

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 a Boolean formula for each of them 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 the directed acyclic graph which has nodes (called reticulation nodes) with more than one incoming edge. Phylogenetic networks have been studied by several researchers [7–9]. 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, 12].

We consider a set of gene trees on the same set of taxa as input data for hybridization network construction. 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 [11, 14] and usually deal with only two trees. However, the exact algorithm PIRN_C which is able to process more than two input trees has been introduced recently [14].

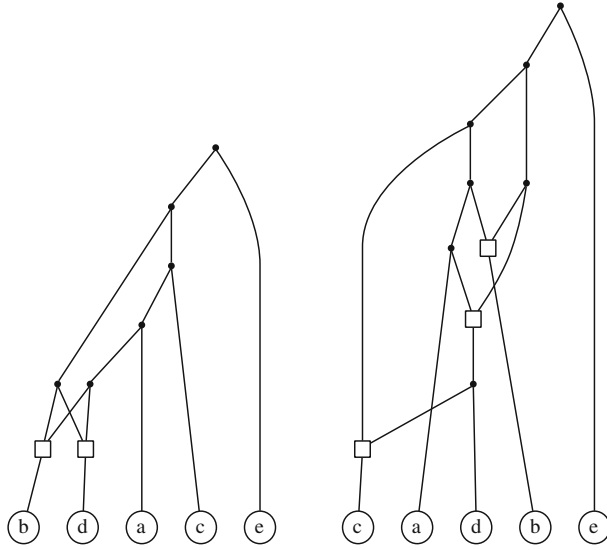


Fig. 2. Possible hybridization networks for the trees from Fig. 1 with two and three hybridization events respectively. Reticulation nodes are shown as boxes.

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 Fig. 2 each of the three trees from Fig. 1 is displayed in both hybridization networks.

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, \dots, 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.

The *most parsimonious hybridization network problem* is defined as follows: given a set of phylogenetic trees T_1, T_2, \dots, 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 [14].

3 Algorithm

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

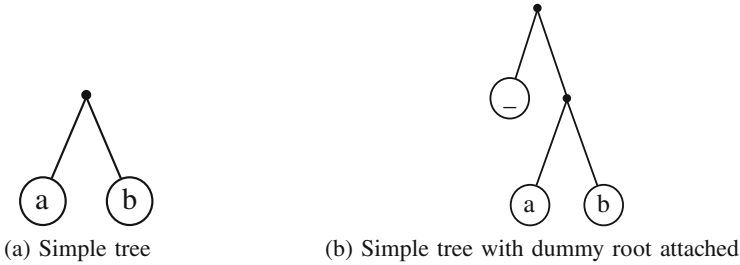


Fig. 3. An illustration of an attachment of the dummy root to the tree.

hybridization network with this hybridization number. We implemented our algorithm in a software tool PhyloSAT which is available for download at GitHub¹.

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, \dots, 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, \dots, T'_k with leaf set A .

After splitting 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 3 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 .

¹ <https://github.com/ctlab/PhyloSAT>.

We decrease k until the solver cannot 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 the 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 usually 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 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} [14], RIATA-HGT [10] and MURPAR [11]. 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 a 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, 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 $v \in L \cup V \setminus \{\rho\}, u \in PP(v)$: $p_{v,u}$ is true iff u is the parent of a regular node v .

3. $p_{v,u}^l$ and $p_{v,u}^r$ for each $v \in R, u \in PP(v) : p_{v,u}^l$ (or $p_{v,u}^r$) 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 tried to use an obvious pairwise encoding that requires $O(n^2)$ clauses for each node as well as more efficient Bimander encoding [6]. However, the Bimander encoding gave no speed boost because of the existence of other constraints that require $O(n^2)$ clauses per node. Thus we will describe the pairwise encoding for simplicity. 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 Sects. 1–4 of Table 1. We also add constraints to order the number of children of regular nodes and parents of reticulation nodes. They are listed in Sect. 5 of Table 1.

A 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 Sects. 6–9 of Table 1 for these clauses. The last step of the network construction is to deal with 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 Sect. 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. 4. 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.

Table 1. Clauses for network structure encoding.

