Constructing Parsimonious Hybridization Networks from Multiple Phylogenetic Trees Using a SAT-solver

Vladimir Ulyantsev and Mikhail Melnik

ITMO University, Saint Petersburg, Russia
{ulyantsev,melnik}@rain.ifmo.ru

Abstract. We present an exact algorithm for constructing minimal hybridization networks from multiple trees which is based on reducing the problem to the Boolean satisfiability problem. The main idea of our algorithm is to iterate over possible hybridization numbers and to construct for each of them a Boolean formula that is satisfiable iff there exists a network with such hybridization number. The proposed algorithm is implemented in a software tool PhyloSAT. The experimental evaluation of our algorithm on biological data shows that our method is as far as we know the fastest exact algorithm for the minimal hybridization network construction problem.

Keywords: Phylogenetic networks, Boolean satisfiability, SAT, bioinformatics, genetics

1 Introduction

A phylogenetic network is a powerful model for reticulate evolutionary processes (such as horizontal gene transfer and hybrid specification). Briefly, a phylogenetic network is a directed acyclic graph which has nodes (called reticulation nodes) with more than one incoming edge. Phylogenetic networks have been studied by several researchers [6–8]. There are several formulations of phylogenetic network construction problem with various modelling assumptions and different types of input data. In this paper we focus on the specific type of phylogenetic networks called hybridization networks [4, 11].

As input data for hybridization network construction we consider a set of gene trees on the same set of taxa. Each gene tree models the evolutionary history of some gene. Due to reticulate evolutionary events, these trees can have different topologies. The aim is to construct a hybridization network containing the smallest possible number of reticulation nodes and displaying each of the input trees.

Most of the algorithms for hybridization network construction are heuristic [10, 13] and usually deal with only two trees. However, the exact algorithm PIRN_C has been introduced recently [13], which is able to process more than two input trees.

In this paper we introduce a new approach for exact parsimonious hybridization network construction from multiple input trees based on satisfiability (SAT) solvers. The SAT problem is known to be NP-complete [3], but state-of-the-art SAT-solvers running on modern hardware are able to solve SAT instances having tens of thousands of variables and hundreds of thousands of clauses in several minutes.

SAT-solver based algorithms have been successfully applied to efficiently calculate evolutionary tree measures [2] as well as to solve problems in other domains such as finite-state machine induction [5], software verification [1].

The general outline of our approach is to convert an instance of hybridization network construction problem to an instance of the SAT problem (a Boolean formula), solve it using a SAT-solver and then if the solution exists convert the satisfying assignment into the hybridization network. Our approach leads to an exact algorithm that outperforms PIRN_C on our tests.

The paper is structured as follows. Section 2 gives the formal definitions, Section 3 describes the Boolean formula construction process and Section 4 gives the experimental results. The paper is concluded in Section 5.

2 Definitions and Background

We define a *phylogenetic tree* as a leaf-labelled tree constructed over a set of taxa. Throughout this paper we assume that trees are rooted and binary.

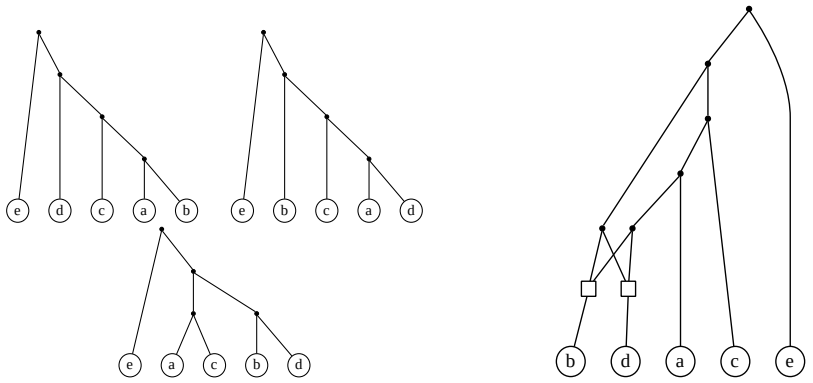
For any node v let $d^-(v)$ be the in-degree of v and $d^+(v)$ be the out-degree of v. A *hybridization network* on a set of taxa X is a directed acyclic graph with a single root ρ and leaves bijectively mapped to the set of taxa X. If $d^-(v) > 1$ then node v is called a *reticulation node*. In this paper we assume that $d^-(v) = 2$ and $d^+(v) = 1$ is true for every reticulation node v. We do this assumption by noting that we can convert a reticulation node with in-degree of three or more to a sequence of reticulation nodes with in-degrees of two [12]. Other nodes are regular tree nodes.

