, $B \cap A'$, $B \cap B'$ is empty.*

Corollary 1 *A pair of bipartitions $A|B$ and $A'|B'$ of the same set is compatible if and only if one side of $A|B$ is a subset of one side of $A'|B'$.*

A tree T restricted to a subset R of its leaf set, denoted $T|_R$, is the minimal subtree of T spanning R with nodes of degree two suppressed. A bipartition $\pi = A|B$ restricted to a subset $R \subseteq A \cup B$ is $\pi|_R = A \cap R|B \cap R$. We have the following intuitive lemma with its proof in the appendix.

Lemma 1 *Let T be a tree with leaf set S and let $\pi = A|B \in C(T)$ be a bipartition induced by $e \in E(T)$. Let $R \subseteq S$.*

- 1 *If $R \cap A = \emptyset$ or $R \cap B = \emptyset$, then $e \notin E(T|_R)$.*
- 2 *If $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$, then for any $\pi' \in C(T|_R)$ induced by $e' \in E(T|_R)$, $\pi|_R = \pi'$ if and only if $e \in P(e')$.*

Corollary 2 *Let T be a tree with leaf set S and let $\pi = A|B \in C(T)$ be a bipartition induced by $e \in E(T)$. Let $R \subseteq S$ such that $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$. Then $\pi|_R \in C(T|_R)$.*

We give an characterization of the vertex to split to add a certain bipartition into a tree in the following lemma. The proof appears in the appendix.

Lemma 2 *Let T be a tree with leaf set S and let $\pi = A|B$ be a bipartition such that $\pi \notin C(T)$ but π is compatible with $C(T)$. Then there exists a vertex $v \in V(T)$ such that there is a division of $N_T(v)$ into $N_A \cup N_B$ such that N_A (N_B respectively) is the set of neighbors which can reach vertices of A (B) but not B (A) in $T - v$. We can split v to add π to $C(T)$.*

We have the following definition of Robinson-Foulds distance and bipartition support where the former measures differences between the topology of the trees and the latter measures the similarity.

Definition 1 *For two trees T, T' with the same leaf set, the Robinson-Foulds distance of them is $\text{RF}(T, T') := |C(T) \setminus C(T')| + |C(T') \setminus C(T)|$.*

Definition 2 *For two trees T, T' with the same leaf set, the bipartition support of them is $\text{bisup}(T, T') := |C(T) \cap C(T')|$.*

The following lemma follows from simple set algebra and we include the proof in the appendix for completeness.

Lemma 3 *For two fully resolved trees $T, T' \in \mathcal{T}_S$, $\text{RF}(T, T') + 2\text{bisup}(T, T') = 4n - 6$ where $n := |S|$.*

Let T_1, T_2, \dots, T_N be a set of N fully resolved trees with leaf sets S_1, S_2, \dots, S_N , respectively. Let $S := \bigcup_{i \in [N]} S_i$.

The Robinson-Foulds supertree problem on these N trees, abbreviated RF-SUPERTREE- N , finds a supertree $T^* \in \mathcal{T}_S$ such that

$$T^* = \operatorname{argmin}_{T \in \mathcal{T}_S} \sum_{i \in [N]} \text{RF}(T|_{S_i}, T_i).$$

The maximum bipartition support supertree problem on these N trees, abbreviated BISUP-SUPERTREE- N , finds a supertree $T^* \in \mathcal{T}_S$ such that

$$T^* = \operatorname{argmax}_{T \in \mathcal{T}_S} \sum_{i \in [N]} \text{bisup}(T|_{S_i}, T_i).$$

We call $\sum_{i \in [N]} \text{bisup}(T|_{S_i}, T_i)$ the support score of T when T_i 's are clear from context. Similarly, $\sum_{i \in [N]} \text{RF}(T|_{S_i}, T_i)$ is the RF score of T .

We pay attention to the special case of the problems where there are only two input trees, i.e., RF-SUPERTREE-2 and BISUP-SUPERTREE-2. We also consider GEN-BISUP-SUPERTREE- N , which generalizes BISUP-SUPERTREE- N such that the input trees can be not fully-resolved. We have the following lemma on the relationship of RF-SUPERTREE- N and BISUP-SUPERTREE- N .

Lemma 4 *Given the same input trees, any tree $T \in \mathcal{T}_S$ is an optimal solution for RF-SUPERTREE- N if and only if it is an optimal solution for BISUP-SUPERTREE- N .*

Proof Applying Lemma 3 to $T|_{S_i}$ and T_i for all $i \in [N]$ and taking the sum of the equations, we have

$$\begin{aligned} \sum_{i \in [N]} (\text{RF}(T|_{S_i}, T_i) + 2\text{bisup}(T|_{S_i}, T_i)) &= \sum_{i \in [N]} |S_i| \\ \sum_{i \in [N]} \text{RF}(T|_{S_i}, T_i) + 2 \sum_{i \in [N]} \text{bisup}(T|_{S_i}, T_i) &= \sum_{i \in [N]} |S_i| \end{aligned}$$

Since the RF score of T and twice the support score of T sums to $\sum_{i \in [N]} |S_i|$, which is a constant fixed from the input, we know that T maximizes the support score if and only if it minimizes the RF score. \square

3 Algorithm for Bisup-Supertree-2

We present a polynomial time algorithm for BISUP-SUPERTREE-2 in this subsection. We first set up the notations for the algorithm and the analysis. Let T_1, T_2, S_1, S_2 , and X be defined as from the problem statement of BISUP-SUPERTREE-2. Let $T_1|_X$ and $T_2|_X$ be the backbone trees of T_1 and T_2 , respectively. Let Π be the set of bipartitions of X . Let Triv and NonTriv denotes the set of trivial and non-trivial bipartitions in $C(T_1|_X) \cup C(T_2|_X)$. For each $e \in E(T_i|_X)$, $i \in \{1, 2\}$, let $P(e)$ denote the path in T_i from which e is obtained by suppressing all degree-two nodes. Let $w(e)$ be the number of edges on $P(e)$.

For any bipartition π of X , let $e_i(\pi)$ denote the edge that induces π in $T_i|_X$ for $i \in \{1, 2\}$. If $e_i(\pi)$ does not exist, any set associated with it is empty. We define a weight function $w : \Pi \rightarrow \mathbb{N}_{\geq 0}$ such that for any bipartition π of X , $w(\pi) = w(e_1(\pi)) + w(e_2(\pi))$. If for any $i \in \{1, 2\}$, no $e_i(\pi)$ induces π in $T_i|_X$, then we use $w(e_i(\pi)) = 0$. Therefore, for any $\pi \notin C(T_1|_X) \cup C(T_2|_X)$, $w(\pi) = 0$. For any set F of bipartitions, $w(F) = \sum_{\pi \in F} w(\pi)$.

For each $i \in \{1, 2\}$ and each $e \in E(T_i|_X)$, let $\text{In}(e)$ be the set of internal nodes of $P(e)$. For each $v \in \text{In}(e)$, let $L(v)$ be the set of leaves in $S_i \setminus X$ whose connecting path to the backbone tree $T_i|_X$ goes through v and let $T(v)$ be the minimal subtree spanning $L(v)$ in T_i . We say $T(v)$ is an extra subtree attached to v . Consider $T(v)$ rooted at the node u which is the neighbor of v in $T(v)$. Let $\mathcal{T}(e) := \{T(v) \mid v \in \text{In}(e)\}$. Then $\mathcal{T}(e)$ is the set of extra subtrees attached to internal nodes of $P(e)$ in T_i .