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

2. $d_{t,v}$ for each $t \in T, v \in R$: $d_{t,v}$ is true iff the 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. $u_{t,v}^r$ for each $t \in T, v \in R$: $u_{t,v}^r$ is true iff the 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 regular node u is an ancestor of node v and node u corresponds to such node u_t from 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 the root that is mapped to some node in tree t . We will say that node u is the direct parent of node v considering tree t . Note that node v is not necessary present in tree t . In Fig. 4 node u is the parent of all the nodes starting from node v considering tree t .

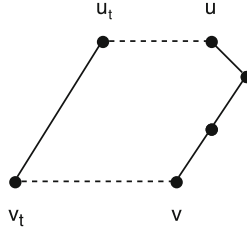


Fig. 4. An illustration of a piece of mapping of the nodes from the tree (on the left) to the nodes from the network (on the right). Nodes that are injectively mapped are connected by a dotted line. When displaying the tree all the edges in the path from u to v will be contracted to the single edge from u_t to v_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, 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 Sects. 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 be true for every tree. Also note that $x_{t,v_t,v}$ implies $u_{t,v}$ by definition. See Sect. 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 . The 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 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 of 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. $x_{t,v_t,v} \rightarrow \neg x_{t,u_t,u}$ for each u and v such that $u < v$. These clauses are presented in Sect. 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 cannot be less than the size of the subtree of the corresponding node v_t in tree t and also it cannot 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 cannot be mapped to the same node in the network. These clauses can be found in Sect. 5 of Table 2.

Next, we add constraints that connect child-parent relations in trees with indirect child-parent relations in the network. First, 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

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

	Clause	Range
1.1	$a_{t,v,u_1} \vee \dots \vee a_{t,v,u_k}$	$v \in V \cup L \cup R; u_1 \dots u_k \in PU(v)$
1.2	$a_{t,v,u} \rightarrow \neg a_{t,v,w}$	$v \in V \cup L \cup R; u, w \in PU(v)$
2.1	$x_{t,t_v,v_1} \vee \dots \vee x_{t,t_v,v_k}$	$t \in T; t_v \in V(t); v_1 \dots v_k \in V$
2.2	$x_{t,t_v,v} \rightarrow \neg x_{t,t_v,w}$	$t \in T; t_v \in V(t); v, w \in V$
2.3	$x_{t,t_v,v} \rightarrow \neg x_{t,t_w,v}$	$t \in T; t_v, t_w \in V(t); v \in V$
3.1	$x_{t,v_t,v} \rightarrow u_{t,v}$	$t \in T; v \in V; v_t \in V(t)$
3.2	$x_{t,\rho_t,\rho}$	$t \in T; \rho_t = \rho(t)$
4.1	$x_{t,u_t,u} \rightarrow a_{t,v,u}$	$t \in T; v \in L; u \in PP(v); u_t \in V(t)$
4.2	$(x_{t,v_t,v} \wedge x_{t,u_t,u}) \rightarrow a_{t,v,u}$	$t \in T; v \in V; u \in PP(v); v_t \in V(t) : u_t = p(v_t)$
4.3	$(x_{t,v_t,v} \wedge a_{t,v,u}) \rightarrow x_{t,u_t,u}$	$t \in T; v \in V; u \in V; v_t \in V(t); u_t = p(v_t) : u < v$
4.4	$x_{t,v_t,v} \rightarrow \neg x_{t,u_t,u}$	$t \in T; v \in V; u \in V; v_t \in V(t); u_t = p(v_t) : u < v$
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

variables between v and u because they will have the same parent considering tree t . We do this with the following constraints:

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

These clauses are listed in Sect. 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 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 the direction of u matches the direction specified in v . These constraints are presented in Sect. 2 of Table 3.

In cases when the direction of u does not match direction specified in v we should not use it. See Sect. 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 Sect. 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 Sect. 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

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

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

$O(tn^3)$ 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 CryptoMiniSat². 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.

² <http://www.msos.org/cryptominisat4/>.

4 Experiments

To test the performance of our algorithm we evaluated PhyloSAT on a grass (Poaceae) dataset provided by the 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 1090 T 3.2 GHz processor on Ubuntu 14.04. All tests were run with a time limit of 1000s. 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

Table 4. Experimental results.

Algorithm	Solved cases	Optimal solutions
PhyloSAT	48	39
PIRN_{C}	29	29
PIRN_{CH}	43	36

Table 5. Comparison of results of PIRN_{CH} and PhyloSAT on test cases with big hybridisation number. Runtimes in seconds are reported in brackets.