Every hybridization network N can be reduced to the tree. To do this we firstly keep only one of the incoming edges for each reticulation node in N. And secondly we contract edges to remove any node v such that $d^-(v) = d^+(v) = 1$. With these two steps we reduce the network N to the tree T'.

We say that hybridization network N *displays* phylogenetic tree T if we can choose the edges of reticulation nodes in such a way that after edge-contraction the obtained tree T' will be isomorphic to T. In Figure 1 each of the three trees is displayed in the hybridization network.

A *hybridization number* of network N with root ρ is commonly defined as $h(N) = \sum_{v \neq \rho} (d^-(v) - 1)$. Note that under our assumptions $h(N)$ simply equals the number of reticulation nodes.

Suppose we are given a set of K phylogenetic trees $T_1, T_2, ...T_k$ over the same set of taxa. The *minimal hybridization network* for that set of trees is a network $N_{\min}$ that displays each tree and has the smallest hybridization number possible. Note that there can be several networks with equal hybridization number.



(a) Three gene trees over the set of taxa {a, b, c, d, e}

(b) A hybridization network for the trees from Fig. 1 (a)

Fig. 1. An illustration of a hybridization network for three trees with two reticulation nodes. Reticulation nodes are shown as boxes.

The most parsimonious hybridization network problem is defined as follows: given a set of phylogenetic trees $T_1, T_2, ..., T_k$ over the same set of taxa, construct the minimal hybridization network for this set of trees.

It has been shown that even for the case of $k = 2$ the construction of such a network is an NP-complete problem [3]. As far as we know there exists only one algorithm for the construction of the most parsimonious hybridization network [13].

3 Algorithm

The main idea of the algorithm is to iterate over possible values of the hybridization number and to construct and solve a Boolean formula that represents a hybridization network with this hybridization number.

3.1 Pre-processing

Before the actual Boolean formula encoding we modify the input and split it into several tasks to reduce the size and the complexity of the problem. We do this according to the rules from [2]. To define these rules we first need to define the term *cluster*: a set of taxa A is a cluster in trees $T_1, T_2, ...T_k$ if there exists a node in each tree with the set of leaves of its subtree equal to the set A. The reduction rules are as follows:

1. **Subtree reduction rule**: replace every subtree which is present in all input trees with a leaf with a new label.
2. **Cluster reduction rule**: for each cluster A replace the subtrees containing it with a leaf with a new label and add a new task for processing which consists of deleted subtrees $T'_1, T'_2, ...T'_k$ with leaf set A.

(a) Simple tree (b) Simple tree with dummy root attached

Fig. 2. An illustration of an attachment of the dummy root to the tree.

After we split the task into a set of simpler tasks, we add a dummy root to each tree of each task along with the new dummy leaf for tree consistency. Figure 2 illustrates the procedure. This is done to ensure that all the trees in the input share a common root. This dummy root will be deleted on the post-processing stage. At this stage we have a set of tasks that will be solved separately and then their results will be merged at the post-processing stage.

3.2 Search of the minimal hybridization number

To solve a subtask we need to find the lowest hybridization number k such that there exists a hybridization network with this hybridization number. To do this we use downwards search, i.e. we iterate through possible values of k starting from the highest and construct a Boolean formula corresponding to the current k. We decrease k until the solver can not satisfy the formula, and this means that the previous value of k was the lowest possible.

There are other strategies of searching the minimal value of k. For example, we can start from zero and increase the value of k until the solver will be able to satisfy the formula, or we can use binary search. The results of the binary search were close to the ones of downwards method, but in some cases, when the binary search tried to satisfy formulae with values of k less than minimal possible hybridization number, its results were also poor. This can be explained by an experimental observation that it is much easier for the solver to produce an answer if the formula is satisfiable than if it is not. In case of an unsatisfiable formula the solver must check every possible answer to ensure that there is no solution; this is not needed if the formula is satisfiable. Because of this the results of the upwards search were very poor. An obvious method for reducing the search time is to limit the range of possible values of k by using different heuristics to find close upper and lower bounds for k. Possible candidates are PIRN_{CH} [13], RIATA-HGT [9] and MURPAR [10]. We do not consider such optimizations in this paper.