We note that $|\mathcal{T}(e)| = |\text{In}(e)| = w(e) - 1$. For any bipartition $\pi \in C(T_1|_X) \cup C(T_2|_X)$, we denote $\mathcal{T}(\pi) := \mathcal{T}(e_1(\pi)) \cup \mathcal{T}(e_2(\pi))$. Let $\text{Extra}(T_i) := \bigcup_{e \in E(T_i|_X)} \mathcal{T}(e)$. Then $\text{Extra} := \text{Extra}(T_1) \cup \text{Extra}(T_2)$ denotes the set of all extra subtrees in T_1 and T_2 .

[figure to help](#)

[add intuition/overview for algorithm](#)

Algorithm 1 Max-BiSup Supertree

Input: two fully resolved trees T_1, T_2 with leaf sets S_1 and S_2 where $S_1 \cap S_2 = X \neq \emptyset$
Output: a fully resolved supertree T on leaf set $S = S_1 \cup S_2$ that maximizes the support score

- 1: compute $C(T_1|_X)$ and $C(T_2|_X)$
- 2: **for** each $\pi \in C(T_1|_X) \cup C(T_2|_X)$ **do**
- 3: compute $\mathcal{T}(e_1(\pi)), \mathcal{T}(e_2(\pi)), \mathcal{T}(\pi)$ and $w(\pi)$
- 4: construct T as a star of leaf set X with center vertex \hat{v} with the root of each $t \in \text{Extra}$ connected to \hat{v} \triangleright let $\tilde{T} = T$
- 5: **for** each $\pi \in \text{Triv}$ **do**
- 6: $T \leftarrow \text{Refine-Triv}(T_1, T_2, T, \pi, \hat{v}, \mathcal{T})$ \triangleright let $\tilde{T} = T$ after for loop
- 7: construct the incompatibility graph $G = (V_1 \cup V_2, E)$, where $V_1 = C(T_1|_X) - C(T_2|_X)$ and $V_2 = C(T_2|_X) - C(T_1|_X)$, and $E = \{(\pi, \pi') \mid \pi \in V_1, \pi' \in V_2, \pi \text{ is not compatible with } \pi'\}$
- 8: compute the maximum weight independent set I in G with weight w
- 9: let $I' = I \cup (C(T_1|_X) \cap C(T_2|_X))$, let $H(\hat{v}) = \text{NonTriv}$, let $sv(\pi) = \hat{v}$ for all $\pi \in \text{NonTriv}$
- 10: **for** each $\pi \in \text{NonTriv} \cap I'$ **do**
- 11: $T \leftarrow \text{Refine}(T_1, T_2, T, \pi, H, sv, \mathcal{T})$ \triangleright let $T^* = T$ after for loop
- 12: refine T arbitrarily at polytomies until it is fully resolved
- 13: return T

Algorithm 2 Refine-Triv

Input: two trees T_1, T_2 with leaf sets S_1 and S_2 where $S_1 \cap S_2 = X \neq \emptyset$, an unrooted tree T on leaf set $S = S_1 \cup S_2$, a trivial bipartition $\pi = a|B$ of X , a vertex $\hat{v} \in V(T)$, a dictionary \mathcal{T}
Output: an tree T' which is a refinement of T such that $C(T|_X) - C(T'|_X) = \pi$

- 1: detach all extra subtrees in $\mathcal{T}(\pi)$ from \hat{v} and attach them onto (\hat{v}, a) such that the subtrees from $\mathcal{T}(e_1(\pi))$ and subtrees from $\mathcal{T}(e_2(\pi))$ are side by side and each group respects the ordering of subtrees in T_i
- 2: return the resulting tree T'

For the analysis of the algorithm, we differentiate between two kinds of bipartitions in $C(T_1) \cup C(T_2)$. Let $\Pi_Y = \{\pi = A|B \in C(T_1) \cup C(T_2) \mid \text{either } A \cap X = \emptyset, \text{ or } B \cap X = \emptyset\}$. Let $\Pi_X = \{\pi = A|B \in C(T_1) \cup C(T_2) \mid A \cap X \neq \emptyset \text{ and } B \cap X \neq \emptyset\}$. Intuitively, Π_X is the set of bipartitions in $C(T_1) \cup C(T_2)$ that are induced by edges in the backbone trees $T_1|_X$ and $T_2|_X$ while Π_Y is the set of bipartitions in $C(T_1) \cup C(T_2)$ that are induced by edges inside extra subtrees or connecting extra subtrees to the backbone trees.

Let $p_X(T)$ and $p_Y(T)$ (we omit the parameters T_1 and T_2 for brevity) be the contributions to the support score of T from bipartitions of Π_X and Π_Y for any $T \in \mathcal{T}_S$, respectively. Formally, we have

$$p_X(T) = |C(T|_{S_1}) \cap C(T_1) \cap \Pi_X| + |C(T|_{S_2}) \cap C(T_2) \cap \Pi_X|,$$

$$p_Y(T) = |C(T|_{S_1}) \cap C(T_1) \cap \Pi_Y| + |C(T|_{S_2}) \cap C(T_2) \cap \Pi_Y|.$$

Claim 1 *If Algorithm 1 returns a tree T such that $p_X(T) \geq p_X(T')$ and $p_Y(T) \geq p_Y(T')$ for any tree T' with leaf set S , then Algorithm 1 solves BISUP-SUPERTREE-2 correctly.*

Algorithm 3 Refine

Input: two trees T_1, T_2 with leaf sets S_1 and S_2 where $S_1 \cap S_2 = X \neq \emptyset$, an unrooted tree T on leaf set $S = S_1 \cup S_2$, a nontrivial bipartition $\pi = A|B$ of X , a dictionary H , a dictionary sv , a dictionary \mathcal{T}

Output: an tree T' which is a refinement of T such that $C(T|_X) - C(T'|_X) = \pi$

- 1: $v \leftarrow sv(\pi)$
- 2: compute $N_A := \{u \in N_{t|x}(v) \mid \exists a \in A \text{ such that } u \text{ can reach } a \text{ in } T|_X - v\}$ and $N_B := \{u \in N_{t|x}(v) \mid \exists b \in B \text{ such that } u \text{ can reach } b \text{ in } T|_X - v\}$.
- 3: $V(T) \leftarrow V(T) \cup \{v_a, v_b\}$, $E(T) \leftarrow E(T) \cup \{(v_a, v_b)\}$
- 4: $H(v_a) \leftarrow \emptyset, H(v_b) \leftarrow \emptyset$
- 5: **for** each $u \in N_A \cup N_B$ **do**
- 6: **if** $u \in N_A$ **then** connect u to v_a
- 7: **else** connect u to v_b
- 8: detach all extra subtrees in $\mathcal{T}(\pi)$ from v and attach them onto (v_a, v_b) such that the subtrees from $\mathcal{T}(e_1(\pi))$ and subtrees from $\mathcal{T}(e_2(\pi))$ are side by side and each group respects the ordering of subtrees in T_i
- 9: **for** each bipartition $\pi' = A'|B' \in H(v)$ such that $\pi' \neq \pi$ **do**
- 10: detach all extra subtrees in $\mathcal{T}(\pi')$ from v
- 11: **if** $A' \subseteq A$ or $B' \subseteq A$ **then**
- 12: $sv(\pi') = v_a$ and $H(v_a) \leftarrow H(v_a) + \pi'$
- 13: attach all extra subtrees in $\mathcal{T}(\pi')$ to v_a
- 14: **else if** $A' \subseteq B$ or $B' \subseteq B$ **then**
- 15: $sv(\pi') = v_b$ and $H(v_b) \leftarrow H(v_b) + \pi'$
- 16: attach all extra subtrees in $\mathcal{T}(\pi')$ to v_b
- 17: **else**
- 18: discard π' and attach all extra subtrees in $\mathcal{T}(\pi')$ to either v_a or v_b
- 19: **for** each remaining extra subtree attached to v **do**
- 20: detach it from v and attach it to either v_a or v_b
- 21: delete v and incident edges from T
- 22: return the resulting tree T'

