RESEARCH

Maximum Bipartition Support Supertree

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, 61820 Urbana, US
Full list of author information is available at the end of the article

Abstract

Keywords:

1 Introduction

space holder

2 Terminologies and Problem Statements

In this section, we introduce basic terminologies and give the problem statements of a few related supertree problems.

Throughout the paper, we consider only unrooted trees. For any tree T, let $V(T)$, $E(T)$, and $L(T)$ denote the vertices, edges, and leaves 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 $T - e$. 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 $\pi_1, \pi_2 \in C(T)$. We include a theorem and a corollary in the appendix that give alternative characterizations of compatibility.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. For every edge $e \in E(T|_R)$, let $P(e)$ denote the path in T from which e is obtained by suppressing degree-two nodes. 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 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 $\mathrm{RF}(T, T') := |C(T) \backslash C(T')| + |C(T') \backslash C(T)|$.

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

Let $T_1, T_2, \ldots, T_N$ be a set of N fully resolved trees with leaf sets $S_1, S_2, \ldots, S_N$, respectively, such that $X := \bigcap_{i\in[N]} S_i \neq \emptyset$ and $S_i \cap S_j = X$ for any $i, j \in [N]$. 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

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

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

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

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

3 Theoretical Results

In this section, we present our theoretical results on a few related supertree problems. For brevity, we only give a proof sketch for most of the lemmas and the full proofs are included in the appendix. We first state a theorem that shows that RF-Supertree-2 is equivalent to Bisup-Supertree-2.

Theorem 1. Given the same input, any tree $T \in \mathcal{T}_S$ is an optimal solution for RF-Supertree-N if and only it is an optimal solution for Bisup-Supertree-N.

Let T_1, T_2, S_1, S_2, and X be defined as from the problem statement. Next we present out main result.

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

We set up some notations for the algorithm and the analysis. In the following part, we always use $i \in [2]$ to index the two trees. Let $T_i|_X$ be the backbone tree of T_i. Let Π be the set of bipartitions of X. Let Triv and NonTriv denote the set of trivial and non-trivial bipartitions in $C(T_1|_X) \cup C(T_2|_X)$. For any $e \in E(T_i|_X)$, let $P(e)$ denote the path in T_i from which e is obtained by suppressing all degree-two nodes. Let $w(e) := |P(e)|$. For any bipartition π of X, let $e_i(\pi)$ denote the edge that induces π in $T_i|_X$. We define a weight function $w : \Pi \to \mathbb{N}_{\geq 0}$ such that for any bipartition π of X, $w(\pi) = w(e_1(\pi)) + w(e_2(\pi))$. If $e_i(\pi)$ does not exist, 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 $e \in E(T_i|_X)$, let $\mathrm{In}(e)$ be the set of internal nodes of $P(e)$. For each $v \in \mathrm{In}(e)$, let $L(v)$ be the set of leaves in $S_i \backslash X$ whose connecting path to $T_i|_X$ 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 $\mathcal{T}(e) := \{T(v) \mid v \in \mathrm{In}(e)\}$. Then $\mathcal{T}(e)$ is the set of extra subtrees attached to e, i.e., the root of each tree is connected to some internal node of $P(e)$ in T_i. We note that $|\mathcal{T}(e)| = |\mathrm{In}(e)| = w(e) - 1$. For any bipartition $\pi \in C(T_1|_X) \cup C(T_2|_X)$, we denote $\mathcal{T}(\pi) := \bigcup_{i \in [2]} \mathcal{T}(e_i(\pi))$. Let $\mathrm{Extra}(T_i) := \bigcup_{e \in E(T_i|_X)} \mathcal{T}(e)$. Then $\mathrm{Extra} := \mathrm{Extra}(T_1) \cup \mathrm{Extra}(T_2)$ denotes the set of all extra subtrees in T_1 and T_2. figure to help Let $G = (V = V_1 \cup V_2, E)$ be the incompatibility graph on $C(T_1|_X) \cup C(T_2|_X)$, such that $V_1 = C(T_1|_X)$ and $V_2 = C(T_2|_X)$, and $E = \{(\pi, \pi') \mid \pi \in V_1, \pi' \in V_2, \pi \text{ is not compatible with } \pi'\}$. We note that G is a bipartite graph since $C(T_i|_X)$ is a compatible set for both $i = 1, 2$ and thus V_i is independent for both $i = 1, 2$.

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 of the backbone trees without degree-two nodes suppressed 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 $p_X(\cdot)$ and $p_Y(\cdot)$ be scores on trees of leaf set S such that

$$p_X(T) = \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_X|, \quad p_Y(T) = \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_Y|.$$

We see that $p_X(T)$ and $p_Y(T)$ decomposes the support score of T into the score contributed by bipartitions in Π_X and the score contributed by bipartitions in Π_Y and the two scores can be maximized sequentially without interference. This is exactly what Algorithm 1 does in three stages. It first construct a tree T that maximizes the $p_Y(\cdot)$ score. Then it builds an incompatibility graph G on $C(T_1|_X) \cup C(T_2|_X)$ and finds a maximum weight independent set I of G that represents a set of compatible bipartitions of $C(T_1|_X) \cup C(T_2|_X)$ whose addition to $T|_X$ gives the maximum potential $p_X(\cdot)$ score. In the end, the algorithm refines T by adding the bipartitions of I to $T|_X$ and moves the extra subtrees around carefully to realize the maximum potential $p_X(\cdot)$ score for T. We note that the order in which the algorithm adds the bipartitions in I does not matter, but for technical reasons, the algorithm actually adds all trivial bipartitions first and then the non-trivial ones.

