A fast tool for minimum hybridization networks

Chen, Zhi-Zhong; Wang, Lusheng; Yamanaka, Satoshi

Published in:
BMC Bioinformatics

Published: 02/07/2012

Document Version:
Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1186/1471-2105-13-155

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 28/08/2019
A fast tool for minimum hybridization networks

Zhi-Zhong Chen1*, Lusheng Wang2 and Satoshi Yamanaka1

Abstract

Background: Due to hybridization events in evolution, studying two different genes of a set of species may yield two related but different phylogenetic trees for the set of species. In this case, we want to combine the two phylogenetic trees into a hybridization network with the fewest hybridization events. This leads to three computational problems, namely, the problem of computing the minimum size of a hybridization network, the problem of constructing one minimum hybridization network, and the problem of enumerating a representative set of minimum hybridization networks. The previously best software tools for these problems (namely, Chen and Wang's HybridNet and Albrecht et al.'s Dendroscope 3) run very slowly for large instances that cannot be reduced to relatively small instances. Indeed, when the minimum size of a hybridization network of two given trees is larger than 23 and the problem for the trees cannot be reduced to relatively smaller independent subproblems, then HybridNet almost always takes longer than 1 day and Dendroscope 3 often fails to complete. Thus, a faster software tool for the problems is in need.

Results: We develop a software tool in ANSI C, named FastHN, for the following problems: Computing the minimum size of a hybridization network, constructing one minimum hybridization network, and enumerating a representative set of minimum hybridization networks. We obtain FastHN by refining HybridNet with three ideas. The first idea is to preprocess the input trees so that the trees become smaller or the problem becomes to solve two or more relatively smaller independent subproblems. The second idea is to use a fast algorithm for computing the rSPR distance of two given phylogenetic trees to cut more branches of the search tree in the exhaustive-search stage of the algorithm. The third idea is that during the exhaustive-search stage of the algorithm, we find two sibling leaves in one of the two forests (obtained from the given trees by cutting some edges) such that they are as far as possible in the other forest. As the result, FastHN always runs much faster than HybridNet. Unlike Dendroscope 3, FastHN is a single-threaded program. Despite this disadvantage, our experimental data shows that FastHN runs substantially faster than the multi-threaded Dendroscope 3 on a PC with multiple cores. Indeed, FastHN can finish within 16 minutes (on average on a Windows-7 (x64) desktop PC with i7-2600 CPU) even if the minimum size of a hybridization network of two given trees is about 25, the trees each have 100 leaves, and the problem for the input trees cannot be reduced to two or more independent subproblems via cluster reductions. It is also worth mentioning that like HybridNet, FastHN does not use much memory (indeed, the amount of memory is at most quadratic in the input size). In contrast, Dendroscope 3 uses a huge amount of memory. Executables of FastHN for Windows XP (x86), Windows 7 (x64), Linux, and Mac OS are available (see the Results and discussion section for details).

Conclusions: For both biological datasets and simulated datasets, our experimental results show that FastHN runs substantially faster than HybridNet and Dendroscope 3. The superiority of FastHN in speed over the previous tools becomes more significant as the hybridization number becomes larger. In addition, FastHN uses much less memory than Dendroscope 3 and uses the same amount of memory as HybridNet.

*Correspondence: zzchen@mail.dendai.ac.jp
1Division of Information System Design, Tokyo Denki University, Ichizaka, Hatoyama, Hiki, Saitama 359-0394, Japan
Full list of author information is available at the end of the article
Background

Constructing the evolutionary history of a set of species is an important problem in the study of biological evolution. Phylogenetic trees are used in biology to represent the ancestral history of a collection of existing species. This is appropriate for many groups of species. However, there are some groups for which the ancestral history cannot be represented by a tree. This is caused by processes such as hybridization, recombination, and lateral gene transfer. We refer to those processes as reticulation events. For this kind of groups of species, it is more appropriate to represent their ancestral history by rooted acyclic digraphs, where vertices of in-degree at least two represent reticulation events.

When studying the evolutionary history of a set of existing species, one can obtain a phylogenetic tree of the set of species with high confidence by looking at a segment of sequences or a set of genes. When looking at another segment of sequences, a different phylogenetic tree can be obtained with high confidence, too. This indicates that reticulation events may occur. Thus, we have the following problem: Given two rooted phylogenetic trees on a set of species that correctly represent the tree-like evolution of different parts of their genomes, what is the smallest number of reticulation events needed to explain the evolution of the species under consideration?

The subtree prune and regraft (rSPR) distance and the hybridization number are two important measures for evolutionary tree comparison and hybridization network construction. Since both problems are NP-hard [1-3], it is challenging to develop programs that can give exact solutions when the two given trees are large or have a large rSPR distance or hybridization number. Previously, several software packages have been developed for these problems [4-9]. This new breakthrough brings us a hope that one can routinely solve these hard problems for two given large trees. However, the previously fastest software packages can still take hours to finish even when the given trees are of moderate sizes. Thus, a faster software tool for these problems is in need.