Proof By definition of support score, any bipartition can only contribute to the support score if it is in $C(T_1) \cup C(T_2)$. It follows by definition of Π_X and Π_Y that Π_X and Π_Y is a disjoint decomposition of $C(T_1) \cup C(T_2)$. Thus, the support score of T equals $p_X(T) + p_Y(T)$ for any tree T on leaf set S . Then if $p_X(T) \geq p_X(T')$ and $p_Y(T) \geq p_Y(T')$ for any tree T' with leaf set S , T achieves the maximum support score among all trees of leaf set S , in particular, it achieves the maximum support score among all trees in \mathcal{T}_S . \square

Therefore, it is enough for us to show that Algorithm 1 finds a tree T that maximizes both $p_X(T)$ and $p_Y(T)$ at the same time.

Lemma 5 *For any tree T of leaf set S and any refinement T' of T , $p_X(T') \geq p_X(T)$ and $p_Y(T') \geq p_Y(T)$.*

Proof Since T' is an refinement of T , $C(T|_{S_i}) \subseteq C(T'|_{S_i})$ for any $i \in \{1, 2\}$. Therefore, $|C(T|_{S_i}) \cap C(T_i) \cap \Pi_X| \leq |C(T'|_{S_i}) \cap C(T_i) \cap \Pi_X|$ for any $i \in \{1, 2\}$, and thus $p_X(T) \leq p_X(T')$. Similarly, $|C(T|_{S_i}) \cap C(T_i) \cap \Pi_Y| \leq |C(T'|_{S_i}) \cap C(T_i) \cap \Pi_Y|$ for any $i \in \{1, 2\}$, and thus $p_Y(T) \leq p_Y(T')$. \square

Lemma 6 *For any tree T of leaf set S , $p_Y(T) \leq |\Pi_Y|$. In particular, let \hat{T} be the tree constructed in Algorithm 1. Then, $p_Y(\hat{T}) = |\Pi_Y|$.*

Proof Since T_1 and T_2 has different leaf sets, $C(T_1)$ and $C(T_2)$ are disjoint. Since $\Pi_Y \subseteq C(T_1) \cup C(T_2)$, $C(T_1) \cap \Pi_Y$ and $C(T_2) \cap \Pi_Y$ forms a disjoint decomposition

of Π_Y . By definition of $p_Y(\cdot)$, for any tree T of leaf set S ,

$$\begin{aligned} p_Y(T) &= |C(T|_{S_1}) \cap C(T_1) \cap \Pi_Y| + |C(T|_{S_2}) \cap C(T_2) \cap \Pi_Y| \\ &\leq |C(T_1) \cap \Pi_Y| + |C(T_2) \cap \Pi_Y| \\ &= |\Pi_Y|. \end{aligned}$$

Fix any $\pi = A|B \in \Pi_Y$. By definition of Π_Y , either $A \cap X = \emptyset$ or $B \cap X = \emptyset$. Assume without loss of generality that $A \cap X = \emptyset$. If $\pi \in C(T_1)$, let e_1 be the edge that induces π in T_1 . Then $A \subseteq S_1 \setminus X$, which implies either e_1 is an internal edge in an extra subtree in $\text{Extra}(T_1)$, or e_1 connects one extra subtree in $\text{Extra}(T_1)$ to the backbone $T_1|_X$. In either case, the construction of \hat{T} ensures that $\pi \in C(\hat{T}|_{S_1})$. Similarly if $\pi \in C(T_2)$, then $\pi \in C(\hat{T}|_{S_2})$ by construction. Therefore, each bipartition $\pi \in \Pi_Y$ contributes 1 to $|C(\hat{T}|_{S_i}) \cap C(T_i) \cap \Pi_Y|$ for exactly one $i \in \{1, 2\}$ and thus it contributes 1 to $p_Y(\hat{T})$. Hence, $p_Y(\hat{T}) = |\Pi_Y|$. \square

Claim 2 *Let \hat{T} be the tree constructed in Algorithm 1, then $p_X(\hat{T}) = 2|X|$.*

Proof For each $v \in X$, consider the bipartition $\pi_v = \{v\} \mid S \setminus \{v\}$ of \hat{T} induced by the edge that connects the leaf v to the center \hat{v} . It is easy to see that $\pi_v|_{S_i} = \{v\} \mid S_i \setminus \{v\} \in C(T_i)$ for any $i \in \{1, 2\}$ as $\pi_v|_{S_i}$ is a trivial bipartition of S_i . By Lemma 2, we have $\pi_v|_{S_i} \in \hat{T}|_{S_i}$. We also know $\pi_v|_{S_i} \in \Pi_X$ as both sides of π_v has non-empty intersection with X . Thus, $\pi_v|_{S_i} \in C(\hat{T}|_{S_i}) \cap C(T_i) \cap \Pi_X$ for any $i \in \{1, 2\}$. So for each $v \in X$, $\pi_v|_{S_1}$ and $\pi_v|_{S_2}$ each contributes 1 to $p_X(\hat{T})$. Therefore, $p_X(\hat{T}) \geq 2|X|$.

Fix any bipartition $\pi = A|B$ induced by any other edge of \hat{T} such that $\pi|_{S_i} \in C(\hat{T}|_{S_i})$ for some $i \in \{1, 2\}$. By construction of \hat{T} , the edge inducing π is either inside an extra subtree or connecting the root of an extra subtree to the center. Therefore, either $A \subseteq S \setminus X$ or $B \subseteq S \setminus X$, which implies $\pi|_{S_i} \notin \Pi_X$ for any $i \in \{1, 2\}$. Hence, there is no other bipartition of \hat{T} such that when restrict to S_i contributes to $p_X(\hat{T})$. Therefore, $p_X(\hat{T}) = 2|X|$. \square

Lemma 7 *Let $\pi = A|B$ be a bipartition of X . Let T be a tree of leaf set S such that $\pi \notin C(T|_X)$ and all bipartitions in $C(T|_X)$ are compatible with π . Let T' be a refinement of T such that for all $\pi' \in C(T'|_{S_i}) \setminus C(T|_{S_i})$ for some $i \in \{1, 2\}$, $\pi'|_X = \pi$. Then, $p_X(T') - p_X(T) \leq w(\pi)$.*

Proof By definition of $p_X(\cdot)$,

$$\begin{aligned} &p_X(T') - p_X(T) \\ &= |C(T'|_{S_1}) \cap C(T_1) \cap \Pi_X| + |C(T'|_{S_2}) \cap C(T_2) \cap \Pi_X| \\ &\quad - (|C(T|_{S_1}) \cap C(T_1) \cap \Pi_X| + |C(T|_{S_2}) \cap C(T_2) \cap \Pi_X|) \\ &= |(C(T'|_{S_1}) \setminus C(T|_{S_1})) \cap C(T_1) \cap \Pi_X| + |(C(T'|_{S_2}) \setminus C(T|_{S_2})) \cap C(T_2) \cap \Pi_X| \\ &= \sum_{i=1,2} |(C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X|. \end{aligned}$$

Therefore, we only need to prove that $\sum_{i=1,2} |(C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \leq w(\pi)$. For any $\pi' \in (C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for any $i \in \{1, 2\}$, we have $\pi'|_X = \pi$.