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
compute $C(T_1|_X)$ and $C(T_2|_X)$
for each $\pi \in C(T_1|_X) \cup C(T_2|_X)$ **do**
    compute $\mathcal{T}(e_1(\pi))$, $\mathcal{T}(e_2(\pi))$, $\mathcal{T}(\pi)$ and $w(\pi)$
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 $\hat{v}$ ▷ let $\hat{T} = T$
for each $\pi = \{a\}|B \in \text{Triv}$ **do**
    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 ▷ let $\tilde{T} = T$ after for loop
construct the incompatibility graph G of $T_1|_X$ and $T_2|_X$
compute the maximum weight independent set I' in $G - (C(T_1|_X) \cap C(T_2|_X)$ with weight w
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 $\pi \in \text{NonTriv}$
for each $\pi \in \text{NonTriv} \cap I$ **do**
    $T \leftarrow \text{Refine}(T_1, T_2, T, \pi, H, sv, \mathcal{T})$ ▷ let $T^* = T$ after for loop
refine T arbitrarily at polytomies until it is fully resolved
return T

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

Lemma 1 formally states that the tree we build at the beginning of the algorithm maximizes the $p_Y(\cdot)$ score. Intuitively, this lemma is true because there are only $|\Pi_Y|$ bipartitions that can contribute to $p_Y(\cdot)$, each of which is either induced by an edge in an extra subtree or induced by an edge connecting an extra subtree to the backbone tree and $\hat{T}$ contains all of them.

Lemma 2. 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 $\pi' \in C(T'|_{S_i}) \backslash C(T|_{S_i})$ for some $i \in [2]$, $\pi'|_X = \pi$. Then, $p_X(T') - p_X(T) \leq w(\pi)$.

Lemma 3. 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 $p_X(T) \leq \sum_{\pi \in F} w(\pi)$.

Lemma 2 shows that the weight function we define on each bipartition represents the maximum potential increase in $p_X(\cdot)$ as a result of adding any one bipartition of X to $T|_X$. The proof of Lemma 2 follows the idea that for any bipartition π of X,

Algorithm 2 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 $C(T|_X) - C(T'|_X) = \pi$

$v \leftarrow sv(\pi)$
compute $N_A := \{u \in N_{T|_X}(v) \mid \exists a \in A$ such that u can reach a in $T|_X - v\}$ and $N_B := \{u \in N_{T|_X}(v) \mid \exists b \in B$ such that u can reach b in $T|_X - v\}$.
$V(T) \leftarrow V(T) \cup \{v_a, v_b\}$, $E(T) \leftarrow E(T) \cup \{(v_a, v_b)\}$
$H(v_a) \leftarrow \emptyset, H(v_b) \leftarrow \emptyset$
for each $u \in N_A \cup N_B$ **do**
    if $u \in N_A$ **then** connect u to v_a
    else connect u to v_b
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
for each bipartition $\pi' = A'|B' \in H(v)$ such that $\pi' \neq \pi$ **do**
    detach all extra subtrees in $\mathcal{T}(\pi')$ from v
    if $A' \subseteq A$ or $B' \subseteq A$ **then**
        $sv(\pi') = v_a$, $H(v_a) \leftarrow H(v_a) + \pi'$, and attach all extra subtrees in $\mathcal{T}(\pi')$ to v_a
    else if $A' \subseteq B$ or $B' \subseteq B$ **then**
        $sv(\pi') = v_b$, $H(v_b) \leftarrow H(v_b) + \pi'$, and attach all extra subtrees in $\mathcal{T}(\pi')$ to v_b
    else
        discard π' and attach all extra subtrees in $\mathcal{T}(\pi')$ to either v_a or v_b
for each remaining extra subtree attached to v **do**
    detach it from v and attach it to either v_a or v_b
delete v and incident edges from T and return the resulting tree T'

there are at most $w(\pi)$ edges in either T_1 or T_2 whose induced bipartition becomes π when it is restricted to X. Therefore, by only adding π to $T|_X$, there are at most $w(\pi)$ more bipartitions in $C(T|_{S_1})$ or $C(T_{S_2})$ that contributes to the increase of $p_X(T)$. The proof of Lemma 3 uses Lemma 2 repeatedly by adding the compatible bipartitions to the tree in an arbitrary order.

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

The above claim states that the algorithm adds all trivial bipartitions of X to $T|_X$ in a way such that $p_X(T)$ actually reaches the full potential of adding those trivial bipartitions. This naturally follows from the way we attach extra subtrees in a way that respect their original ordering in the trees T_1 or T_2.