3.3 Encoding the Boolean formula

Having a set of trees T over a set of taxa A and a fixed hybridization number k, we will construct the Boolean formula which is satisfiable iff there exists a

hybridization network that displays each tree in T and its hybridization number equals to k. To do this we first notice that a network over the set of taxa of size n with hybridization number k will have $2(n+k)-1$ nodes. As we add a dummy root and a dummy leaf, we finally have $2(n+k)+1$ nodes, k of which are reticulation nodes, $n+1$ are leaves and others are usual tree nodes.

We enumerate all the nodes in such a way that leaves are numbered in range $[0, n]$, regular nodes have numbers in range $[n+1, 2n+k]$ and all reticulation nodes have numbers in range $[2n+k+1, 2(n+k)]$. We also assume that the number of any leaf or regular node is less than the number of its parent. This is done to avoid consideration of isomorphic networks during SAT solving. Such enumeration allows us to define the following sets of nodes for each node v: $PC(v)$ is the set of possible children of v, $PP(v)$ is the set of possible parents of v and $PU(v)$ is the set of possible ancestors of v. Also let R be the set of reticulation nodes, L be the set of leaves and V be the set of regular nodes. Now we will describe variables and clauses required to construct the Boolean formula.

Network structure encoding. First of all we encode the structure of the network. We introduce the following literals.

1. $l_{v,u}$ and $r_{v,u}$ for each $v \in V, u \in PC(v)$: $l_{v,u}$ (or $r_{v,u}$) is true iff regular node v has node u as its left (right) child.
2. $p_{v,u}$ for each $v \in L \cup V \backslash \{\rho\}, u \in PP(v)$: $p_{v,u}$ is true iff u is the parent of a regular node v.
3. $p^l_{v,u}$ and $p^r_{v,u}$ for each $v \in R, u \in PP(v)$: $p^l_{v,u}$ (or $p^r_{v,u}$) is true iff u is the left (right) parent of a reticulation node v.
4. $c_{v,u}$ for each $v \in R, u \in PC(v)$: $c_{v,u}$ is true iff u is a child of a reticulation node v.

This takes $O((n+k)^2)$ literals for specifying the network structure, and by noticing that $k < n$ we have $O(n^2)$ literals.

To encode the uniqueness of parents and children we use an obvious pattern that requires $O(n^2)$ clauses for each node. For example consider parent variables. We state that a node can have at least one parent and at most one parent. For parents of the regular node v this can be expressed in the following way:

$$\left(\bigvee_{u \in PP(v)} p_{v,u} \right) \wedge \left(\bigwedge_{i,j \in PP(v); i<j} (p_{v,i} \rightarrow \neg p_{v,j}) \right).$$

Using this pattern, we add the uniqueness constraints for literals l, r, p, p^l, p^r and c. These clauses are defined in sections 1–4 of Table 1. We also add constraints to order the numbers of children of regular nodes and parents of reticulation nodes. They are listed in section 5 of Table 1.

Network is consistent if for every pair of nodes the parent relation implies the child relation and vice versa. Thus we add constraints that connect parent literals with children literals for all the types of nodes. See sections 6–9 of Table 1 for these clauses. The last step of network construction is to deal with

Table 1. Clauses for network structure encoding.