We differentiate three different cases for the proof of the above statement: 1) $\pi \notin C(T_1|_X) \cup C(T_2|_X)$, 2) $\pi \in C(T_1|_X) \Delta C(T_2|_X)$, 3) $\pi \in C(T_1|_X) \cap C(T_2|_X)$.

Case 1): Let $\pi \notin C(T_1|_X) \cup C(T_2|_X)$. Since no edge induces π in $T_1|_X$ or $T_2|_X$, we have $w(\pi) = 0$. Assume for contradiction that there exists a bipartition $\pi' \in (C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for some $i \in \{1, 2\}$. Since $\pi \notin C(T_1|_X) \cup C(T_2|_X)$ and $\pi'|_X = \pi$, by Corollary 2, $\pi' \notin C(T_i)$ for any $i \in \{1, 2\}$. This contradicts with the fact that $\pi' \in C(T_i)$ for some $i \in \{1, 2\}$. Therefore, the assumption that there exists such a bipartition π' is wrong and $\sum_{i=1,2} |(C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| = 0 \leq w(\pi)$.

Case 2): Let $\pi \in C(T_1|_X) \Delta C(T_2|_X)$. Assume without loss of generality that $\pi \in C(T_1|_X) \setminus C(T_2|_X)$. Then, we have $w(\pi) = w(e_1)$. Let $\pi' \in (C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for some $i \in \{1, 2\}$. Since $\pi'|_X = \pi$ and $\pi \notin C(T_2|_X)$, by Corollary 2, we have $\pi' \notin C(T_2)$. Since $\pi' \in C(T_i)$ for some $i \in \{1, 2\}$, it must be that $\pi' \in C(T_1)$. By Lemma 1, the edge which induces π' in T_1 is an edge on $P_1(e_1)$. Since there are $w(e_1)$ edges on $P_1(e_1)$, there are at most $w(e_1)$ distinct such bipartitions π' 's, and thus the statement is proved.

Case 3): Let $\pi \in C(T_1|_X) \cap C(T_2|_X)$. Then we have $w(\pi) = w(e_1) + w(e_2)$. Fix any $\pi' \in (C(T'|_{S_1}) \setminus C(T|_{S_1})) \cap C(T_1) \cap \Pi_X$. Since $\pi' \in C(T_1)$ and $\pi'|_X = \pi \in C(T_1|_X)$, by Lemma 1, the edge e' that induces π' is an edge on $P_1(e_1)$. Recall that $w(e_1) = |P_1(e_1)|$, then we have $|(C(T'|_{S_1}) \setminus C(T|_{S_1})) \cap C(T_1) \cap \Pi_X| \leq |P_1(e_1)| = w(e_1)$. Similarly, $|(C(T'|_{S_2}) \setminus C(T|_{S_2})) \cap C(T_2) \cap \Pi_X| \leq |P_2(e_2)| = w(e_2)$. Therefore, $\sum_{i=1,2} |(C(T'|_{S_i}) \setminus C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \leq w(\pi)$. \square

Lemma 8 *For any compatible set F of bipartitions of X , let T be a tree of leaf set S such that $C(T|_X) = F$. Then $p_X(T) \leq \sum_{\pi \in F} w(\pi)$.*

Proof Fix an arbitrary ordering of bipartitions in F and let them be $\pi_1, \pi_2, \dots, \pi_k$, where $k = |F|$. Let $F_i = \{\pi_1, \dots, \pi_i\}$ for any $i \in \{0, 1, \dots, k\}$. In particular, $F_0 = \emptyset$ and $F_k = F$. Let T^i be obtained by contracting any edge e in T such that $\pi_e \in \Pi_X$ and $\pi_e|_X \notin F_i$. Then $C(T^i|_X) = F_i$. In particular, we know $C(T^0|_X) = \emptyset$. By construction, T^i is a refinement of T^{i-1} for any $i \in \{1, 2, \dots, k\}$ such that for any $\pi' \in C(T^i) \setminus C(T^{i-1})$, $\pi'|_X = \pi_i$. Then by Lemma 7, $p_X(T^i) - p_X(T^{i-1}) \leq w(\pi_i)$. Therefore,

$$p_X(T) - p_X(T^0) = \sum_{i=1}^k p_X(T^i) - p_X(T^{i-1}) \leq \sum_{i=1}^k w(\pi_i).$$

We also know that $p_X(T^0) = 0$ (expand on this) and thus $p_X(T) \leq \sum_{\pi_i \in F} w(\pi_i)$ as desired. \square

Claim 3 Let \tilde{T} be the tree constructed in Algorithm 1, then $p_X(\tilde{T}) = \sum_{\pi \in \text{Triv}} w(\pi)$.

Proof Let $\pi = a|B$ be a trivial bipartition of X . We know both $e_1(\pi)$ and $e_2(\pi)$ exist and abbreviate them by e_1 and e_2 . Consider all extra subtrees in $\mathcal{T}(e_1)$ and index the extra subtrees as t_1, t_2, \dots, t_p such that t_1 is the closest a in T_1 and $p = |\mathcal{T}(e_1)| = w(e_1) - 1$. Similarly, index the extra subtrees in $\mathcal{T}(e_2)$ to be t'_1, t'_2, \dots, t'_q such that t'_1 is closest to a in T_2 and $q = |\mathcal{T}(e_2)| = w(e_2) - 1$. For each $k \in [w(e_1)]$, we define

$$A_k := \bigcup_{i=1}^{k-1} L(t_i) \cup a, \quad \pi_k := A_k|S_1 \setminus A_k,$$

and for each $k \in [w(e_2)]$, we define

$$A'_k := \bigcup_{i=1}^{k-1} L(t'_i) \cup a, \quad \pi'_k := A'_k|S_2 \setminus A'_k.$$

It follows by definition that π_k for any $k \in [w(e_1)]$ is the bipartition induced by the k th edge on $P(e_1)$ in T_1 numbered from the side of a , which implies $\pi_k \in C(T_1)$ for any $k \in [w(e_1)]$. Similarly, $\pi'_k \in C(T_2)$ for any $k \in [w(e_2)]$. In particular, we notice that $\pi_1 = \pi'_1 = \pi$. Clearly, all these bipartitions are also in Π_X because both sides have none empty intersection with X .

Since Algorithm 2 moves all extra subtrees in $\mathcal{T}(\pi)$ onto the edge (\hat{v}, a) and orders them such that extra subtrees in $\mathcal{T}(e_1)$ (and $\mathcal{T}(e_2)$, respectively) follows the order of trees within the group on T_1 (T_2), i.e., t_1 (t'_1) is closest to a and t_p (t'_q) is furthest away, it is easy to see that we also have $\pi_k \in C(T|_{S_1})$ for any $k \in [w(e_1)]$ and $\pi'_k \in C(T|_{S_2})$ for any $k \in [w(e_2)]$, where T is the tree obtained after add π to the backbone through Algorithm 2. Therefore, $|C(T|_{S_1}) \cap C(T_1|_X) \cap \Pi_X|$ is increased by $w(e_1) - 1$ by the algorithm as $\pi_k \notin C(T|_{S_1})$ before the algorithm for all $k \in []$ except $k = 1$. Similarly, $|C(T|_{S_1}) \cap C(T_2|_X) \cap \Pi_X|$ is increased by $w(e_2) - 1$, so $p_X(T)$ is increased by $w(e_1) + w(e_2) - 2 = w(\pi) - 2$ by running Algorithm 2 on T and π . Since running Algorithm 2 to add other trivial bipartitions of X never destroys the bipartitions of S_1 or S_2 already in T , we have $p_X(\tilde{T}) = p_X(\hat{T}) + \sum_{\pi \in \text{Triv}} (w(\pi) - 2) = 2|X| + \sum_{\pi \in \text{Triv}} (w(\pi) - 2) = \sum_{\pi \in \text{Triv}} w(\pi)$. \square