Lemma 4. 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 2 on T and π, with the auxiliary data structures H, sv, and $\mathcal{T}$. Then, $p_X(T') - p_X(T) = w(\pi)$.

Lemma 4 shows that Algorithm 2 adds any non-trivial bipartition of X to the tree T in a way that realizes the maximum potential increase of $p_X(T)$ of adding that bipartition. The proof mainly relies on three invariants of Algorithm 2. One is that the auxiliary data structures H and sv makes sure that we can correctly find the vertex to split to add any bipartition to $T|_X$. Second is that the extra subtrees of any bipartition π to be added are attached to the vertex whose splitting allows the addition of π. Third is that we always make sure that the extra subtrees of previously added bipartitions of X do not interfere with compatible bipartitions that can still be added to the tree. With these invariants, for any bipartition π to

be added, Algorithm 2 is able to split the vertex correctly and move extra subtrees around in a way that respects the original ordering of the extra subtrees, which makes each bipartition in T_i or T_2 that is induced by an edge in $P(e_1(\pi))$ or in $P(e_2(\pi))$ also present in T_{S_1} or T_{S_2} after the refinement. There are exactly $w(\pi)$ such bipartitions and they contribute $w(\pi)$ to $p_X(T)$.

Claim 2. Let I be defined as in Algorithm 1. I is a maximum weight compatible subset of $C(T_1|_X) \cup C(T_2|_X)$.

This claim follows naturally from the fact that I is a maximum weight independent set in G where any two bipartitions in $C(T_1|_X) \cup C(T_2|_X)$ have an edge between them if and only if they are incompatible.

Next, we restate our main theorem and present the proof using the lemmas and claims we have shown.

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

Proof: First we prove the claim that $p_X(T^*) \geq p_X(T)$ for any tree T of leafset S, where T^* is defined as from Algorithm 1. Let T be any tree of leaf set S. Let $F = C(T|_X)$. Then by Lemma 3, $p_X(T) \leq \sum_{\pi \in F} w(\pi) = \sum_{F \cap (C(T_1|_X) \cup C(T_2|_X))} w(\pi)$. 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 2. We also know by Claim 1 and Lemma 4, $p_X(T^*) = \sum_{\pi \in I'} w(\pi) + \sum_{\pi \in (C(T_1|_X) \cap C(T_2|_X))} w(\pi) = w(I)$. Therefore, $p_X(T) \leq w(I) = p_X(T^*)$.

From Lemma 1, the above claim, and the fact that refinement of a tree never decreases its scores $p_X(\cdot)$ and $p_Y(\cdot)$, we know that $p_X(T^*) \geq p_X(T)$ and $p_Y(T^*) \geq p_Y(T)$ for any tree with leaf set S. By definition of support score, any bipartition can only contribute to the support score if it is in $C(T_1) \cup C(T_2)$. Since Π_X and Π_Y is a disjoint decomposition of $C(T_1) \cup C(T_2)$, the support score of T equals $p_X(T) + p_Y(T)$ for any tree T on leaf set S. Therefore, T^* achieves the maximum support score among all trees of leaf set S. Again, since refinement of a tree never decreases its support score, the tree returned by Algorithm 1 also achieves maximum support score and is a fully resolved tree by construction. We conclude that Algorithm 1 solves Bisup-Supertree-2 correctly.

We include the running time analysis of Algorithm 1 in appendix. □

Last, 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, which was shown to be NP-hard, to Gen-Bisup-Supertree-3.

Theorem 3. [add reference] Maximum Weight Independent Set on tripartite graphs is NP-hard.

Theorem 4. Gen-Bisup-Supertree-3 is NP-hard.

We describe the reduction and sketch a high level proof idea. Let $G = (V = V_1 \cup V_2 \cup V_3, E)$ be any tripartite graph and $w : V \to \mathbb{N}_{>0}$ be a positive integral weight

function. We associate four different leaves with each edge of G so that we have $4|E|$ leaves in total. Let the leaf set be L. The idea is that for any edge, we associate two incompatible mini-bipartitions of the four leaves designated for that edge to the two end vertex of that edge. Then we construct an bipartition π of L for each vertex of G by taking separate unions of the two sides of the mini-bipartitions of that vertex from all edges incident to it and adding all other leaves from edges not incident to it to one side of π. This construction ensures that for any two vertices v, v', their associated bipartitions are compatible if and only if there is no edge between them. Then we can construct three trees of leaf set L using the three compatible set of bipartitions associated with the three parts of the vertices as the bipartition set of the trees. For any edge in any tree that induces a bipartition π, we split the edge into a path of length $w(v)$ where $v \in V(G)$ is the vertex associated with π and we attach a new leaf to each internal node of the path. For any independent set I of G, we build a supertree by following Algorithm 1 except that we do not construct the incompatibility graph to find an MWIS but uses the bipartitions associated with I to refine $T|_L$. Then we can show that an independent set I is a maximum weight independent set of G if and only if a supertree built using I is a maximum bipartition support tree with respect to the three trees because we build the three trees such that G would be the incompatibility graph we would have built.

slightly more discussion on what we don't know and open question We note that we do not know the complexity of Bisup-Supertree-N for any $N \geq 2$.