Number	Clause	Range
1.1	$p_{v,u_1} \vee ... \vee p_{v,u_k}$	$v \in V; u_1 ... u_k \in PP(v)$
1.2	$p_{v,u} \rightarrow \neg p_{v,w}$	$v \in V; u, w \in PP(v)$
2.1	$l_{v,u_1} \vee ... \vee l_{v,u_k}$	$v \in V; u_1 ... u_k \in PC(v)$
2.2	$l_{v,u} \rightarrow \neg l_{v,w}$	$v \in V; u, w \in PC(v)$
2.3	$r_{v,u_1} \vee ... \vee r_{v,u_k}$	$v \in V; u_1 ... u_k \in PC(v)$
2.4	$r_{v,u} \rightarrow \neg r_{v,w}$	$v \in V; u, w \in PC(v)$
3.1	$c_{v,u_1} \vee ... \vee c_{v,u_k}$	$v \in R; u_1 ... u_k \in PC(v)$
3.2	$c_{v,u} \rightarrow \neg c_{v,w}$	$v \in R; u, w \in PC(v)$
4.1	$p^l_{v,u_1} \vee ... \vee p^l_{v,u_k}$	$v \in R; u_1 ... u_k \in PP(v)$
4.2	$p^l_{v,u} \rightarrow \neg p^l_{v,w}$	$v \in R; u, w \in PP(v)$
4.3	$p^r_{v,u_1} \vee ... \vee p^r_{v,u_k}$	$v \in R; u_1 ... u_k \in PP(v)$
4.4	$p^r_{v,u} \rightarrow \neg p^r_{v,w}$	$v \in R; u, w \in PP(v)$
5.1	$l_{v,u} \rightarrow \neg r_{v,w}$	$v \in V; u, w \in PC(v) : u \geq w$
5.2	$p^l_{v,u} \rightarrow \neg p^r_{v,w}$	$v \in R; u, w \in PP(v) : u \geq w$
6.1	$l_{v,u} \rightarrow p_{u,v}$	$v \in V; u \in V; u \in PC(v)$
6.2	$r_{v,u} \rightarrow p_{u,v}$	$v \in V; u \in V; u \in PC(v)$
6.3	$p_{u,v} \rightarrow (l_{v,u} \vee r_{v,u})$	$v \in V; u \in V; u \in PC(v)$
7.1	$l_{v,u} \rightarrow (p^l_{u,v} \vee p^r_{u,v})$	$v \in V; u \in R; u \in PC(v)$
7.2	$r_{v,u} \rightarrow (p^l_{u,v} \vee p^r_{u,v})$	$v \in V; u \in R; u \in PC(v)$
7.3	$p^l_{u,v} \rightarrow (l_{v,u} \vee r_{v,u})$	$v \in V; u \in R; u \in PC(v)$
7.4	$p^r_{u,v} \rightarrow (l_{v,u} \vee r_{v,u})$	$v \in V; u \in R; u \in PC(v)$
8.1	$c_{v,u} \rightarrow p_{u,v}$	$v \in R; u \in V; u \in PC(v)$
8.2	$p_{u,v} \rightarrow c_{v,u}$	$v \in R; u \in V; u \in PC(v)$
9.1	$c_{v,u} \rightarrow (p^l_{u,v} \vee p^r_{u,v})$	$v \in R; u \in R; u \in PC(v)$
9.2	$p^l_{u,v} \rightarrow c_{v,u}$	$v \in R; u \in R; u \in PC(v)$
9.3	$p^r_{u,v} \rightarrow c_{v,u}$	$v \in R; u \in R; u \in PC(v)$
10.1	$c_{v,u} \rightarrow \neg p^l_{v,w}$	$v \in R; u \in PC(v); w \in PP(v) : u \geq w$
10.2	$c_{v,u} \rightarrow \neg p^r_{v,w}$	$v \in R; u \in PC(v); w \in PP(v) : u \geq w$

the enumeration around reticulation nodes. To do this, we add constraints to fix relative numbers of children and parents of reticulation nodes. They are defined in section 10 of Table 1.

Since we need $O(n^2)$ clauses for each node to represent the uniqueness of its parents and children and $O(n)$ clauses for each node to represent the parents-children relation, we finally get $O(n^3)$ clauses in total to represent the network structure.

Mapping trees to the network. To express that the network contains all the input trees we add literals that represent the mapping of the tree nodes to the network nodes.

1. $x_{t,v_t,v}$ for each $t \in T, v_t \in V(t), v \in V$: $x_{t,s,v}$ is true iff regular node v represents node v_t from tree t, i.e. x literals represent injective mapping of network nodes to tree nodes. An example of such mapping is shown in

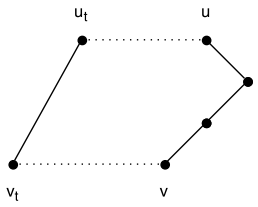


Fig. 3. An illustration of a piece of mapping of the nodes from the tree (on the left) to the nodes from the network (on the right). Nodes that are injectively mapped are connected with dotted line. When displaying the tree all the edges in the path from u to v will be contracted to the single edge form u_t to v_t.

Figure 3. Note that leaves of the trees are bijectively mapped to leaves of the network thus there is no need to introduce x variables for them.