In general, there may exist two or more minimum hybridization networks displaying two given phylogenetic trees $T_1$ and $T_2$ with the same leaf set $X$. In some cases, we may want to enumerate all minimum hybridization networks displaying both $T_1$ and $T_2$. Unfortunately, it is not hard to construct two example phylogenetic trees $T_1$ and $T_2$ such that there are too many minimum hybridization networks displaying both $T_1$ and $T_2$. So, we instead want to enumerate only a representative set of minimum hybridization networks displaying both $T_1$ and $T_2$. Here, a hybridization network $N$ represents another hybridization network $N'$ if for every pair $(x, y)$ of species in $X$, $x$ and $y$ fall into the same connected component of $F_N$ if and only if they fall into the same connected component of $F_{N'}$, where $F_N$ (respectively, $F_{N'}$) is the forest obtained from $N$ (respectively, $N'$) by removing all the edges entering reticulate nodes. HybridNet [6] and Dendroscope 3 [4] are able to enumerate a representative set of minimum hybridization networks for two given phylogenetic trees. If the problem for the two given trees can be reduced to relatively smaller independent subproblems (by so-called “cluster reductions”), Dendroscope is much faster than HybridNet; otherwise, the two have almost the same speed. Unfortunately, both tools run very slowly when the minimum hybridization number of a hybridization network of two given trees is large (say, larger than 23) and the problem for the trees cannot be reduced to relatively smaller independent subproblems. Thus, a much faster tool is in need.

Results and discussion

We have developed a new tool (called FastHN) for the problem of enumerating a representative set of minimum hybridization networks of two given phylogenetic trees. Of course, FastHN can also compute the minimum hybridization number of a hybridization network of two given phylogenetic trees and construct a single minimum hybridization network of two given phylogenetic trees. FastHN is implemented in ANSI C and is available at http://rnc.r.dendai.ac.jp/~chen/fastHN.html, or http://www.cs.cityu.edu.hk/~iwang/software/FastHN/fastHN.html, where one can download executables for Windows XP (x86), Windows 7 (x64), Linux, and Mac OS.

After downloading FastHN, one can run it as follows:

FastHN T1 T2 OPTION HEURISTIC or simply FastHN T1 T2 OPTION

Here, T1 and T2 are two text files each containing a phylogenetic tree in the Newick format (ended with a semicolon). The label of each leaf in an input tree should be a string consisting of letters in \{0, 1, \ldots, 9, a, b, \ldots, z, A, B, \ldots, Z, \ldots\}. There is no limit on the length of the label of each leaf.

OPTION is a string in the set \{HN, MAAF, MAAFs\} controlling the output as follows:

- **HN**: The output is the hybridization number of T1 and T2.
- **MAAF**: The output is one MAAF of T1 and T2 together with one minimum hybridization network for the MAAF.
- **MAAFs**: The output is all MAAFs of T1 and T2 together with one minimum hybridization network for each MAAF.

FastHN outputs an MAAF (respectively, MAF) by printing out the leaf sets of the trees in the MAAF (respectively, MAF), while it outputs a hybridization network in its extended Newick format [10]. When OPTION
is MAAFs (respectively, MAFs), *FastHN* outputs the MAAFs (respectively, MAFs) without repetition. We remind the reader that one can view a tree in the Newick format and a network in the extended Newick format by using *Dendroscope* due to [11].

**HEURISTIC** is a 3-bit binary string specifying the version of *FastHN* as follows:

- The first bit is 1 if and only if *FastHN* adopts initial cluster reductions.
- The second bit is 1 if and only if *FastHN* adopts Heuristic 1.
- The last bit is 1 if and only if *FastHN* adopts Heuristic 2.

**HEURISTIC** can be omitted; in that case, it is set to be 111.

To compare the efficiency of *FastHN* with the previous bests (namely, *HybridNet* [6] and *Dendroscope* 3 [4]), we have run them on both simulated datasets and biological datasets for the problem of computing all MAAFs of two given phylogenetic trees. The experiment has been performed on a Windows-7 (x64) desktop PC with i7-2600 CPU and 4GB RAM. It is worth mentioning that *FastHN* is multi-threaded, its total elapsed time can be seen from the figure, *FastHN* with Heuristic 1 and/or Heuristic 2 is much faster than *HybridNet* and *Dendroscope* 3. This difference in speed becomes more significant as the hybridization number becomes larger. Moreover, Heuristic 1 contributes the most to the saving of running time. Indeed, when Heuristic 1 is used, both Heuristic 2 and initial cluster reductions do not help much. It is worth noting that *Beiko* and *Hamilton’s program* tends to create a pair of trees without a relatively large common clusters. This is why initial cluster reductions do not help much for tree-pairs randomly generated by their program.

The comparison is done on 120 randomly generated tree-pairs with relatively small hybridization numbers, where each tree has 100 leaves. If the running time of a program for a tree-pair is more than 600 seconds, then it has been rounded down to 600 seconds. Among the 120 pairs, *Dendroscope* 3 takes more than 600 seconds for 28 pairs, *FastHN* without Heuristic 1, 2, or initial cluster reductions takes more than 600 seconds for 20 pairs, *FastHN* with only initial cluster reductions takes more than 600 seconds for 18 pairs, and *FastHN* with Heuristic 1 or 2 takes less than 60 seconds for every pair.

