# 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 algorithm for the minimal hybridization network construction problem which is based on reducing the problem to an instance of the Boolean satisfiability problem. We created a software tool called PhyloSAT which implements our algorithm and is available for download from https://github.com/ZumZoom/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:** Hybridization networks, Boolean satisfiability, SAT, bioinformatics, genetics

## 1 Introduction

A phylogenetic network is a powerful tool to model 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. All the gene trees have the same set of leaves called 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 $\text{PIRN}_\text{C}$ 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 convert the satisfying assignment into the hybridization network. The formula construction process should ensure that the network has the minimal possible number of reticulate nodes. Our approach leads to an exact algorithm that outperforms $\text{PIRN}_\text{C}$ on all the test cases.

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.

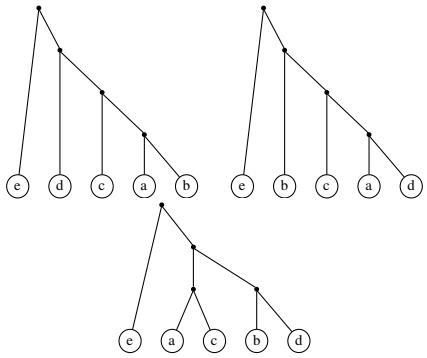
Let $d^-(v)$ be the in-degree of $v$ and $d^+(v)$ be the out-degree of $v$. A *hybridization network* on a set of taxa $X$ is a directed acyclic graph with a single root $\rho$ and leaves bijectively mapped to the set of taxa $X$. If $d^-(v) > 1$ then node $v$ is called a *reticulation node*. In this 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 $T'$.

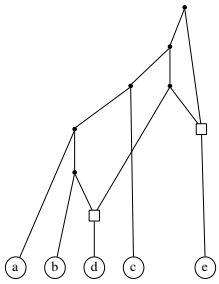
Now suppose we have a phylogenetic tree $T$ and a hybridization network $N$. We say that network $N$ *displays* $T$ if we can choose the edges of reticulation nodes in such a way that after edge-contraction the obtained tree $T'$ will be isomorphic to $T$. In Figure 1 each of the three trees is displayed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

A *hybridization number* of network $N$ with root $\rho$ is commonly defined as $h(N) = \sum_{v!=\rho} (d^-(v) - 1)$. 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 lowest hybridization number possible. Note that there can be several networks with equal hybridization number.




(a) Three gene trees




(b) Hybridisation 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 look 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. **Sub-tree reduction rule**: replace every sub-tree which is present in all input trees with a leaf with a new label.
2. **Cluster reduction rule**: for each cluster $A$ replace the sub-trees containing it with a leaf with a new label and add a new task for processing which consists of deleted sub-trees $T'_1, T'_2, ...T'_k$ 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. We look through possible values of $k$ starting from the highest (?? why do we choose max $k$ equals to the size of taxa ??) and construct a boolean formula corresponding to the current $k$. We decrease $k$ until the solver can't satisfy the formula, and this means that the previous value of $k$ was the lowest possible. There are another 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 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 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 should check every possible answer to ensure that there is no solution which 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 $\text{PIRN}_{\text{CH}}$ [13], RIATA-HGT [9] and MURPAR [10]. We do not consider such optimisations in this paper.

### 3.3 Encoding the boolean formula

Having a set of trees $T$ over a set of taxa $N$ 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 numerate 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 numeration allows us to define the following sets of nodes for each node $v$ such that $PC(v)$ 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 literals 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 $v \in L \cup V/root, u \in PP(v)$
   $p_{v,u}$ is true iff $u$ is the parent of a regular node $v$.
3. $lp_{v,u}$ and $rp_{v,u}$ for each $v \in R, u \in PP(v)$
   $lp_{v,u}$ (or $rp_{v,u}$) is true iff $u$ is the left (right) parent of a reticulation node $v$.
4. $ch_{v,u}$ for each $v \in R, u \in PC(v)$
   $ch_{v,u}$ 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. We say that the node can have at least one parent and at most one parent. This can be expressed in the following way (for example for parents of regular node $v$):

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

Using this pattern, we add the uniqueness constraints for literals $l$, $r$, $p$, $lp$, $rp$ and $ch$. We also add constraints to order the numbers of children of regular nodes and parents of reticulation nodes.

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.

The last step of network construction is to deal with the numeration around reticulation nodes. To do this, we add constraints to fix relative numbers of children and parents of reticulation nodes.

The proposed constraints are listed in Table 1. Since as 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.