2. $d_{t,v}$ for each $t \in T, v \in R : d_{t,v}$ is true iff left parent edge of reticulation node v is used to display tree t, i.e. it specifies direction of necessary parent to display current tree.
3. $u^r_{t,v}$ for each $t \in T, v \in R : u^r_{t,v}$ is true iff child of reticulation node v is used to display tree t.
4. $u_{t,v}$ for each $t \in T, v \in V : u_{t,v}$ is true iff regular node v is used to display tree t.
5. $a_{t,v,u}$ for each $t \in T, v \in V, u \in PU(v) : a_{t,v,u}$ is true iff the regular node u is an ancestor of node v and node u corresponds to such node u_t from the tree t that all the edges on the path from u to v are contracted to a single edge of tree t, i.e. node u is the first node on the path from v to root that is mapped to some node in the tree t. We will say that the node u is the direct parent of the node v considering tree t. Note that the node v is not necessary present in tree t. In Figure 3 node u is the parent of all the nodes starting from the node v considering tree t.

We have extra $O(tn^2)$ literals to match input trees to the network. Hence, we have $O(tn^2)$ literals in total.

We specify constraints for relations between network nodes and tree nodes. First of all we define at-least-one and at-most-one constraints for literals x and a. Note that in case of x literals we have a restriction that at most one node from the tree corresponds to the network node, and that at most one node from the network corresponds to the tree node. These clauses are defined in sections 1–2 of Table 2. Because of dummy roots we know that roots of input trees are mapped to the root of the network so we have $x_{t,\rho_t,\rho}$ set to true for every tree. Also note that $x_{t,v_t,v}$ implies $u_{t,v}$ by definition. See section 3 of Table 2 for these clauses.

Furthermore, if we know that node v is a leaf and its parent u corresponds to node u_t from tree t, then we can conclude that $a_{t,v,u}$ is true i.e. u is the direct parent of v in tree t. Same observation can be done for non-leaves, i.e. if we know that v corresponds to v_t and another node u corresponds to u_t and u_t is parent

Table 2. Clauses for the mapping of the tree nodes to the network nodes.

	Clause	Range
1.1	$a_{t,v,u_1} \vee ... \vee a_{t,v,u_k}$	$v \in V \cup L \cup R; u_1...u_k \in PU(v)$
1.2	$a_{t,v,u} \to \neg a_{t,v,w}$	$v \in V \cup L \cup R; u, w \in PU(v)$
2.1	$x_{t,t_v,v_1} \vee ... \vee x_{t,t_v,v_k}$	$t \in T; t_v \in V(t); v_1...v_k \in V$
2.2	$x_{t,t_v,v} \to \neg x_{t,t_v,w}$	$t \in T; t_v \in V(t); v, w \in V$
2.3	$x_{t,t_v,v} \to \neg x_{t,t_w,v}$	$t \in T; t_v, t_w \in V(t); v \in V$
3.1	$x_{t,v_t,v} \to u_{t,v}$	$t \in T; v \in V; v_t \in V(t)$
3.2	$x_{t,\rho_t,\rho}$	$t \in T; \rho_t = \rho(t)$
4.1	$x_{t,u_t,u} \to a_{t,v,u}$	$t \in T; v \in L; u \in PP(v); u_t \in V(t)$
4.2	$(x_{t,v_t,v} \wedge x_{t,u_t,u}) \to a_{t,v,u}$	$t \in T; v \in V; u \in PP(v); v_t \in V(t) : u_t = p(v_t)$
4.3	$(x_{t,v_t,v} \wedge a_{t,v,u}) \to x_{t,u_t,u}$	$t \in T; v \in V; u \in PP(v); v_t \in V(t) : u_t = p(v_t)$
4.4	$x_{t,v_t,v} \to \neg x_{t,u_t,u}$	$t \in T; v \in V; u \in V; v_t \in V(t); u_t = p(v_t) : u < v$
5.1	$\neg x_{t,v_t,v}$	$t \in T; v \in V; v_t \in V(t) : v_t < \text{size}(\text{subtree}(v_t))$
5.2	$\neg x_{t,v_t,v}$	$t \in T; v \in V; v_t \in V(t) : v_t > \text{size}(t) - \text{depth}(v_t)$
5.3	$\neg x_{t,v_t,v} \vee \neg x_{t',v_{t'},v}$	$t, t' \in T; v \in V; v_t \in V(t); v_{t'} \in V(t')$: subtrees of t and t' have disjoint sets of taxa

of v_t then u is the direct parent of v considering tree t. On the other hand if we know that v corresponds to v_t and we know that u is the direct parent of v considering tree t, then u should correspond to parent of v_t. We also should take care about enumeration, so we add a constraint stating that if node u_t is the parent of the node v_t, then the number of the corresponding node u should be greater than the one of the node v, i.e. $x_{t,v_t,v} \to \neg x_{t,u_t,u}$ for each u and v such that $u < v$. These clauses are presented in section 4 of Table 2.