4 Experiments and Results

space holder

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

Appendix A: General Theorems and Lemmas on Trees and Bipartitions

The following theorem and corollary gives alternative characterizations of compatibility between two bipartitions.

Theorem 5 (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 $A'|B'$.

We provide a lemma and a corollary that shows the relationship between bipartitions of a tree restricted to a subset of leaves and bipartitions restricted to a subset of leaves of a tree.

Lemma 5. 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 $e \in P(e')$.

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 exists 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 exists $e'' \in E(T|_R)$ such that $e'' \neq e'$ and $e \in P(e'')$. By the arguement 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 $e \in P(e')$. □

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 a characterization of the vertex that we can split to add a given bipartition into a tree in the following lemma.

Lemma 6. lemVertexToSplit 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 (and N_B respectively) is the set of neighbors which can reach some vertices of A (B) but not any vertex of B (A) in $T - v$. We can split such a vertex v to add π to $C(T)$.

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 omponent 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 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'. □

The following lemma follows from simple set algebra and is useful for our proof of Theorem 1.

Lemma 7. For two fully resolved trees $T, T' \in \mathcal{T}_S$, $\mathrm{RF}(T, T') + 2\mathrm{bisup}(T, T') = 4n - 6$ where $n := |S|$.

Proof: By definition we have

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

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

Appendix B: Proofs for Section 3

Theorem 1. Given the same input, any tree $T \in \mathcal{T}_S$ is an optimal solution for RF-Supertree-N if and only it is an optimal solution for Bisup-Supertree-N.

Proof: Applying Lemma 7 to $T|_{S_i}$ and T_i for all $i \in [N]$ and taking the sum of the equations, we have

$$\sum_{i\in[N]} (\mathrm{RF}(T|_{S_i}, T_i) + 2\mathrm{bisup}(T|_{S_i}, T_i)) = \sum_{i\in[N]} (4|S_i| - 6)$$
$$\sum_{i\in[N]} \mathrm{RF}(T|_{S_i}, T_i) + 2\sum_{i\in[N]} \mathrm{bisup}(T|_{S_i}, T_i) = \sum_{i\in[N]} (4|S_i| - 6)$$

Since the RF score of T and twice the support score of T sums to $\sum_{i\in[N]}(4|S_i|-6)$, 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. □

Lemma 1. For any tree T of leaf set S, $p_Y(T) \leq |\Pi_Y|$. In particular, let $\hat{T}$ be the tree defined 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 $A \subseteq S_1 \backslash X$, which implies either e_1 is an internal edge in an extra subtree in $\mathrm{Extra}(T_1)$, or e_1 connects one extra subtree in $\mathrm{Extra}(T_1)$ 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 $i \in [2]$ and thus it contributes 1 to $p_Y(\hat{T})$. Hence, $p_Y(\hat{T}) = |\Pi_Y|$. □

Lemma 2. 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 $\pi' \in C(T'|_{S_i}) \backslash C(T|_{S_i})$ for some $i \in [2]$, $\pi'|_X = \pi$. Then, $p_X(T') - p_X(T) \leq w(\pi)$.

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

$$\begin{aligned} p_X(T') - p_X(T) &= \sum_{i\in[2]} |C(T'|_{S_i}) \cap C(T_i) \cap \Pi_X| - \sum_{i\in[2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_X| \\ &= \sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X|. \end{aligned}$$

Therefore, we only need to prove that $\sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \leq w(\pi)$.

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 $\pi \notin C(T_1|_X) \cup C(T_2|_X)$. We recall that $w(\pi) = 0$. Assume for contradiction that there exists a bipartition $\pi' \in (C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for some $i \in [2]$. 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 $i \in [2]$. This contradicts with the fact that $\pi' \in C(T_i)$ for some $i \in [2]$. Therefore, the assumption that there exists such a bipartition π' is wrong and $\sum_{i\in[2]} |(C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| = 0 \leq w(\pi)$.

Case 2): Let $\pi \in C(T_1|_X)\Delta C(T_2|_X)$. Assume without loss of generality that $\pi \in C(T_1|_X)\backslash C(T_2|_X)$. Then, we have $w(\pi) = w(e_1)$. Let $\pi' \in (C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for some $i \in [2]$. Since $\pi'|_X = \pi$ and $\pi \notin C(T_2|_X)$, by Corollary 2, we have $\pi' \notin C(T_2)$ and thus $\pi' \in C(T_1)$. By Lemma 5, the edge which induces π' in T_1 is an edge on $P(e_1)$. Since there are $w(e_1)$ edges on $P(e_1)$, 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 $\pi' \in (C(T'|_{S_1})\backslash C(T|_{S_1})) \cap C(T_1) \cap \Pi_X$. Since $\pi' \in C(T_1)$ and $\pi'|_X = \pi \in C(T_1|_X)$, by Lemma 5, the edge e' that induces π' is an edge on $P(e_1)$. Recall that $w(e_1) = |P(e_1)|$, then we have $|(C(T'|_{S_1})\backslash C(T|_{S_1})) \cap C(T_1) \cap \Pi_X| \leq |P(e_1)| = w(e_1)$. Similarly, $|(C(T'|_{S_2})\backslash C(T|_{S_2})) \cap C(T_2) \cap \Pi_X| \leq |P(e_2)| = w(e_2)$. Therefore, $\sum_{i\in[2]} |(C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \leq w(\pi)$. □