Test instance	PhyloSAT	PIRN_{CH}	Optimal solution
2NdhfPhyt	6 (9)	6 (6)	6
3NdhfPhytRpoc	8 (1000)	8 (28)	6
3PhytRbclRpoc	6 (11)	6 (3)	6
3RbclWaxyIts	6 (1000)	7 (4)	6
4NdhfRbclWaxyIts	7 (1000)	7 (35)	≥ 6
4PhytRbclRpocIts	9 (1000)	8 (377)	≥ 6
2RbclRpoc	7 (1000)	7 (42)	7
3NdhfWaxyIts	8 (1000)	8 (90)	≥ 7
3PhytRbclIts	11 (1000)	8 (120)	≥ 7
3PhytRpocIts	7 (1000)	7 (59)	7
4NdhfPhytRbclRpoc	10 (1000)	10 (287)	≥ 7
4NdhfPhytRpocIts	10 (1000)	-	≥ 7
2NdhfPhyt	8 (12)	-	8
2NdhfRbcl	8 (1)	8 (851)	8
2PhytIts	8 (41)	8 (372)	8
3NdhfPhytRbcl	9 (123)	-	9
2NdhfRpoc	9 (954)	9 (484)	9
3NdhfRbclRpoc	13 (1000)	-	≥ 10
3NdhfPhytIts	13 (1000)	-	≥ 11

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. In 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 3 cases but was unable to prove their optimality. 8 more test cases had isomorphic trees in input, so they had an obvious answer of zero. Results of the simulation are summarized in Table 4. Also, the more precise results of comparison are in Table 5. We included only cases with hybridization number more than 6, because on less complicated tests results of all the algorithms were similar. The results of PIRN_C are not included because it did not manage to solve any of these cases.

Experiments showed that in some cases PhyloSAT is able to find an optimal network but then it spends considerable time trying to prove its optimality 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 much 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 useful and will save much computation time.

5 Conclusion

We have proposed an algorithm for constructing an exact parsimonious hybridization network from multiple phylogenetic trees. 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, Daniil Chivilikhin and Fedor Tsarev for helpful comments and conversations.

References

1. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. *Adv. Comput.* **58**, 117–148 (2003)

2. Bonet, M.L., John, K.S.: Efficiently calculating evolutionary tree measures using SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 4–17. Springer, Heidelberg (2009)
3. Bordewich, M., Semple, C.: Computing the minimum number of hybridization events for a consistent evolutionary history. *Discret. Appl. Math.* **155**(8), 914–928 (2007)
4. Chen, Z.Z., Wang, L.: Hybridnet: a tool for constructing hybridization networks. *Bioinform.* **26**(22), 2912–2913 (2010)
5. Heule, M.J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 66–79. Springer, Heidelberg (2010)
6. Hölldobler, S., Nguyen, V.: An efficient encoding of the at-most-one constraint. Technical report, KRR Group 2013–04, Technische Universität Dresden, 01062 Dresden, Germany (2013)
7. Huson, D.H., Rupp, R., Scornavacca, C.: *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, New York (2010)
8. Morrison, D.A.: *Introduction to Phylogenetic Networks*. RJR Productions, Uppsala (2011)
9. Nakhleh, L.: Evolutionary phylogenetic networks: models and issues. In: Heath, L.S., Ramakrishnan, H. (eds.) *Problem Solving Handbook in Computational Biology and Bioinformatics*, pp. 125–158. Springer, Berlin (2011)
10. Nakhleh, L., Ruths, D., Wang, L.-S.: RIATA-HGT: a fast and accurate heuristic for reconstructing horizontal gene transfer. In: Wang, L. (ed.) *COCOON 2005*. LNCS, vol. 3595, pp. 84–93. Springer, Heidelberg (2005)
11. Park, H.J., Nakhleh, L.: MURPAR: a fast heuristic for inferring parsimonious phylogenetic networks from multiple gene trees. In: Bleris, L., Măndoiu, I., Schwartz, R., Wang, J. (eds.) *ISBRA 2012*. LNCS, vol. 7292, pp. 213–224. Springer, Heidelberg (2012)
12. Semple, C.: *Hybridization Networks*. Department of Mathematics and Statistics, University of Canterbury, New York (2006)
13. Wu, Y.: Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees. *Bioinformat.* **26**(12), i140–i148 (2010)
14. Wu, Y.: An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. *J. Comput. Biol.* **20**(10), 792–804 (2013)