We also add some heuristic constraints related to the trees structure. Notice that the number of the node in the network can not be less than the size of the subtree of the corresponding node v_t in tree t and also it can not be greater than the size of the tree minus depth of v_t. Also if node v_t in tree t and node $v_{t'}$ in tree t' have disjoint sets of taxa in their subtrees then they can not be mapped to the same node in the network. These clauses can be found in section 5 of Table 2.

Next, we add constraints that connect child-parent relations in trees with indirect child-parent relations in the network. First of all consider the regular node u that is the direct parent of the node v and u is used to display tree t then u is also the parent of v considering tree t. Vice versa, if we know that u is the parent of v in tree t, then u should be used for displaying that tree. If we do not use u for displaying tree t then we should share the information stored in the a variables between v and u because they will have the same parent considering tree t. We do this with the following constraints:

$$((p_{v,u} \wedge \neg u_{t,u} \wedge a_{t,u,w}) \to a_{t,v,w}) \wedge ((p_{v,u} \wedge \neg u_{t,u} \wedge a_{t,v,w}) \to a_{t,u,w})\,.$$

These clauses are listed in section 1 of Table 3.

Now consider the reticulation node v and its parent u. If u is a reticulation node then we should share information about its parent considering tree t but

Table 3. Clauses for translating child-parent relations from the trees to the network.

	Clause	Range
1.1	$(p_{v,u} \wedge u_{t,u}) \rightarrow a_{t,v,u}$	$t \in T; v \in V \cup L; u \in PP(v); u \in V$
1.2	$(p_{v,u} \wedge a_{t,v,u}) \rightarrow u_{t,u}$	$t \in T; v \in V \cup L; u \in PP(v); u \in V$
1.3	$(p_{v,u} \wedge \neg u_{t,u} \wedge a_{t,u,w}) \rightarrow a_{t,v,w}$	$t \in T; v \in V \cup L; u \in PP(v); u \in V; w \in PP(u)$
1.4	$(p_{v,u} \wedge \neg u_{t,u} \wedge a_{t,v,w}) \rightarrow a_{t,u,w}$	$t \in T; v \in V \cup L; u \in PP(v); u \in V; w \in PP(u)$
2.1	$(p^l_{v,u} \wedge d_{t,v} \wedge a_{t,u,w}) \rightarrow a_{t,v,w}$	$t \in T; v \in R; u \in PP(v); u \in R; w \in PU(u)$
2.2	$(p^l_{v,u} \wedge d_{t,v} \wedge a_{t,v,w}) \rightarrow a_{t,u,w}$	$t \in T; v \in R; u \in PP(v); u \in R; w \in PU(u)$
2.3	$(p^r_{v,u} \wedge \neg d_{t,v} \wedge a_{t,u,w}) \rightarrow a_{t,v,w}$	$t \in T; v \in R; u \in PP(v); u \in R; w \in PU(u)$
2.4	$(p^r_{v,u} \wedge \neg d_{t,v} \wedge a_{t,v,w}) \rightarrow a_{t,u,w}$	$t \in T; v \in R; u \in PP(v); u \in R; w \in PU(u)$
2.5	$(p^l_{v,u} \wedge d_{t,v} \wedge u_{t,u}) \rightarrow a_{t,v,u}$	$t \in T; v \in R; u \in PP(v); u \in V$
2.6	$(p^r_{v,u} \wedge \neg d_{t,v} \wedge u_{t,u}) \rightarrow a_{t,v,u}$	$t \in T; v \in R; u \in PP(v); u \in V$
2.7	$(p^l_{v,u} \wedge d_{t,v} \wedge \neg u_{t,u} \wedge a_{t,u,w}) \rightarrow a_{t,v,w}$	$t \in T; v \in R; u \in PP(v); u \in V; w \in PU(u)$
2.8	$(p^l_{v,u} \wedge d_{t,v} \wedge \neg u_{t,u} \wedge a_{t,v,w}) \rightarrow a_{t,u,w}$	$t \in T; v \in R; u \in PP(v); u \in V; w \in PU(u)$
2.9	$(p^r_{v,u} \wedge \neg d_{t,v} \wedge \neg u_{t,u} \wedge a_{t,u,w}) \rightarrow a_{t,v,w}$	$t \in T; v \in R; u \in PP(v); u \in V; w \in PU(u)$
2.10	$(p^r_{v,u} \wedge \neg d_{t,v} \wedge \neg u_{t,u} \wedge a_{t,v,w}) \rightarrow a_{t,u,w}$	$t \in T; v \in R; u \in PP(v); u \in V; w \in PU(u)$
3.1	$(p^l_{v,u} \wedge \neg d_{t,v}) \rightarrow \neg u^r_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in R$
3.2	$(p^r_{v,u} \wedge d_{t,v}) \rightarrow \neg u^r_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in R$
3.3	$(p^l_{v,u} \wedge \neg d_{t,v}) \rightarrow \neg u_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in V$
3.4	$(p^r_{v,u} \wedge d_{t,v}) \rightarrow \neg u_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in V$
4.1	$(p^l_{v,u} \wedge d_{t,v} \wedge u^r_{t,v}) \rightarrow u^r_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in R$
4.2	$(p^r_{v,u} \wedge \neg d_{t,v} \wedge u^r_{t,v}) \rightarrow u^r_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in R$
4.3	$\neg u^r_{t,v} \rightarrow \neg u^r_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in R$
4.4	$\neg u^r_{t,v} \rightarrow \neg u_{t,u}$	$t \in T; v \in R; u \in PP(v); u \in V$
4.5	$c_{v,u} \rightarrow u^r_{t,v}$	$t \in T; v \in R; u \in PC(v); u \in V \cup L$
5.1	$p_{v,u} \rightarrow \neg a_{t,u,w}$	$t \in T; v \in V \cup L; u \in PP(v); u \in R;$ $w \in PU(u) : w \leq v$
5.2	$(p_{v,u} \wedge a_{t,u,w}) \rightarrow a_{t,v,w}$	$t \in T; v \in V \cup L; u \in PP(v); u \in R;$ $w \in PU(u) : w > v$
5.3	$(p_{v,u} \wedge a_{t,v,w}) \rightarrow a_{t,u,w}$	$t \in T; v \in V \cup L; u \in PP(v); u \in R;$ $w \in PU(u) : w > v$