To test how the number of leaves in an input tree influences the running time of the algorithms, we next use *Beiko* and *Hamilton’s program* to generate 120 pairs of trees each of which has 50 leaves. The first (respectively, second) 60 pairs are generated by setting $r = 14$ (respectively, $r = 17$). It turns out that among the 120 generated tree-pairs, 3 (respectively, 17, 26, 26, 20, 20, or 6) tree-pairs have hybridization number 11 (respectively, 12, 13, 14, 15, 16, or 17). Moreover, for each $h \in \{9, 10\}$, there is exactly one generated tree-pair with hybridization number $h$. Figure 2 summarizes the average running time of the programs for those generated tree-pairs with hybridization number in the range $[12..17]$, where each average is taken over those tree-pairs with the same hybridization number. As can be seen from the figure, the superiority of *FastHN* over *HybridNet* and *Dendroscope* 3 remains the same (as in Figure 1) if Heuristic 1 or 2 is used. Moreover, Heuristic 1 contributes the most to the saving of running time.

The comparison is done on 120 randomly generated tree-pairs with relatively small hybridization numbers, where each tree has 50 leaves.

To compare the performance of the algorithms for tree-pairs with relatively large hybridization numbers, we further use *Beiko* and *Hamilton’s program* to generate 60 pairs of trees by setting $r = 25$, where each tree has 100 leaves. It turns out that among the 60 generated tree-pairs,
4 (respectively, 10, 15, 18, or 12) tree-pairs have hybridization number 21 (respectively, 22, 23, 24, or 25). Moreover, there is exactly one generated tree-pair with hybridization number 20. Figure 3 summarizes the average running time of the two best versions of FastHN for the generated tree-pairs, where each average is taken over those tree-pairs with the same hybridization number. As can be seen from the figure, both versions take less than 16 minutes (on average) even when the hybridization number is as large as 25, while FastHN with both Heuristics 1 and 2
As can be seen from Table 1, for most of the tree-pairs, there is not much difference in speed between Dendroscope 3 and FastHN with initial cluster reductions. This is because most of the tree-pairs have small hybridization numbers. For the tree-pair (ndhf, ITS), FastHN with cluster reductions runs substantially faster than FastHN without cluster reductions. This is because the problem for this pair can be reduced to two tree-pairs of roughly equal sizes by initial cluster reductions in the preprocessing stage of the algorithm.

Discussion

Roughly speaking, FastHN consists of two stages, namely, the preprocessing stage and the exhaustive-search stage. In the preprocessing stage, FastHN performs only subtree reductions and cluster reductions. Indeed, other kinds of reductions are also known. One of them is chain reduction [15]. Performing chain reductions on the input trees results in trees whose nodes are weighted. Unfortunately, it seems that Whidden et al.’s \(O(2.42^d n)\)-time algorithm for computing the rSPR distance \(d\) of two given phylogenetic trees with \(n\) leaves does not work when the trees are weighted. This is why FastHN does not perform chain reductions.

In the exhaustive-search stage, FastHN also performs subtree reductions whenever possible, but does not perform cluster reductions. The main reason of not performing cluster reductions in the 2nd stage is that performing a cluster reduction is too time-consuming (namely, takes \(O(n^2)\) time, where \(n\) is the number of leaves in the trees).

When running FastHN, one can decide whether to adopt initial cluster reductions, Heuristic 1, or Heuristic 2. If two input trees have relatively large common clusters,
performing initial cluster reductions on them lead to solving independent and significantly smaller subproblems. So, we should always choose to adopt initial cluster reductions. Moreover, as can be seen from our simulated results, we should always choose to adopt Heuristic 1 because it enables the algorithm to save a lot of time by cutting more branches of the search tree in the exhaustive-search stage. Our simulated results also show that adopting both Heuristics 1 and 2 makes FastHN run faster (on average) than adopting only Heuristic 1. Thus, in general, we should choose to adopt Heuristic 2 as well. However, in our experiments, we have found some tree-pairs for which FastHN with Heuristic 1 but without Heuristic 2 runs significantly faster than FastHN with both Heuristics 1 and 2. Hence, as long as Heuristic 1 is adopted, there is still room to decide whether to adopt Heuristic 2 as well.

**Conclusions**

Our experiments show that FastHN runs substantially faster than the previously best tools (namely, HybridNet and Dendroscope 3). The fast speed of FastHN originates from two key new ideas (which have not been used to solve the problems before, as far as we know):

- We use a fast algorithm for computing the rSPR distance of two given phylogenetic trees to cut more branches of the search tree during the exhaustive-search stage of FastHN.
- During the exhaustive-search stage of FastHN, we always try to find a pair of sibling leaves in one of the two forests (obtained from the given trees by cutting some edges) such that the two leaves is as far apart as possible in the other forest.

**Methods**

Throughout this section, a **rooted forest** always means a directed acyclic graph in which every node has in-degree at most 1 and out-degree at most 2.