Lemma 3. 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 $\pi_1, \pi_2, \ldots, \pi_k$, where $k = |F|$. Let $F_i = \{\pi_1, \ldots, \pi_i\}$ for any $i \in \{0, 1, \ldots, k\}$. 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 $i \in \{1, 2, \ldots, k\}$ such that for any $\pi' \in C(T^i)\backslash C(T^{i-1})$, $\pi'|_X = \pi_i$. Then by Lemma 2, $p_X(T^i) - p_X(T^{i-1}) \leq w(\pi_i)$. Therefore,

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

Since $C(T^0|_X) = \emptyset$, by Corollary 2, $C(T^0|_{S_i}) = \emptyset$ for both $i \in [2]$. Then, $C(T^0|_{S_i}) \cap C(T_i) \cap \Pi_X = \emptyset$ for both $i \in [2]$, which implies $p_X(T^0) = 0$. Thus, $p_X(T) \leq \sum_{\pi_i\in F} w(\pi_i)$ as desired. □

Let $\hat{T}$ be the tree defined in Algorithm 1. We have the following claim about $p_X(\hat{T})$, which will needed for the proof of Claim 1.

Claim 3. $p_X(\hat{T}) = 2|X|$.

Proof: For each $v \in X$, consider the bipartition $\pi_v = \{v\} \mid S\backslash\{v\}$ of $\hat{T}$ induced by the edge that connects the leaf v to the center $\hat{v}$. It is easy to see that $\pi_v|_{S_i} = \{v\} \mid S_i\backslash\{v\} \in C(T_i)$ for any $i \in [2]$ 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 $i \in [2]$. 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 $i \in [2]$. 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 $A \subseteq S\backslash X$ or $B \subseteq S\backslash X$, which implies $\pi|_{S_i} \notin \Pi_X$ for any $i \in [2]$. Hence, there

is no other bipartition of $\hat{T}$ such that when restrict to S_i contributes to $p_X(\hat{T})$. Therefore, $p_X(\hat{T}) = 2|X|$. □

Claim 1. 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 $t_1, t_2, \ldots, t_p$ 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 $t'_1, t'_2, \ldots, t'_q$ 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

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

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

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

It follows by definition that π_k for any $k \in [w(e_1)]$ is the bipartition induced by the kth 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 1 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 line 6 of 1. 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, $|C(T|_{S_2}) \cap C(T_2|_X) \cap \Pi_X|$ 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 one execution of line 6 in Algorithm 1 on T and π. It is easy to see that line 6 of Algorithm 1 never destroys the bipartitions of S_1 or S_2 already in T, we have $p_X(\tilde{T}) = p_X(\hat{T}) + \sum_{\pi \in \text{Triv}}(w(\pi) - 2) = 2|X| + \sum_{\pi \in \text{Triv}}(w(\pi) - 2) = \sum_{\pi \in \text{Triv}} w(\pi)$. The second last equation follows from Claim 3 that $p_X(\hat{T}) = 2|X|$. □

Lemma 8 proves that the auxiliary data structures of Algorithm 1 are maintaining the desired information, that the algorithm can perform the detaching and reattaching of the extra subtrees correctly, and that the moving around of extra subtrees due to newly added bipartitions to $T|_X$ never destroys biparitions of S_1 or S_2 that is already present in the tree. These invariants are important to the proof of Lemma 4.

Lemma 8. 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 $\pi \in$ NonTriv, $sv(\pi)$ is the vertex to split to add π to $C(T|_X)$. For any internal vertex v, the set of bipartitions $H(v) \subseteq$ 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 $C(T|_X) =$ Triv and thus $T|_X$ is a star with leaf set X. Thus all bipartitions in NonTriv are compatible with T. For any $\pi \in$ 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 $\pi \in$ NonTriv are all connected to $\hat{v}$, so invariant 2 also holds. For any $\pi \in C(T|_X) =$ 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 kth refinement step. We will show that after the kth refinement step, i.e., one execution of Algorithm 2, the invariants still hold for the resulting tree T'. Since $sv(\pi) = \pi$ at the beginning of Algorithm 2, 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 respcetively) 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 2 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 2 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 $N_A \backslash N_{A'} = N_{B'} \backslash N_B$. Since all vertices in N_B are connected to v_b in T' while vertices in $N_{B'} \backslash N_B$ are connected to v_a, $N_{B'} \backslash N_B \cup \{v_b\}$

is the set of all neighbors of v_a which can reach leaves of B' in $T'|_X - v_a$. Then $N_{T'|_X}(v_a) = N_A \cup \{v_b\} = N_{A'} \cup (N_A \backslash N_{A'} \cup \{v_b\}) = N_{A'} \cup (N_{B'} \backslash N_B \cup v_b)$ implies that $N_{A'}$ and $N_{B'} \backslash N_B \cup \{v_b\}$ 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 $N_{B'} \backslash N_B \cup \{v_b\}$ 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 kth 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 2 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 2, 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 spliting 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. □