Lemma 9 At any stage of the Algorithm 1 at and after line 11, we have the following invariants of T and the auxiliary data structures H and sv :

- 1 For any bipartition $\pi \in \text{NonTriv}$, $sv(\pi)$ is the vertex to split to add π to $C(T|_X)$. For any internal vertex v , the set of bipartitions $H(v) \subseteq \text{NonTriv}$ is the set of bipartitions which can be added to $C(T|_X)$ by splitting v .
- 2 For any $\pi = A|B \in H(v)$, for all $t \in \mathcal{T}(\pi)$, the root of t is a neighbor of v .
- 3 For any $\pi = A|B \in C(T|_X)$ induced by edge e and any $\pi' = A'|B'$ that is compatible with π . Let $C(A), C(B)$ be the two components containing the leaves of A and B in $T|_X - e$. If A' or B' is a subset of A , then all $t \in \mathcal{T}(\pi')$ are attached to an edge or a vertex in $C(A)$. If A' or B' is a subset of B , then all $t \in \mathcal{T}(\pi')$ are attached to an edge or a vertex in $C(B)$.

Proof We prove the invariants by induction on the number of refinement steps k performed on T . When $k = 0$, we have $T = \tilde{T}$ and $C(T|_X) = \text{Triv}$ and thus $T|_X$ is a star with leaf set X . Thus all bipartitions in NonTriv are compatible with T . For any $\pi \in \text{NonTriv}$, \hat{v} is the vertex to refine in $T|_X$ to add π to $C(T|_X)$. Therefore, $sv(\pi)$ and $H(\hat{v})$ are both correct. The roots of all extra subtrees in $\mathcal{T}(\pi)$ for any $\pi \in \text{NonTriv}$ are all connected to \hat{v} , so invariant 2 also holds. For any $\pi \in C(T|_X) = \text{Triv}$, let $\pi = a|B$. Therefore, $C(a)$ is the vertex a and $C(B)$ is the rest of the star of $T|_X$. For any bipartition $\pi' \neq \pi$, either π' is trivial and thus $\mathcal{T}(\pi')$ are all attached to the edge connecting the leaf with \hat{v} or π' is non-trivial and thus all of $\mathcal{T}(\pi')$ are attached to \hat{v} . In either case, any extra subtree in $\mathcal{T}(\pi')$ is attached to a vertex or an edge in $C(B)$. This proves invariant 3 and thus concludes our proof for the base case.

Assume that all invariants hold after any $k' < k$ steps of refinement. Let $\pi = A|B$ be the bipartition to add in the k th refinement step. We will show that after the k th refinement step, i.e., one execution of Algorithm 3, the invariants still hold for the resulting tree T' . Since $sv(\pi) = \pi$ at the beginning of Algorithm 3, π can be added to $C(T|_X)$ by splitting v , i.e., there exists a division of neighbors of v in $T|_X$ into $N_A \cup N_B$ such that N_A (or N_B respectively) consists of neighbors of v which can reach vertices of A (or B) in $T|_X - v$. Then, the algorithm correctly connects N_A to v_a and N_B to v_b so the new edge (v_a, v_b) induces the bipartition $\pi = A|B$ in $T|_X$. For any vertex $u \neq v$ and any bipartition $\pi' \in H(u)$, the invariants 1 and 2 still hold after Algorithm 3 as we do not change $H(u)$, $sv(\pi')$, or the extra subtrees attached to u . For any bipartition $\pi' = A'|B' \in H(v)$ such that $\pi' \neq \pi$, if π' is not compatible with π , then it is cannot be added to $C(T'|_X)$ since π is added, so the algorithm correctly discard π' and does not add it to $H(v_a)$ or $H(v_b)$. If π' is compatible with π , we will show that the invariants 1 and 2 hold for π' .

By Corollary 1, one side of $A'|B'$ is a subset of one side of $A|B$. Consider the case where one side of $A'|B'$ is a subset of A . The other case is symmetrical. Also assume without loss of generality that $A' \subseteq A$, then $B \subseteq B'$. In this case, Algorithm 3 adds π' to $H(v_a)$ and set $sv(\pi) = v_a$. We will show that this step preserves the invariants. Since $\pi' \in H(v)$, before adding π we can split v to add π' to $C(T|_X)$. Then there exists a division of neighbors of v in $T|_X$ into $N_{A'}$ and $N_{B'}$ such that $N_{A'}$ (or $N_{B'}$, respectively) consists of neighbors of v which can reach vertices of A' (or B') in $T|_X - v$. It is easy to see that $N_{A'} \subseteq N_A$ and $N_B \subseteq N_{B'}$. Since $N_A \cup N_B = N_{A'} \cup N_{B'} = N_{T|_X}(v)$, we have $N_A \setminus N_{A'} = N_{B'} \setminus N_B$. Since all vertices in N_B are connected to v_b in T' while vertices in $N_{B'} \setminus N_B$ are connected to v_a , $N_{B'} \setminus N_B \cup \{v_b\}$ is the set of all neighbors of v_a which can reach leaves of B' in $T'|_X - v_a$. Then $N_{T'|_X}(v_a) = N_A \cup \{v_b\} = N_{A'} \cup (N_A \setminus N_{A'} \cup \{v_b\}) = N_{A'} \cup (N_{B'} \setminus N_B \cup \{v_b\})$ implies that $N_{A'}$ and $N_{B'} \setminus N_B \cup \{v_b\}$ gives an division of neighbors of v_a such that $N_{A'}$ are the neighbors that can reach leaves of A' in $T'|_X - v_a$ and $N_{B'} \setminus N_B \cup \{v_b\}$ are the neighbors that can reach leaves of B' in $T'|_X - v_a$. Such a division proves that v_a is the correct vertex to refine in $T'|_X$ to add π' to $C(T'|_X)$ after the k th refinement. Therefore, invariant 1 holds with respect to π' . Since $\pi' \in H(v)$ before adding π , we also have for all $t \in \mathcal{T}(\pi')$, the root of t is connected to v before adding π .

Then, Algorithm 3 attaches roots of all trees in $\mathcal{T}(\pi')$ to v_a and since $\pi' \subseteq H(v_a)$, invariant 2 holds for π' .