Let $F$ be a rooted forest. The **roots** (respectively, **leaves**) of $F$ are those nodes whose in-degrees (respectively, out-degrees) are 0. The **size** of $F$, denoted by $|F|$, is the number of roots in $F$ minus 1. A node $v$ of $F$ is **unifurcate** if it has only one child in $F$. If a root $v$ of $F$ is unifurcate, then **contracting** $v$ in $F$ is the operation that modifies $F$ by deleting $v$. If a non-root node $v$ of $F$ is unifurcate, then contracting $v$ in $F$ is the operation that modifies $F$ by first adding an edge from the parent of $v$ to the child of $v$ and then deleting $v$.

For convenience, we view each node $u$ of $F$ as an ancestor and descendant of $u$ itself. A node $u$ is **lower than** another node $v \neq u$ in $F$ if $u$ is a descendant of $v$ in $F$. The **lowest common ancestor** (LCA) of a set $U$ of nodes in $F$ is the lowest node $v$ in $F$ such that for every node $u \in U$, $v$ is an ancestor of $u$ in $F$. For a node $v$ of $F$, the **subtree of $F$ rooted at $v$** is the subgraph of $F$ whose nodes are the descendants of $v$ in $F$ and whose edges are those edges connecting two descendants of $v$ in $F$. If $v$ is a root of $F$, then the subtree of $F$ rooted at $v$ is a **component tree** of $F$. $F$ is a **rooted tree** if it has only one root.

A **rooted binary forest** is a rooted forest in which the out-degree of every non-leaf node is 2. Let $F$ be a rooted binary forest. $F$ is a **rooted binary tree** if it has only one root. If $v$
is a non-root node of \( F \) with parent \( p \) and sibling \( u \), then detaching the subtree of \( F \) rooted at \( v \) is the operation that modifies \( F \) by first deleting the edge \((p,v)\) and then contracting \( p \). A detaching operation on \( F \) is the operation of detaching the subtree of \( F \) rooted at a non-root node.

Hybridization networks and phylogenetic trees
Let \( X \) be a set of existing species. A hybridization network on \( X \) is a directed acyclic graph \( N \) in which the set of nodes of out-degree 0 (still called the leaves) is \( X \), each non-leaf node has out-degree 2, there is exactly one node of in-degree 0 (called the root), and each non-root node has in-degree larger than 0. Note that the in-degree of a non-root node in \( N \) may be larger than 1. A node of in-degree larger than 1 in \( N \) is called a reticulation node of \( N \). Intuitively speaking, a reticulation node corresponds to a reticulation event. The hybridization number of a reticulation node in \( N \) is its in-degree in \( N \) minus one. The hybridization number of \( N \) is the total hybridization number of reticulation nodes in \( N \).

A phylogenetic tree on \( X \) is a rooted binary tree whose leaf set is \( X \). A hybridization network \( N \) on \( X \) displays a phylogenetic tree \( T \) on \( X \) if \( N \) has a subgraph \( M \) such that \( M \) is a rooted tree, the root of \( M \) has exactly two children in \( M \), and modifying \( M \) by contracting its unifurcate nodes yields \( T \). A hybridization network of two phylogenetic trees \( T_1 \) and \( T_2 \) on \( X \) is a hybridization network \( N \) on \( X \) such that \( N \) displays both \( T_1 \) and \( T_2 \). A hybridization network of \( T_1 \) and \( T_2 \) is minimum if its hybridization number is minimized among all hybridization networks of \( T_1 \) and \( T_2 \). Obviously, if \( N \) is a minimum hybridization network of \( T_1 \) and \( T_2 \), then the in-degree of every reticulation node in \( N \) is exactly 2 and hence the hybridization number of \( N \) is equal to the number of reticulation nodes in \( N \). For convenience, we define the hybridization number of \( T_1 \) and \( T_2 \) to be the minimum hybridization number of a hybridization network of \( T_1 \) and \( T_2 \).

We are now ready to define one problem studied in this paper:

Hybridization Network Construction (HNC):

Input: Two phylogenetic trees \( T_1 \) and \( T_2 \) on the same set \( X \) of species.
Goal: To construct a minimum hybridization network of \( T_1 \) and \( T_2 \).

Agreement forests
Throughout this subsection, let \( T_1 \) and \( T_2 \) be two phylogenetic trees on the same set \( X \) of species. If we can apply a sequence of detaching operations on each of \( T_1 \) and \( T_2 \) so that they become the same forest \( F \), then we refer to \( F \) as an agreement forest (AF) of \( T_1 \) and \( T_2 \). A maximum agreement forest (MAF) of \( T_1 \) and \( T_2 \) is an agreement forest of \( T_1 \) and \( T_2 \) whose size is minimized over all agreement forests of \( T_1 \) and \( T_2 \). The size of an MAF of \( T_1 \) and \( T_2 \) is called the \( rSPR \) distance between \( T_1 \) and \( T_2 \). The following lemma is shown in [16].

Lemma 1 [16] Given two phylogenetic trees \( T_1 \) and \( T_2 \), we can compute the \( rSPR \) distance between \( T_1 \) and \( T_2 \) in \( O(2.42^d n) \) time, where \( n \) is the number of leaves in \( T_1 \) and \( T_2 \) and \( d \) is the \( rSPR \) distance between \( T_1 \) and \( T_2 \).