Lemma 4. 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 2 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 8, $v = sv(\pi)$ is the vertex to split to add π to $T|_X$ and thus the Algorithm 2 correctly splits v into v_a and v_b and connects them to appriopriate 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 $t_1, t_2, \ldots, t_p$ 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 $t'_1, t'_2, \ldots, t'_q$ 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 $i \in [2]$.

For each $k \in [w(e_1)]$, we define $A_k := \bigcup_{i=1}^{k-1} L(t_i) \cup L(\mathcal{T}_1(A)) \cup A$, $\pi_k := A_k|S_1 \backslash A_k$, and for each $k \in [w(e_2)]$, we define $A'_k := \bigcup_{i=1}^{k-1} L(t'_i) \cup L(\mathcal{T}_2(A)) \cup A$, $\pi'_k := A'_k|S_2 \backslash A'_k$. We know that for each $k \in [w(e_1)]$, $S_1 \backslash A_k = \bigcup_{i=k}^{p} L(t_i) \cup L(\mathcal{T}_1(B)) \cup B$. Thus, for any $k \in [w(e_1)]$, π_k is the bipartition induced by the kth 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 $(S_1 \backslash A_k) \cap X = B \neq \emptyset$, 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 8, 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 2 and thus the algorithm attaches them all onto (v_a, v_b) in the order of $t_1, t_2, \ldots, t_p$, 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 $v_1, v_2, \ldots, v_p$ 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 8, 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 $A_k|S_1 \backslash A_k$ in $T'|_{S_1}$. Hence, $\pi_k \in C(T'|_{S_1})$ as desired. □

Claim 2. Let I be defined as in Algorithm 1. I is a maximum weight compatible subset of $C(T_1|_X) \cup C(T_2|_X)$.

Proof: Let G, I' be defined as in Algorithm 1. 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 I' is a maximum weight independent set in $G - C(T_1|_X) \cap C(T_2|_X)$, it is easy to see that I is a maximum weight independent set in G. Since G is the incompatibility graph on $V = C(T_1|_X) \cup C(T_2|_X)$ where there is no edge between any two bipartitions if and only if they are compatible, I corresponds to a maximum weight compatible subset of $C(T_1|_X) \cup C(T_2|_X)$. □

We give the running time analysis for Algorithm 1 to complete the proof of Theorem 2.

Proof: First we analyze the running time of Algorithm 2. 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 16 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 17 can again take $O(n)$ time. The rest of the algorithm takes constant time. Overall, Algorithm 2 runs in $O(n + |X|^2)$ 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 biparition, 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 alway 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 assemling the set of edges asscociated 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 $5 - 6$ essentially runs line 8 of 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 biparitions can be checked in $O(|X|)$ time. For line 8, we can reduce Maximum Weight Independent Set to Minimum Cut problem in a directed graph $D = (\tilde{V} = V \cup \{s, t\}, \tilde{E})$ with dummy source and sink. Then the Minimum Cut problem can be solved by a standard Maximum Flow Algorithm. Since the best Maximum Flow algorithm runs in $O(|\tilde{V}||\tilde{E}|)$ time and $|\tilde{V}| = O(|X|)$ and $|\tilde{E}| = O(|X|^2)$, this line runs in $O(|X|^3)$ time. Line 9 runs in $O(|X|)$ time. Line $10 - 11$ runs Algorithm 2 $O(|X|)$ 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 donimated by $O(|n|^2|X|)$. □

Theorem 4. Gen-Bisup-Supertree-3 is NP-hard.

Proof: Let $G = (V = V_1 \cup V_2 \cup V_3, E)$ be any tripartite graph and $w : V \to \mathbb{N}_{>0}$ be a positive integral weight function. We make a few assumptions about G without loss of generality. First, we can assume that there is no isolated vertex in G as otherwise we can obtain G' by removing the set Z of isolated vertices and any set I is a maximum weight independent set for G' if and only if $I \cup Z$ is a maximum weight independent set for G. We may also assume that there are no two vertices $v \in V_i$ and $v' \in V_j$ for $i, j \in [3]$ and $i \neq j$ such that v and v' are each other's only neighbor because otherwise we can delete all such pairs and obtain a graph G'. Let Z be the set constructed by taking the vertex with larger weight from each of such pairs. Then any set I is a maximum weight independent set for G' if and only if $I \cup Z$ is a maximum weight independent set for G. Lastly, we can assume that there are no pair of vertices $v, v' \in V_i$ for any $i \in [3]$ such that $\delta(v) = \delta(v')$ because otherwise we can obtain G' by merging v and v' into one vertex with the sum of the weights repeatedly until no such pair exists and G and G' would have the same maximum independent set. By our assumption, we know that for any pair of vertices $v, v' \in V$, $\delta(v) \neq \delta(v')$.