We have showed that invariants 1 and 2 hold for the tree T' with the auxiliary data structure H for all internal nodes and data structure sv for all bipartitions still compatible T' . Next we show that invariant 3 holds. Since π is the only bipartition added to $C(T'|_X)$, we only need to show two things: 1) for any $pi' = A'|B' \in C(T|_X)$, trees in $\mathcal{T}(pi')$ are attached to $C(A')$ or $C(B')$ appropriately, 2) for any π'' compatible with π , trees in $\mathcal{T}(\pi'')$ are attached to $C(A)$ or $C(B)$ appropriately. For 1), we assume without loss of generality that $\pi' = A'|B'$ such that $A' \subseteq A$, then $B \subseteq B'$. Therefore, $(v_a, v_b) \in C(B')$ and since all $t \in \mathcal{T}(\pi)$ are attached onto (v_a, v_b) by Algorithm 3, the invariant 3 holds with respect to π' . For 2), we assume without loss of generality that $\pi'' = A''|B''$ is compatible with π such that $A'' \subseteq A$. Then either $\pi'' \in C(T|_X)$ and thus π'' is induced by an edge e'' which is in $C(A)$ or $\pi'' \notin C(T|_X)$ and thus there exists a vertex v in $C(A)$ such that we can add π'' to $C(T|_X)$ by splitting v . In the former case, all $t \in \mathcal{T}(\pi'')$ are attached on e'' , in the latter case, all $t \in \mathcal{T}(\pi'')$ are attached on v by invariant 2 before adding π . Therefore, in both cases the invariant 3 holds with respect to π , which concludes the proof for invariant 3 and our inductive proof overall. \square

Lemma 10 *Let T be a tree from Algorithm 1 before a refinement step. Let $\pi = A|B \in \text{NonTriv} \cap I'$. Let T' be a refinement of T obtained from running Algorithm 3 on T and π , with the auxiliary data structures H , sv , and \mathcal{T} . Then, $p_X(T') - p_X(T) = w(\pi)$.*

Proof We know T is a refinement of \tilde{T} . Since $C(\tilde{T}|_X) = \text{Triv} \subseteq C(T_1|_X) \cap C(T_2|_X) \subseteq I'$ and we only refine by bipartitions from I' , we know $C(T|_X) \subseteq I'$. Since $\pi \in \text{NonTriv} \cap I'$ and I' is a compatible set, all bipartitions in $C(T|_X)$ are compatible with π . Thus it is possible to refine $T|_X$ with π such that $C(T'|_X) - C(T|_X) = \pi$. By invariant 1 of Lemma 9, $v = sv(\pi)$ is the vertex to split to add π to $T|_X$ and thus the Algorithm 3 correctly splits v into v_a and v_b and connects them to appropriate neighbors such that in $T'|_X$, (v_a, v_b) induces π .

We abbreviate $e_1(\pi)$ and $e_2(\pi)$ by e_1 and e_2 . Consider all extra subtrees in $\mathcal{T}(e_1)$ and index the extra subtrees as t_1, t_2, \dots, t_p such that t_1 is the closest a in T_1 and $p = |\mathcal{T}(e_1)| = w(e_1) - 1$. Similarly, index the extra subtrees in $\mathcal{T}(e_2)$ to be t'_1, t'_2, \dots, t'_q such that t'_1 is closest to a in T_2 and $q = |\mathcal{T}(e_2)| = w(e_2) - 1$. Let $C_i(A)$, $C_i(B)$ be the component in $T_i|_X - e_i$ that contains the leaf set A or B , respectively. We define the extra subtrees in T_i on the side of A or on the side of B to be

$$\mathcal{T}_i(A) = \bigcup_{e \in C_i(A)} \mathcal{T}(e), \mathcal{T}_i(B) = \bigcup_{e \in C_i(B)} \mathcal{T}(e).$$

For any set \mathcal{T} of trees, let $L(\mathcal{T})$ denote the union of the leafset of trees in \mathcal{T} . We note that $\text{Extra}(T_i) = \mathcal{T}_i(A) \cup \mathcal{T}_i(B) \cup \mathcal{T}(e_i)$ and thus $A \cup L(\mathcal{T}_i(A)) \cup L(\mathcal{T}(e_i)) \cup L(\mathcal{T}_i(B)) \cup B = S_i$ for $i \in \{1, 2\}$.

For each $k \in [w(e_1)]$, we define $A_k := \bigcup_{i=1}^{k-1} L(t_i) \cup L(\mathcal{T}_1(A)) \cup A$, $\pi_k := A_k|_{S_1 \setminus A_k}$, and for each $k \in [w(e_2)]$, we define $A'_k := \bigcup_{i=1}^{k-1} L(t'_i) \cup L(\mathcal{T}_2(A)) \cup A$, $\pi'_k := A'_k|_{S_2 \setminus A'_k}$. We know that for each $k \in [w(e_1)]$, $S_1 \setminus A_k = \bigcup_{i=k}^p L(t_i) \cup L(\mathcal{T}_1(B)) \cup B$. Thus, for any $k \in [w(e_1)]$, π_k is the bipartition induced by the k th edge on $P(e_1)$ in T_1 , where the edges are numbered from the side of A . Therefore, $\pi_k \in C(T_1)$ for any $k \in [w(e_1)]$. Similarly, $\pi'_k \in C(T_2)$ for any $k \in [w(e_2)]$.

Since for any $k \in [w(e_1)]$, $A_k \cap X = A \neq \emptyset$ and $(S_1 \setminus A_k) \cap X = B \neq \emptyset$, we have $\pi_k|_X = \pi$ and $\pi_k \in \Pi_X$. Similarly, for each $k \in [w(e_2)]$, $\pi'_k \in \Pi_X$ and $\pi'_k|_X = \pi$. We also know that since $\pi \notin C(T|_X)$, by Corollary 2, $\pi_k \notin C(T|_{S_1})$ for any $k \in [w(e_1)]$ and $\pi'_k \notin C(T|_{S_2})$ for any $k \in [w(e_2)]$. We claim that $\pi_k \in C(T'|_{S_1})$ for all $k \in [w(e_1)]$ and $\pi'_k \in C(T'|_{S_2})$ for all $k \in [w(e_2)]$. Then, $|C(T'|_{S_1}) \cap C(T_1) \cap \Pi_X| - |C(T|_{S_1}) \cap C(T_1) \cap \Pi_X| = w(e_1)$ and $|C(T'|_{S_2}) \cap C(T_2) \cap \Pi_X| - |C(T|_{S_2}) \cap C(T_2) \cap \Pi_X| = w(e_2)$, and thus $p_X(T') - p_X(T) = w(e_1) + w(e_2) = w(\pi)$.

Now we only need to prove the claim. Fix $k \in [w(e_1)]$, we will show that $\pi_k \in C(T'|_{S_1})$. The claim of $\pi'_k \in C(T'|_{S_2})$ for any $k \in [w(e_2)]$ follows by symmetry. By invariant 2 of Lemma 9, we know that all extra subtrees of $\mathcal{T}(e_1) \cup \mathcal{T}(e_2)$ were attached to v at the beginning of Algorithm 3 and thus the algorithm attaches them all onto (v_a, v_b) in the order of t_1, t_2, \dots, t_p , where t_1 is closest to A . Let the attaching vertex of t_i onto (v_a, v_b) be v_i for any $i \in [w(e_1)]$. Then we note $P((v_a, v_b))$ is the path from v_a to v_1, v_2, \dots, v_p and then to v_b . Fix any $t \in \mathcal{T}_1(A)$, let e be the edge such that $t \in \mathcal{T}(e)$ in $T_1|_X$, i.e., e is the edge t attaches to in $T_1|_X$. Since e and e_1 are both edges of $T_1|_X$, $\pi_e = A'|B'$ is compatible with π . By definition of $\mathcal{T}_1(A)$, $e \in C_1(A)$ and thus one component in $T_1|_X - e$ is a subgraph of $C_1(A)$. Therefore, the leaves of that component is a subset of A , i.e., $A' \subseteq A$ or $B' \subseteq A$. By invariant 3 of Lemma 9, t is in the component with leaf set A in $T'|_X - (v_a, v_b)$ and thus in a component with vertices of A in $T' - P(e_1)$. Therefore, if we delete any edge on $P((v_a, v_b))$ in T' , t is in the same component as A . Similarly, for any $t \in \mathcal{T}_1(B)$, all leaves of t are in the same component with B if we delete any edge on $P((v_a, v_b))$. In particular, consider $T'|_{S_1} - (v_{k-1}, v_k)$, the leaves of the component containing v_{k-1} is exactly $A \cup L(\mathcal{T}_1(A)) \cup \bigcup_{i=1}^{k-1} L(t_i) = A_k$. Therefore, the edge (v_{k-1}, v_k) induces the bipartition $A_k|_{S_1 \setminus A_k}$ in $T'|_{S_1}$. Hence, $\pi_k \in C(T'|_{S_1})$ as desired. \square

Let G, I, I' be defined as in Algorithm 1. Let $G' = (V'_1 \cup V'_2, E')$ be the full incompatibility graph of $T_1|_X$ and $T_2|_X$ such that $V'_1 = C(T_1|_X)$ and $V'_2 = C(T_2|_X)$, and $E' = \{(\pi, \pi') \mid \pi \in V'_1, \pi' \in V'_2, \pi \text{ is not compatible with } \pi'\}$.