Let \( F \) be an agreement forest of \( T_1 \) and \( T_2 \). Obviously, for each \( i \in \{1,2\} \), the leaves of \( T_i \) one-to-one correspond to the leaves of \( F \). For convenience, we hereafter identify each leaf \( v \) of \( F \) with the leaf of \( T_i \) corresponding to \( v \). Similarly, for each \( i \in \{1,2\} \), the non-leaf nodes of \( F \) correspond to distinct non-leaf nodes of \( T_i \). More precisely, a non-root node \( u \) of \( F \) corresponds to the LCA of \( \{v_1,\ldots,v_\ell\} \) in \( T_i \), where \( v_1,\ldots,v_\ell \) are the leaf descendants of \( u \) in \( F \). Again for convenience, we hereafter identify each non-root node \( u \) of \( F \) with the non-leaf node of \( T_i \) corresponding to \( u \). With these correspondences, we can use \( F \), \( T_1 \), and \( T_2 \) to construct a directed graph \( GF \) as follows:

- The nodes of \( GF \) are the roots of \( F \).
- For every two roots \( r_1 \) and \( r_2 \) of \( F \), there is an edge from \( r_1 \) to \( r_2 \) in \( GF \) if and only if \( r_1 \) is an ancestor of \( r_2 \) in \( T_1 \) or \( T_2 \).

We refer to \( GF \) as the decision graph associated with \( F \). If \( GF \) is acyclic, then \( F \) is an acyclic agreement forest (AAF) of \( T_1 \) and \( T_2 \); otherwise, \( F \) is a cyclic agreement forest (CAF) of \( T_1 \) and \( T_2 \). If \( F \) is an AAF of \( T_1 \) and \( T_2 \) and its size is minimized over all AAFs of \( T_1 \) and \( T_2 \), then \( F \) is a maximum acyclic agreement forest (MAAF) of \( T_1 \) and \( T_2 \). Note that our definition of an AAF is the same as those in [15,17] but is different from that in [16]. Moreover, it is known that the size of an MAAF of \( T_1 \) and \( T_2 \) is equal to the hybridization number of \( T_1 \) and \( T_2 \) [18]. The following lemma is shown in [19]:

Lemma 2 [19] Suppose that \( C \) is a cycle of \( GF \) and \( r_1,\ldots,r_\ell \) are the nodes of \( C \). Then, each \( r_j \in \{r_1,\ldots,r_\ell\} \) has two children \( u_j \) and \( u_j' \) in \( F \). Moreover, for every non-root node \( v \) of \( F \) not contained in \( \{u_1,u_1',\ldots,u_\ell,u_\ell'\} \), \( C \) remains a cycle in \( GF \) after \( F \) is modified by detaching the subtree of \( F \) rooted at \( v \).

Let \( N \) be a minimum hybridization network of \( T_1 \) and \( T_2 \). Suppose that we modify \( N \) to obtain a forest \( F(N) \) by first removing all edges entering reticulation nodes, then removing those nodes \( v \) such that neither \( v \) nor its descendants are in \( X \), and further contracting all unifurcate nodes. Obviously, \( F(N) \) is an AAF of \( T_1 \) and \( T_2 \) and the size of \( F(N) \) is exactly the hybridization number of
N. So, each MAAF \( F' \) of \( T_1 \) and \( T_2 \) represents the set of all minimum hybridization networks \( N \) such that \( F(N) \) is the same as \( F' \). Thus, to enumerate a representative set of minimum hybridization networks of \( T_1 \) and \( T_2 \), the idea in previous work [6] has been to enumerate all MAAFs of \( T_1 \) and \( T_2 \) and construct a minimum hybridization network for each enumerated MAAF. Since we can easily use an MAAF of \( T_1 \) and \( T_2 \) to construct a hybridization network displaying \( T_1 \) and \( T_2 \) [6], the difficulty is in how to enumerate all MAAFs of \( T_1 \) and \( T_2 \).

We are now ready to define another problem studied in this paper:

**Hybridization Network Enumeration (HNE):**

**Input:** Two phylogenetic trees \( T_1 \) and \( T_2 \) on the same set \( X \) of species.

**Input:** Two phylogenetic trees \( T_1 \) and \( T_2 \) on the same set \( X \) of species.

**Goal:** To enumerate all MAAFs of \( T_1 \) and \( T_2 \) and construct a minimum hybridization network of \( T_1 \) and \( T_2 \) from each MAAF of \( T_1 \) and \( T_2 \).

Basically, HNE is the problem of enumerating a representative set of minimum hybridization networks of two given phylogenetic trees. As in previous studies [5,6,8], when we consider HNC and HNE, we always assume that each given phylogenetic tree has been modified by first introducing a new root and a dummy leaf and then letting the old root and the dummy leaf be the children of the new root.

The following lemma is shown in [19]:

**Lemma 3** [19] The dummy leaf alone does not form a component tree of an MAAF of \( T_1 \) and \( T_2 \).

**Extending Whidden et al.'s Algorithm**

Throughout this subsection, let \( T_1 \) and \( T_2 \) be two phylogenetic trees on the same set \( X \) of species. We sketch the fastest known algorithm (due to Whidden et al. [16]) for computing an MAF of \( T_1 \) and \( T_2 \), and then state a slight extension of the algorithm that will be used in our algorithm for HNE.