We order the edges of G arbitrarily as $e_1, e_2, \ldots, e_m$, where $m := |E|$. Let $n := \max\{|V_1|, |V_2|, |V_3|\}$. We create a leaf set of size $4m$ by having four distinct leaves for each edge of G. Let the leaves associated with e_i be $L_i := \{\ell_i^1, \ell_i^2, \ell_i^3, \ell_i^4\}$ and let $L := \bigcup_i L_i$. For any edge $e_i = (u, w)$, let $A_i(u) := \{\ell_i^1, \ell_i^2\}$, $B_i(u) := \{\ell_i^3, \ell_i^4\}$, $A_i(w) := \{\ell_i^1, \ell_i^3\}$ and $B_i(w) := \{\ell_i^2, \ell_i^4\}$. We notice that $A_i(u)|B_i(u)$ and $A_i(w)|B_i(w)$ are both bipartitions of L_i and they are not compatible.For each $v \in V$, let $A(v) = \bigcup_{e_i \in \delta(v)} A_i(v)$ and $B(v) = \bigcup_{e_i \in \delta(v)} B_i(v)$ and $Q(v) = \bigcup_{e_i \in E \setminus \delta(v)} L_i$. Let $\pi(v) = A(v)|B(v) \cup C(v)$. Since $A(v) \cup B(v) \cup C(v) = \bigcup_{e_i \in \delta(v)} A_i(v) \cup B_i(v) + \bigcup_{e_i \in E \setminus \delta(v)} L_i = \sum_{e_i \in E} L_i = L$, $\pi(v)$ is a bipartition of L for any $v \in V$. By our assumption, for any $v, v' \in V$, at least one of $\delta(v) \setminus \delta(v')$ and $\delta(v') \setminus \delta(v)$ is nontempty. WLOG, we can assume that there exists $e_k \in \delta(v) \setminus \delta(v')$ for some $k \in [m]$. It follows that $\pi(v) \neq \pi(v')$ since $L_k \subset A(v')$ while $L_k \nsubseteq A(v)$. Therefore, $\pi(v) \neq \pi(v')$ for any $v, v' \in V$. Let $\Pi_i := \{\pi(v) \mid v \in V_i\}$ and let $\Pi = \Pi_1 \cup \Pi_2 \cup \Pi_3$. Then there is a bijection between Π_i and V_i for each $i \in [3]$. Since Π_i's are disjoint and V_i's are disjoint, there is also a bijection between V and Π.

We claim that for all $v, v' \in V$, $\pi(v)$ is compatible with $\pi(v')$ if and only if $(v, v') \notin E$. Assume the claim is true, then for each $i \in [3]$, $\pi(v)$ are compatible with each other for all $v \in V_i$. Therefore, there exist trees T_1, T_2, and T_3 on leaf set L such that $C(T_i) = \Pi_i\}$ for all $i \in [3]$. We can construct each T_i in polynomial time by refining a star of leafset L with bipartitions in Π_i one by one. Then for any edge $e \in E(T_i)$ that induces π in T_i for some $i \in [3]$, let $v \in V_i$ be the vertex such that $\pi(v) = \pi$. We insert $w(v) - 1$ new vertices onto edge e, each of which is connected to a new vertex with a label that is different from all existing vertices in all three trees. Then, we have $L(T_1) \cap L(T_2) \cap L(T_3) = L$ and $C(T_i|_L) = \Pi_i$ for all $i \in [3]$.

Let I be any independent set of G. Let $w(I) = \sum_{v \in I} w(v)$. Let $\Pi(I) := \{\pi(v) \mid v \in I\}$, then $\Pi(I)$ is a compatible set of bipartitions by the claim. Therefore, then we can construct a tree T of leaf set $L(T_1) \cup L(T_2) \cup L(T_3)$ such that $C(T|_L) = \Pi(I)$ by refining a star of leafset L with bipartitions in $\Pi(I)$ one by one and then inserting

attaching all extra leaves of T_i onto the edge that induce the same bipartition of L as the edge they were attached to in $T_i|_L$ in the order they apper in T_i. We will show that I is an maximum weight independent set of G if and only if the constructed tree T has maximum score with respect to T_1, T_2, and T_3.