Claim 4 I' is a maximum weight compatible subset of $C(T_1|_X) \cup C(T_2|_X)$ with weight function w .

Proof Since all bipartitions in $C(T_1|_X)$ are compatible with each other and all bipartitions in $C(T_2|_X)$ are compatible with each other, all bipartitions in $C(T_1|_X) \cap C(T_2|_X)$ are compatible with all bipartitions in $C(T_1|_X) \cup C(T_2|_X)$. Therefore, $C(T_1|_X) \cap C(T_2|_X)$ is a set of isolated vertices in G' . Since $V \setminus V' =$

$C(T_1|_X) \cap C(T_2|_X)$, we know that G' is just G with extra isolated vertices. Therefore, it is easy to see that I' is a maximum weight independent set in G' . Since G' is the full incompatibility graph on $V' = C(T_1|_X) \cup C(T_2|_X)$, I' corresponds to a maximum weight compatible subset of $C(T_1|_X) \cup C(T_2|_X)$. \square

Claim 5 *Let T^* be the tree defined in Algorithm 1, $p_X(T^*) \geq p_X(T)$ for any tree T of leafset S .*

Proof Let T be any tree of leaf set S . Let $F = C(T|_X)$. Then by Lemma 8, $p_X(T) \leq \sum_{\pi \in F} w(\pi) = \sum_{F \cap (C(T_1|_X) \cup C(T_2|_X))} w(\pi)$. The equality follows from that $w(\pi) = 0$ for any $\pi \notin C(T_1|_X) \cup C(T_2|_X)$. Since $F \cap (C(T_1|_X) \cup C(T_2|_X))$ is a compatible subset of $C(T_1|_X) \cup C(T_2|_X)$, $w(F \cap (C(T_1|_X) \cup C(T_2|_X))) \leq w(I')$ by Claim 4. We also know by Claim 3 and Lemma 10, $p_X(T^*) = \sum_{\pi \in I} w(\pi) + \sum_{\pi \in (C(T_1|_X) \cap C(T_2|_X))} w(\pi) = w(I')$. Therefore, $p_X(T) \leq w(I') = p_X(T^*)$. \square

Theorem 2 *Algorithm 1 correctly solves BISUP-SUPERTREE-2 in $O(n^2|X|)$ time, where $n = \max\{|S_1|, |S_2|\}$.*

Proof From Lemma 6 and Claim 5 and Lemma 5, we know that T^* defined in Algorithm 1 satisfy that $p_X(T^*) \geq p_X(T)$ and $p_Y(T^*) \geq p_Y(T)$ for any tree with leaf set S . Then by Claim 1, Algorithm 1 correctly solves BISUP-SUPERTREE-2. Next we analyze the running time of Algorithm 1.

First we analyze the running time of Algorithm 3. Line 2 takes $O(|X|^2)$ time as we can do DFS search in $T|_X - v$ from every neighbor of v and check in $O(|X|)$ time if any newly discovered vertex is a vertex in A or B and label the neighbors of v accordingly. Line 5 to 7 takes $O(|X|)$ time as v has $O(|X|)$ neighbors. Line 8 takes $O(n)$ time as there are at most $O(n)$ extra subtrees in $\mathcal{T}(\pi)$. Line 9 to 18 takes $O(n + |X|^2)$ time as there are at most $O(n)$ extra subtrees to be moved and there are at most $O(|X|)$ bipartitions in $H(v)$ with each of the containment conditions checkable in $O(|X|)$ time if labels of leaves are stored in a pre-processed sorted list instead of a set. Line 19 can again take $O(n)$ time. The rest of the algorithm takes constant time. Overall, Algorithm 3 runs in $O(n + |X|^2)$ time. Algorithm 2 essentially performs line 8 of Algorithm 3 for trivial bipartitions and runs in $O(n)$ time.

For Algorithm 1, line 1 takes $O(n^2 + n|X|^2)$ time as we need to compute $\pi_e|_X$ and take the union for all $e \in E(T_1) \cup E(T_2)$. There are $O(n)$ edges in $E(T_1) \cup E(T_2)$. Computing $\pi_e|_X$ takes $O(n)$ time by running DFS on $T_i - e$ for $e \in E(T_i)$ to obtain π_e and taking intersection of both sides of e with X , separately. Taking union of the bipartitions takes $O(n|X|^2)$ time as whenever we add a new bipartition, it needs to be compared to the $O(|X|)$ existing ones in the set and since both have size $O(|X|)$ the comparison can be done in $O(|X|)$ time again if they are represented by two sorted list instead of two sets. In this step, we can always maintain a set of edges in T_i for each bipartition π such that $\pi_e|_X = \pi$. In line 2 to 3, we first compute the path $P(e_i(\pi))$ for each π by assembling the set of edges associated with π from last step into a path. This takes $O(n^2)$ time by counting the times any vertex appear in the set of edges and those which only appear once are the end of the path

while those appear twice are internal nodes of the path. Then we can find the extra subtree attached to each internal vertex v of the path $P(e_i(\pi))$ by DFS in $T_i - v$ from the neighbor of v that does not appear in the path. Therefore, the procedure takes $O(n)$ time for each bipartition and thus takes $O(n^2|X|)$ time overall. Line 4 takes $O(n)$ time and line 5 – 6 runs Algorithm 2 $O(|X|)$ times using a total of $O(n|X|)$ time. Line 7 constructs an incompatibility graph with $|V_1 \cup V_2| = O(|X|)$ and $|E| = O(|X|^2)$ in $O(|X|^3)$ time as compatibility of two bipartitions can be checked in $O(|X|)$ time. Line 8 runs in $O(|V||E|) = O(|X|^3)$ time. (write algorithm seperately?) Line 9 runs in $O(|X|)$ time. Line 10 – 11 runs Algorithm 3 $O(|X|)$ times with a total of $O(n|X| + |X|^3)$ time. Line 12 runs random refinement steps at most $|X|$ times, each of which can take $O(n)$ time. Since $|X| \leq n$, $|X|^3 \leq n|X|^2 \leq n^2|X|$, and thus, the overall running time of the algorithm is dominated by $O(n^2|X|)$. \square

3.1 Hardness for BISUP-SUPERTREE-3

In this subsection, we show that GEN-BISUP-SUPERTREE- N is NP-hard even when $N = 3$. We reduce the maximum weight independent set problem on tripartite graphs to GEN-BISUP-SUPERTREE-3. We first reproduce the theorem that proves the maximum weight independent set problem on tripartite graphs to be NP-hard.

Theorem 3 GEN-BISUP-SUPERTREE-3 is NP-hard.