The basic idea behind Whidden et al.'s algorithm is as follows. For \( k = 0, 1, 2, \ldots \) (in this order), we try to find an AF of \( T_1 \) and \( T_2 \) of size \( k \) and stop immediately once such an AF is found. To find an AF of \( T_1 \) and \( T_2 \) of size \( k \), we start by setting \( F_1 = T_1 \) and \( F_2 = T_2 \) and associating a label set \( \{x\} \) to each leaf \( x \) of \( F_1 \) and \( F_2 \). We then repeatedly modify \( F_1 \) and \( F_2 \) (until either \( |F_1| > k \) or \( F_1 \) becomes a forest without edges) as follows. We find two arbitrary sibling leaves \( u \) and \( v \) in \( F_2 \). If \( u \) and \( v \) are also siblings in \( F_1 \), then we modify \( F_1 \) and \( F_2 \) separately by merging the identical subtrees of \( F_1 \) and \( F_2 \) rooted at the parent of \( u \) and \( v \) each into a single leaf whose label set is the union of the label sets of \( u \) and \( v \). On the other hand, if \( u \) and \( v \) are not siblings in \( F_1 \), then we distinguish three cases as follows.

**Case 1:** \( u \) and \( v \) are in different component trees of \( F_1 \).

In this case, in order to transform \( F_1 \) and \( F_2 \) into an AF of \( T_1 \) and \( T_2 \), we have two choices to modify them, namely, by either detaching the subtree rooted at \( u \) or detaching the subtree rooted at \( v \).

**Case 2:** \( u \) and \( v \) are in the same component tree of \( F_1 \) and either (1) \( u \) and the parent of \( v \) are siblings in \( F_1 \) or (2) \( v \) and the parent of \( u \) are siblings in \( F_1 \). In this case, if (1) (respectively, (2)) holds, then we modify \( F_1 \) by detaching the subtree rooted at the sibling of \( v \) (respectively, \( u \)).

**Case 3:** \( u \) and \( v \) are in the same component tree of \( F_1 \) and neither (1) nor (2) in Case 2 holds. In this case, in order to transform \( F_1 \) and \( F_2 \) into an AF of \( T_1 \) and \( T_2 \), we have three choices to modify them. The first two choices are the same as those in Case 1. In the third choice, we modify \( F_1 \) by detaching the subtrees rooted at those non-root nodes \( w \) such that the parent of \( w \) appears on the (not necessarily directed) path between \( u \) and \( v \) in \( F_1 \) but \( w \) does not.

By the above three cases, we always have the following:

(a) \( |F_1| \geq |F_2| \).

(b) All component trees of \( F_2 \) except at most one have no edges.

(c) For each component tree \( \Gamma_2 \) of \( F_2 \) without edges, \( F_1 \) has a component tree \( \Gamma_1 \) without edges such that the label sets associated with the unique leaves of \( \Gamma_1 \) and \( \Gamma_2 \) are identical.

Once \( |F_1| \) becomes larger than \( k \), we know that \( F_1 \) and \( F_2 \) have no AF of size \( k \). On the other hand, once \( F_1 \) becomes a forest without edges, we can use the label sets \( L(v) \) of the leaves \( v \) of \( F_1 \) to obtain an AF of \( T_1 \) and \( T_2 \) of size \( |F_1| \) by modifying \( T_1 \) as follows. For each leaf \( v \) of \( F_1 \) such that \( L(v) \) does not contain the dummy leaf, detach the subtree of \( T_1 \) rooted at the LCA of the leaves in \( L(v) \).

Now, we are now ready to make a key observation in this paper. By (b) and (c) in the above, Whidden et al.’s MAF algorithm can actually be used to solve the following slightly more general problem in \( O(2.42^k n) \) time:

**rSPR Distance Checking (rSPRDC):**

**Input:** \( (T_1, T_2, k, F_1, F_2) \), where \( T_1 \) and \( T_2 \) are two phylogenetic trees on the same set \( X \) of species, \( k \) is an integer, \( F_1 \) (respectively, \( F_2 \)) is a rooted forest obtained from \( T_1 \) (respectively, \( T_2 \)) by performing zero or more detaching operations, and every component tree of \( F_2 \) except at most one is identical to a component tree of \( F_1 \).
**Goal:** To decide if performing $k$ more detaching operations on $F_1$ leads to an AF of $T_1$ and $T_2$.

Finally, if we want to enumerate all MAFs of $T_1$ and $T_2$, then we need to modify Whidden et al.’s algorithm as follows. First, we do not distinguish Cases 2 and 3 because modifying $F_1$ as in Case 2 may lose some MAF of $T_1$ and $T_2$. Moreover, whenever an AF of $T_1$ and $T_2$ of size $k$ is found, we do not stop immediately and instead continue to find other AFs of $T_1$ and $T_2$ of size $k$. The resulting algorithm runs more slowly, namely, in $O(3^k n)$ time.

**Speeding up HybridNet**

Throughout this subsection, let $T_1$ and $T_2$ be two phylogenetic trees on the same set $X$ of species. We first sketch how HybridNet enumerates all MAAFs of $T_1$ and $T_2$, and then explain how to speed it up.