only when direction of parent u matches direction specified in v. If u is a regular node and u is used for displaying tree t, then u is the direct parent of v considering tree t. On the other hand, if u is not used then we should share information about its parent considering tree t also only when direction of u matches direction specified in v. These constraints are presented in section 2 of Table 3.

In cases when direction of u does not match direction specified in v we should not use it. See section 3 of Table 3 for these clauses. If u is a reticulation node, its direction matches direction specified in v and v is used, then u should also be used. Note that if we do not use the child of node v for displaying then we also should not use its parents. And the crucial point is that if the child of node v is a regular node then we should use v for displaying. These clauses are defined in section 4 of Table 3.

We also add clauses to forbid incorrect numeration, if node v has a reticulation parent u then there cannot exist a node w that its number is less than the

number of v, and $a_{t,u,w}$ is true. And for all w with numbers greater than v we add clauses to share information of a variables. See section 5 of Table 3 for these clauses.

Again the most expensive clauses are clauses that represent uniqueness of variables a and x. We have $O(n^2)$ clauses for each node for each tree, so we have $O(tn^3)$ clauses to map trees to the network. This sums up to $O(tn^3)$ clauses in total.

3.4 Solving the Boolean formula and post-processing

To solve the generated Boolean formula we use a SAT-solver CryptoMiniSat 4.2.0 (http://www.msoos.org/cryptominisat4/). Choosing the most appropriate solver is not considered in this paper, however several experimentations were made by M. Bonet and K. John [2] so it is one of the topics of further research.

After solving a task we reconstruct the network from the SAT-solver output. After that we delete the dummy root and the corresponding leaf from the network. And when all the subtasks of the original task are solved, we merge their networks into a single hybridization network corresponding to the original task.

4 Experiments

To test the performance of our algorithm we evaluated PhyloSAT on a grass (Poaceae) dataset provided by the Grass Phylogeny Working Group (Grass Phylogeny Working Group, 2001). There are 57 test cases in the dataset with up to 47 taxa. All experiments were performed using a machine with an AMD Phenom II X6 1090T 3.2 GHz processor on Ubuntu 14.04. All tests were run with a reasonable time limit of 1000 seconds. For comparison we also ran PIRN_C and PIRN_CH on the same test cases.