Appendix A: Proofs from Section 2

Proof of Lemma 3

Proof By definition we have

$$\begin{aligned}
 & \text{RF}(T, T') + 2\text{bisup}(T, T') \\
 &= |C(T) \setminus C(T')| + |C(T') \setminus C(T)| + 2|C(T) \cap C(T')| \\
 &= |C(T) \setminus C(T') \cup (C(T) \cap C(T'))| + |C(T') \setminus C(T) \cup (C(T) \cap C(T'))| \\
 &= |C(T)| + |C(T')| \\
 &= 4n - 6.
 \end{aligned}$$

The last equality follows from that every fully resolved tree with n leaves has $2n - 3$ bipartitions. \square

Proof of Lemma 1

Proof Let T_R be the minimal subtree of T that spans R . It follows that the leaf set of T_R is R and $T|_R$ is obtained from T_R by suppressing all degree-two nodes. Let $\pi' = A'|B'$. By definition of e inducing $\pi = A|B$, the vertices of A are all disconnected from vertices of B in $T - e$. If $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$, then e is necessary to connect $R \cap A$ with $R \cap B$, and thus e must be in any tree spanning R and in particular $e \in E(T_R)$. Since T_R is a subgraph of T , the two components in $T_R - e$ are subgraphs of the two components in $T - e$. Thus, the leaves of the two components in $T_R - e$ are exactly $R \cap A$ and $R \cap B$. We also

know that suppressing degree-two nodes does not change the connectivity between any leaves so the leaves of the two components in $T_R - P(e')$ (with vertices on the path also deleted) are the same as the leaves of the two components in $T|_R - e'$, which are A' and B' . If $e \in P(e')$, since all internal nodes of $P(e')$ have degree two with both incident edges on $P(e')$, there is no leaf which exists in any of the two components in $T_R - e$ but does not exist in the corresponding component in $T_R - P(e')$. Therefore, $\pi|_R = R \cap A|_R \cap B = A'|_B' = \pi'$. If $e \notin P(e')$, then since $e \in E(T_R)$, there must exist $e'' \in E(T|_R)$ such that $e'' \neq e'$ and $e \in P(e'')$. By the argument above, $\pi|_R = \pi''$ where π'' is the bipartition induced by e'' in $T|_R$. Since $e'' \neq e'$, we know $\pi' \neq \pi''$ and thus $\pi|_R \neq \pi'$. This concludes our proof that $\pi|_R = \pi'$ if and only if $e \in P(e')$. \square

Proof of Lemma 2

Proof Since π is compatible with $C(T)$ but $\pi \notin C(T)$, by definition, there exists a tree T' such that $C(T') = C(T) + \pi$. Let $e = (v_a, v_b)$ be the edge that induces π in T' such that the component containing v_a in $T' - (v_a, v_b)$ has leafset A and the component containing v_b in $T' - (v_a, v_b)$ has leafset B . If we contract (v_a, v_b) , then T' becomes T . Let v be the vertex of T corresponding to the vertex of T' created from contracting (v_a, v_b) . Let N_a, N_b be the neighbors of v_a and v_b in $T' - (v_a, v_b)$, respectively. Let N_A, N_B be vertices in T corresponding to N_a and N_b . We note that $N_A \cup N_B = N_T(v)$. Since in $T' - (v_a, v_b)$, no vertex in N_a can reach any vertex of B , the same is true in $T' - v_a - v_b$. But each vertex in N_a can reach some vertex of A in $T' - v_a - v_b$ by either being a leaf in A or in the same component of some leaf in A . Similarly, in $T' - v_a - v_b$, no vertex of N_b can reach any vertex of A , but every vertex of N_b can reach some vertex of B . By construction, $T' - v_a - v_b$ has the same topology as $T - v$, and thus N_A (and N_B respectively) is a set of neighbors of v which can reach some vertex of A (B) but no vertex of B (A). Therefore, v is the vertex desired.

To obtain T' from T , we can delete v and add two new vertices v_a, v_b with an edge between them. We also connect all vertices in N_A to v_a and all vertices in N_B to v_b . Then it is easy to see that (v_a, v_b) induces π in T' . \square

Appendix B: Maximum Independent Set in Bipartite Graphs

Given an undirected bipartite graph $G = (V = A \cup B, E)$ with weights on vertices $w : V \rightarrow \mathbb{N}$, the Maximum Independent Set problem tries to find an independent set $I \subseteq V$ that maximizes $w(I)$, where $w(S) = \sum_{v \in S} w(v)$ for any $S \subseteq V$. We propose a weighted variant of the algorithm from [red reference] to solve the problem.

We first turn the graph into a directed flow network $G' = (V \cup \{s, t\}, E')$ where s, t are the newly added source and sink, respectively. To obtain E' , we direct all edges in E from A to B , add an edge from s to each vertex $u \in A$ and add an edge from each vertex $v \in B$ to t . We set the capacities $c : E' \rightarrow \mathbb{N}$ such that $c(e) = \infty$ if $e \in E$, $c(e) = w(u)$ if $e = (s, u)$ and $c(e) = w(v)$ if $e = (v, t)$. Then we claim that any s, t -cut (S, T) in G' has a finite capacity k if and only if $(S \cap A) \cup (T \cap B)$ is an independent set of weight $w(V) - k$ in G .

We first observe that $((S \cap A) \cup (T \cap B)) \cup ((S \cap B) \cup (T \cap A)) = (S \cup T) \cap (A \cup B) = A \cup B = V$.

Suppose $(S \cap A) \cup (T \cap B)$ is an independent set of weight $w(V) - k$ in G . Since $((S \cap A) \cup (T \cap B)) \cup ((S \cap B) \cup (T \cap A)) = V$, we have the weight of $(S \cap B) \cup (T \cap A)$ is $w(V) - (w(V) - k) = k$. Since $(S \cap A) \cup (T \cap B)$ is an independent set, there is no edge from $S \cap A$ to $T \cap B$. There is also no edge from $S \cap B$ to $T \cap A$ since edges in E are directed from A to B . Therefore the cut (S, T) consist of only edges from s to $T \cap A$ and from $S \cap B$ to t . Together the capacities of those edges are exactly the weight of the set $(S \cap B) \cup (T \cap A)$, which is k .

For the other direction of the proof, suppose (S, T) is an s, t -cut of finite capacity k . Since the cut has finite capacity, it does not contain any edge derived from E . In particular, there is no edge from $S \cap A$ to $T \cap B$ in G' , which implies there is no edge between $S \cap A$ and $T \cap B$ in G . Since there is also no edge among $S \cap A$ and $T \cap B$ in G , $(S \cap A) \cup (T \cap B)$ is an independent set. Since the edges in (S, T) solely consist of edges from s to $T \cap A$ and from $S \cap B$ to t , the sum of their capacities is k . Therefore, the weight of the set $(S \cap B) \cup (T \cap A)$ is k and the weight of $(S \cap A) \cup (T \cap B)$ is $w(V) - k$.

Since $w(V)$ is a fixed constant, we conclude that any s, t -cut (S, T) is a minimum cut in G' if and only if $(S \cap A) \cup (T \cap B)$ is an maximum independent set in G . By the standard Max-flow Min-cut theorem, a minimum s, t -cut in a directed graph is equivalent to the maximum s, t -flow. Thus, we can solve the Maximum Independent Set problem on bipartite graphs through a maximum flow algorithm.

Competing interests

The authors declare that they have no competing interests.

Author's contributions

Text for this section ...

Acknowledgements

Text for this section ...

References

1. Warnow T. Computational phylogenetics: an introduction to designing methods for phylogeny estimation. Cambridge University Press; 2017.

Figures

Tables

Additional Files