First, we need several definitions. For a rooted forest $F$, we use $\mathcal{L}(F)$ to denote the family of the leaf sets of the component trees of $F$. Let $F$ and $F'$ be two forests each obtained by performing zero or more detaching operations on $T_i$. If $F \neq F'$ and for every set $Y \in \mathcal{L}(F)$, there is a set $Y' \in \mathcal{L}(F')$ with $Y \subseteq Y'$, then we say that $F$ is finer than $F'$ and $F'$ is coarser than $F$.

To enumerate all MAAFs of $T_1$ and $T_2$, the idea behind HybridNet is to design an algorithm for the following problem:

**Generalized Agreement Forest (GAF)**

**Input:** $(T_1, T_2, k, F_1)$, where $T_1$ and $T_2$ are two phylogenetic trees on the same set $X$ of species, $k$ is an integer, and $F_1$ is a rooted forest obtained from $T_1$ by performing zero or more detaching operations.

**Goal:** To find a sequence of AFs of $T_1$ and $T_2$ including all AFs $F$ of $T_1$ and $T_2$ such that (1) $F$ can be obtained by performing at most $k$ detaching operations on $F_1$ (or equivalently, at most $|F_1| + k$ detaching operations on $T_2$) and (2) no AF of $T_1$ and $T_2$ is finer than $F_1$ and coarser than $F$.

In the supplementary material of [19], an $O(3^k n)$-time algorithm for solving GAF is detailed. The algorithm differs from Whidden et al.’s algorithm for enumerating all MAFs of $T_1$ and $T_2$ only in that we start with $F_1$ (as it is given) and $F_2 = T_2$ (instead of starting with $F_1 = T_1$ and $F_2 = T_2$) and then repeatedly modify $F_1$ and $F_2$ until either $|F_1| > k + k_0$ or $F_1$ becomes a forest without edges, where $k_0$ is the original size of $F_1$. Now, we are now ready to make two other key observations in this paper. To speed up Chen and Wang’s algorithm for solving GAF, we modify it as follows:

- **Heuristic 1:** Every time before we start to make multiple choices of modifying $F_1$ and $F_2$, we call the algorithm for rSPRDC in Lemma 1 on input $(T_1, T_2, k - |F_1| + k_0, F_1, F_2)$ to check if performing $k - |F_1| + k_0$ more detaching operations on $F_1$ leads to an AF of $T_1$ and $T_2$.

As the result, if we know that performing $k - |F_1| + k_0$ more detaching operations on $F_1$ does not lead to an AF of $T_1$ and $T_2$, then no more choice of modifying $F_1$ and $F_2$ is necessary; otherwise, we proceed to make multiple choices of modifying $F_1$ and $F_2$ the same as before but with the following difference:

- **Heuristic 2:** Instead of selecting two arbitrary sibling leaves $u$ and $v$ in $F_2$ (cf. the Extending Whidden et al.’s Algorithm section), we select two sibling leaves $u$ and $v$ in $F_2$ such that they are as far apart as possible in $F_1$.

The intuition behind Heuristic 2 is that if $u$ and $v$ are far apart in $F_1$, then either $u$ and $v$ fall into two different connected components of $F_1$ so that we do not have to try Case 3 in the Extending Whidden et al.’s Algorithm section, or $u$ and $v$ fall into the same connected component of $F_1$ and we can detach a lot of subtrees from $F_1$ in Case 3.

Finally, to enumerate all MAAFs of $T_1$ and $T_2$, we initialize $k = 0$ and then proceed as follows.

1. Simulate the speed-up algorithm for GAF on input $(T_1, T_2, k, F_1)$. During the simulation, whenever an AF $F$ of $T_1$ and $T_2$ is enumerated, perform one of the following steps depending on whether $F$ is acyclic or not:
   a. If $F$ is acyclic, output it.
   b. If $F$ is cyclic, then output all AAFs $F'$ of $T_1$ and $T_2$ such that $F'$ can be obtained from $F$ by performing $k - |F|$ detaching operations on $F$.

2. If at least one AAF of $T_1$ and $T_2$ was outputted in Step 11a or 11b, then stop; otherwise, increase $k$ by 1 and go to Step 1.

Note that Step 11b is nontrivial. As described in the supplementary material of [19], Lemma 2 is very helpful for this purpose. More specifically, we first find a cycle $C$ in $G_{F'}$ in $O(|F'|^2)$ time. By Lemma 2, in order to make $F'$ acyclic, we have to choose one node $r$ of $C$ and modify $F'$ by detaching the subtree of $F'$ rooted at an (arbitrary) child of $r$. Note that since $r$ is a root of $F'$, the subtree of $F'$ rooted at a child of $r$ is achieved by simply deleting $r$ from $F'$ and is hence independent of the choice of the child. Moreover, if the parent $r'$ of the dummy leaf in $F'$ is a node of $C$, then by Lemma 3, we can exclude $r'$ from consideration when choosing $r$. So, we have at most
We only explain how to enumerate all MAAFs of \( T \) in this way, we again construct \( G_T \), and test if it is acyclic. If it is acyclic, then we can output \( F' \); otherwise, we again find a cycle \( C \) in \( G_T \) and use it to modify \( F' \) as before. We repeat modifying \( F' \) in this way, until either \( F' \) becomes acyclic, or \( |F'| = k \) and \( G_{F'} \) is still cyclic. Once \( F' \) becomes acyclic, we output it. The total time taken by Step 11b is \( O(k^4) \), because we make a total number of at most \( O((k-1)^{k-|F'|}) \) choices for breaking cycles.