Suppose I is a maximum weight independent set of G. In the construction, for each $\pi \in \Pi_i \cap C(T|_L)$, let v be such that $\pi(v) = \pi$. Then there are $w(v)$ bipartitions in $C(T|_{L(T_i)})$ for some $i \in [3]$, induced by the $w(v)$ edges created by attaching $w(v)-1$ leaves onto the edge that induces π in $T|_L$, that become π when restricted to the set L. These $w(v)$ bipartitions are also present in $C(T_i)$. Therefore, $|C|_{L(T_i)} \cap C(T_i)| = \sum_{v:\pi(v)\in C(T|_L)\cap\Pi_i} w(v)$. Then, the support score of T is

$$\begin{aligned}
\sum_{i\in[3]} \text{bisup}(T|_{L(T_i)}, T_i) &= \sum_{i\in[3]} |C(T|_{L(T_i)}) \cap C(T_i)| \\
&\geq \sum_{i\in[3]} \sum_{v:\pi(v)\in C(T|_L)\cap\Pi_i} w(v) \\
&= \sum_{i\in[3]} \sum_{v:\pi(v)\in \Pi(I)\cap\Pi_i} w(v) \\
&= \sum_{i\in[3]} \sum_{v\in I\cap V_i} w(v) \\
&= w(I)
\end{aligned}$$

Let T' be a tree with maximum support score with respect to T_1, T_2, and T_3. Consider the minimally resolved such T'. We know that $T|_L$ does not contain any bipartition that is not in $C(T_1|_L) \cup C(T_2|_L) \cup C(T_3|_L) = \Pi_1 \cup \Pi_2 \cup \Pi_3 = \Pi$ as otherwise we can let π be such a bipartition and contract the edge that induces π to obtain a less resolved tree with the same support score, which contradicts with the fact that T' is minimally resolved. Therefore, $C(T'|_L) \subseteq \Pi$. Let $I' = \{v \mid \pi(v) \in C(T'|_L)\}$. Since there is a bijection between V and Π, I' is well-defined and $I' \subseteq V$. Since Π' is a compatible set of bipartitions, I' is an independent set in G. For each $v \in I' \cap V_i$, there are at most $w(v)$ bipartitions in $C(T'|_{L(T_i)}) \cap C(T_i)$ that becomes $\pi(v)$ when restricted to the set L as there are only $w(v)$ such bipartitions in $C(T_i)$. For any $\pi \in C(T'|_{L(T_i)}) \cap C(T_i)$, we also have $\pi|_L \in C(T'|_L) \cap C(T_i|_L) = \Pi' \cap \Pi_i$, so there exists a vertex $v \in I' \cap V_i$ such that $\pi(v) = \pi$. Therefore, support score of T' is

$$\begin{aligned}
\sum_{i\in[3]} \text{bisup}(T'|_{L(T_i)}, T_i) &= \sum_{i\in[3]} |C(T'|_{L(T_i)}) \cap C(T_i)| \\
&\leq \sum_{i\in[3]} \sum_{v\in I'\cap V_i} w(v) \\
&= w(I') \leq w(I)
\end{aligned}$$

The last inequality follows by that I is maximum weight independent set of G. Therefore, T as defined has support score no less than T' and thus must has maximum support score.

On the other hand, if I is not a maximum weight independent set in G, then the tree T constructed does not have maximum support score because there exists another independent set I' such that $w(I'') > w(I)$. We can similarly construct T''

from bipartitions associated with I'' and then inserting extra leaeves. Then support score of T'' which is equavalent to $w(I'')$ will be larger than support score of T. This proves the correctness of our reduction. Next we only need to prove the claim that for all $v, v' \in V$, $\pi(v)$ is compatible with $\pi(v')$ if and only if $(v, v') \notin E$.

If $(v, v') \in E$, let (v, v') be e_k. Then there are two symmetrical cases: either $\{\ell_k^1, \ell_k^2\} \subseteq A(v)$ and $\{\ell_k^1, \ell_k^3\} \subseteq A(v')$ or $\{\ell_k^1, \ell_k^3\} \subseteq A(v)$ and $\{\ell_k^1, \ell_k^2\} \subseteq A(v')$. Assume without loss of generality that it is the former case. Then we have $\{\ell_k^3, \ell_k^4\} \subseteq B(v)$ and $\{\ell_k^2, \ell_k^4\} \subseteq B(v')$. Therefore, $A(v) \cap A(v')\{\ell_k^1\}$, $A(v) \cap B(v') = \{\ell_k^2\}$, $B(v) \cap A(v') = \{\ell_k^3\}$, and $B(v) \cap B(v') = \{\ell_k^4\}$. By Theorem 5, $\pi(v)$ and $\pi(v')$ are incompatible.

Suppose $(v, v') \notin E$, then $\delta(v) \cap \delta(v') = \emptyset$. We know that $A(v) \subseteq \bigcup_{e_i \in \delta(v)} L_i$ and $A(v') \subseteq \bigcup_{e_i \in \delta(v')} L_i$. Since $\bigcup_{e_i \in \delta(v)} L_i \cap \bigcup_{e_i \in \delta(v')} L_i = \bigcup_{e_i \in \delta(v) \cap \delta(v')} L_i = \emptyset$, $A(v) \cap A(v') = \emptyset$. Then by Theorem 5, $\pi(v)$ and $\pi(v')$ are compatible. □

Appendix C: Maximum Independent Set in Bipartite Graphs

Given an undirected bipartite graph $G = (V = A \cup B, E)$ with weights on vertices $w : V \to \mathbb{N}$, the Maximum Independent Set problem tries to find a 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 [need 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 $c : E' \to \mathbb{N}$ 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.