Out of 57 test cases, 9 were not solved even by heuristic algorithms in time. PhyloSAT was able to produce the optimal answer for 28 test cases. From these 28 test cases PIRN_C was able to solve only 21 and in all of them hybridization number was less than 6 which shows that PIRN_C is not capable of building hybridization networks with large hybridization number. On all test cases PIRN_C was slower than PhyloSAT. Even PIRN_CH did not solve 2 of these 28 test cases in time. PIRN_CH was faster than PhyloSAT only on 5 test cases and was significantly slower on 2 test cases. Twelve more test cases were not solved by PIRN_C, but PhyloSAT was able to produce some (possibly non-optimal) network. PIRN_CH did not solve 3 of these 12 cases in time, in 3 cases produced less optimal network than PhyloSAT, in 2 cases more optimal and in the rest 4 cases results were equal. From these 12 cases PIRN_CH produced an optimal network for only 2, and PhyloSAT found an optimal network for 4 cases but was unable to prove their optimality. 8 more test cases had equal trees in input, so they had an obvious answer. Results of the simulation are summarized in Table 4.

Experiments showed that in some cases PhyloSAT is able to find an optimal network but then it spends a lot of time trying to find the network with

Table 4. Experimental results

Algorithm	Solved cases	Optimally solved cases
PhyloSAT	48	32
PIRN_{C}	26	26
PIRN_{CH}	43	28

hybridization number one less than optimal and that time is much higher than reasonable limit. We can avoid wasting time on useless computation in cases when we find a network with hybridization number equal to the heuristic lower bound (like PIRN_{CH} does). Besides we found that it also costs a lot of time to build a network with a big hybridization number when the minimal hybridization number is small. Thus, a close upper bound for the minimal hybridization number will also be very useful and will save lots of computational time.

5 Conclusion

We have proposed an algorithm for constructing an exact parsimonious hybridization network from multiple phylogenetic trees. An implementation of our algorithm is available for download at https://github.com/ctlab/PhyloSAT. Experiments showed that PhyloSAT outperforms PIRN_{C} in all cases and performs reasonably well comparing to the heuristic PIRN_{CH}. However in the cases of large hybridization numbers search and construction of optimal network is still a very challenging problem. In the future we plan to use existing heuristics and estimations on lower and upper bounds of the hybridization number to limit search bounds and thus reduce the running time of our algorithm.

Acknowledgements

This work was financially supported by the Government of Russian Federation, Grant 074-U01. Authors would like to thank Igor Buzhinsky and Daniil Chivilikhin for helpful comments and conversations.

References

1. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in computers 58, 117–148 (2003)
2. Bonet, M.L., John, K.S.: Efficiently calculating evolutionary tree measures using SAT. In: Theory and Applications of Satisfiability Testing-SAT 2009, pp. 4–17. Springer (2009)
3. Bordewich, M., Semple, C.: Computing the minimum number of hybridization events for a consistent evolutionary history. Discrete Applied Mathematics 155(8), 914–928 (2007)
4. Chen, Z.Z., Wang, L.: Hybridnet: a tool for constructing hybridization networks. Bioinformatics 26(22), 2912–2913 (2010)

5. Heule, M.J., Verwer, S.: Exact dfa identification using sat solvers. In: Grammatical Inference: Theoretical Results and Applications, pp. 66–79. Springer (2010)
6. Huson, D.H., Rupp, R., Scornavacca, C.: Phylogenetic networks: concepts, algorithms and applications. Cambridge University Press (2010)
7. Morrison, D.A.: Introduction to phylogenetic networks. RJR Productions (2011)
8. Nakhleh, L.: Evolutionary phylogenetic networks: models and issues. In: Problem solving handbook in computational biology and bioinformatics, pp. 125–158. Springer (2011)
9. Nakhleh, L., Ruths, D., Wang, L.S.: RIATA-HGT: a fast and accurate heuristic for reconstructing horizontal gene transfer. In: Computing and Combinatorics, pp. 84–93. Springer (2005)
10. Park, H.J., Nakhleh, L.: MURPAR: a fast heuristic for inferring parsimonious phylogenetic networks from multiple gene trees. In: Bioinformatics Research and Applications, pp. 213–224. Springer (2012)
11. Semple, C.: Hybridization networks. Department of Mathematics and Statistics, University of Canterbury (2006)
12. Wu, Y.: Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees. Bioinformatics 26(12), i140–i148 (2010)
13. Wu, Y.: An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. Journal of Computational Biology 20(10), 792–804 (2013)