Experiments show that Heuristics 1 and 2 help us speed up the algorithm substantially. However, the two heuristics may not help in the worst case. That is, we are unable to prove that the two heuristics improve the worst-case time complexity of the algorithm which is \( O(3^d|X| + 3^d(k-1)^{k-d+3}) \) (as shown in [19]), where \( d \) is the size of an MAF of \( T_1 \) and \( T_2 \). We note that \( k \) and \( d \) are usually quite close.

The new algorithm for HNE

In this subsection, we only design an algorithm for HNE. Note that it is trivial to obtain a faster algorithm for HNC by modifying the algorithm for HNE so that it stops immediately once an MAAF is found.

Throughout this subsection, let \( T_1 \) and \( T_2 \) be two phylogenetic trees on the same set \( X \) of species. As mentioned before, we can easily use an MAAF of \( T_1 \) and \( T_2 \) to construct a hybridization network displaying the MAAFs of \( T_1 \) and \( T_2 \). So, we only explain how to enumerate all MAAFs of \( T_1 \) and \( T_2 \).

In the last subsection, we have explained how to speed up HybridNet so that it can enumerate all MAAFs of \( T_1 \) and \( T_2 \) within shorter time. Indeed, we can make HybridNet even faster. The idea is to preprocess \( T_1 \) and \( T_2 \) so that the given trees become smaller or the problem becomes to solve two or more smaller independent subproblems. More specifically, we perform the following two reductions on \( T_1 \) and \( T_2 \) until neither of them is available.

Subtree reduction

Suppose that \( T_1 \) has a non-leaf node \( v_1 \) and \( T_2 \) has a non-leaf node \( v_2 \) such that the subtree of \( T_1 \) rooted at \( v_1 \) is identical to the subtree of \( T_2 \) rooted at \( v_2 \). Then, we modify \( T_1 \) (respectively, \( T_2 \)) by merging the subtree of \( T_1 \) (respectively, \( T_2 \)) rooted at \( v_1 \) (respectively, \( v_2 \)) into a single leaf whose label set is the union of the label sets of the merged leaves. It is known [2] that this reduction preserves the MAAFs of \( T_1 \) and \( T_2 \).

Cluster reduction

Suppose that subtree reductions on \( T_1 \) and \( T_2 \) are not available but \( T_1 \) has a non-leaf node \( v_1 \) and \( T_2 \) has a non-leaf node \( v_2 \) such that the subtree of \( T_1 \) rooted at \( v_1 \) has the same leaf set as the subtree of \( T_2 \) rooted at \( v_2 \). Then, we split \( T_1 \) (respectively, \( T_2 \)) into two trees \( T_1' \) and \( T_2' \) (respectively, \( T_1'' \) and \( T_2'' \)) as follows. \( T_1' \) (respectively, \( T_1'' \)) is simply the subtree of \( T_1 \) (respectively, \( T_2 \)) rooted at \( v_1 \) (respectively, \( v_2 \)), while \( T_1'' \) (respectively, \( T_2'' \)) is obtained by merging the subtree of \( T_1 \) (respectively, \( T_2 \)) rooted at \( v_1 \) (respectively, \( v_2 \)) into a single leaf whose label set is the union of the label sets of the merged leaves. It is known [20] that the set of MAAFs of \( T_1 \) and \( T_2 \) is the Cartesian product of the set of MAAFs of \( T_1' \) and \( T_2' \) and the set of MAAFs of \( T_1'' \) and \( T_2'' \).

After the preprocessing stage, if no cluster reduction has been performed in the preprocessing stage, then we run the sped-up HybridNet (as described in the last subsection) on \( T_1' \) and \( T_2' \); otherwise, we have obtained two or more subproblems. Suppose that we have \( h \) subproblems and the \( i \)-th subproblem (\( 1 \leq i \leq h \)) is to enumerate all MAAFs of two trees \( T_1_i \) and \( T_2_i \). Then, for each \( 1 \leq i \leq h \), we run the sped-up HybridNet to enumerate the set \( M_i \) of MAAFs of \( T_1_i \) and \( T_2_i \). Finally, we output the Cartesian product \( M_1 \times \cdots \times M_h \).

Competing interests

The authors declare that they have no competing interests.

Author’s contributions

ZZC was in charge of algorithm design, algorithm implementation, design of experiments, and manuscript preparation. LW participated in algorithm design and manuscript editing. SY was involved in algorithm implementation and testing. All authors read and approved the final manuscript.

Acknowledgements

We would like to thank the reviewers for their valuable suggestions and comments. ZZC was supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 24500023. LW is supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 121608].

Author details

1Division of Information System Design, Tokyo Denki University, Ishizaka, Hatoyama, Hiki, Saitama 359-0394, Japan. 2Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong.

Received: 19 March 2012 Accepted: 30 June 2012 Published: 2 July 2012

References