**Table 1.** Clauses for network structure encoding

| Clauses | Range |
|---|---|
| $p_{v,u_1} \vee ... \vee p_{v,u_k}$ | $v \in V, u_1...u_k \in PP(v)$ |
| $p_{v,u} \rightarrow \neg p_{v,w}$ | $v \in V, u, w \in PP(v)$ |
| $l_{v,u_1} \vee ... \vee l_{v,u_k}$ | $v \in V, u_1...u_k \in PC(v)$ |
| $l_{v,u} \rightarrow \neg l_{v,w}$ | $v \in V, u, w \in PC(v)$ |
| $r_{v,u_1} \vee ... \vee r_{v,u_k}$ | $v \in V, u_1...u_k \in PC(v)$ |
| $r_{v,u} \rightarrow \neg r_{v,w}$ | $v \in V, u, w \in PC(v)$ |
| $l_{v,u} \rightarrow \neg r_{v,w}$ | $v \in V, u, w \in PC(v) : u \geq w$ |
| $ch_{v,u_1} \vee ... \vee ch_{v,u_k}$ | $v \in R, u_1...u_k \in PC(v)$ |
| $ch_{v,u} \rightarrow \neg ch_{v,w}$ | $v \in R, u, w \in PC(v)$ |
| $lp_{v,u_1} \vee ... \vee lp_{v,u_k}$ | $v \in R, u_1...u_k \in PP(v)$ |
| $lp_{v,u} \rightarrow \neg lp_{v,w}$ | $v \in R, u, w \in PP(v)$ |
| $rp_{v,u_1} \vee ... \vee rp_{v,u_k}$ | $v \in R, u_1...u_k \in PP(v)$ |
| $rp_{v,u} \rightarrow \neg rp_{v,w}$ | $v \in R, u, w \in PP(v)$ |
| $lp_{v,u} \rightarrow \neg rp_{v,w}$ | $v \in R, u, w \in PP(v) : u \geq w$ |
| $l_{v,u} \rightarrow p_{u,v}$ | $v \in V, u \in V, u \in PC(v)$ |
| $r_{v,u} \rightarrow p_{u,v}$ | $v \in V, u \in V, u \in PC(v)$ |
| $p_{u,v} \rightarrow (l_{v,u} \vee r_{v,u})$ | $v \in V, u \in V, u \in PC(v)$ |
| $l_{v,u} \rightarrow (lp_{u,v} \vee rp_{u,v})$ | $v \in V, u \in R, u \in PC(v)$ |
| $r_{v,u} \rightarrow (lp_{u,v} \vee rp_{u,v})$ | $v \in V, u \in R, u \in PC(v)$ |
| $lp_{u,v} \rightarrow (l_{v,u} \vee r_{v,u})$ | $v \in V, u \in R, u \in PC(v)$ |
| $rp_{u,v} \rightarrow (l_{v,u} \vee r_{v,u})$ | $v \in V, u \in R, u \in PC(v)$ |
| $ch_{v,u} \rightarrow p_{u,v}$ | $v \in R, u \in V, u \in PC(v)$ |
| $p_{u,v} \rightarrow ch_{v,u}$ | $v \in R, u \in V, u \in PC(v)$ |
| $ch_{v,u} \rightarrow (lp_{u,v} \vee rp_{u,v})$ | $v \in R, u \in R, u \in PC(v)$ |
| $lp_{u,v} \rightarrow ch_{v,u}$ | $v \in R, u \in R, u \in PC(v)$ |
| $rp_{u,v} \rightarrow ch_{v,u}$ | $v \in R, u \in R, u \in PC(v)$ |
| $ch_{v,u} \rightarrow \neg lp_{v,w}$ | $v \in R, u \in PC(v), w \in PP(v) : w \leq u$ |
| $ch_{v,u} \rightarrow \neg rp_{v,w}$ | $v \in R, u \in PC(v), w \in PP(v) : w \leq u$ |

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

1. $x_{t,s,v}$ for each $t \in T, s \in V(t), v \in V$
   $x_{t,s,v}$ is true iff regular node $v$ represents node $s$ from tree $t$, i.e. $x$ literals represent bijective mapping of network nodes to tree nodes. 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. $dir_{t,v}$ for each $t \in T, v \in R$
   $dir_{t,v}$ is true iff left parent edge of reticulation node $v$ is used to display tree $t$ and it is false in case of right edge, i.e. it specifies direction of necessary parent to display current tree
3. $rused_{t,v}$ for each $t \in T, v \in R$
   $rused_{t,v}$ is true iff child of reticulation node $v$ is used to display tree $t$
4. $used_{t,v}$ for each $t \in T, v \in V$
   $used_{t,v}$ is true iff regular node $v$ is used to display tree $t$
5. $up_{t,v,u}$ for each $t \in T, v \in V, u \in PU(v)$
   $up_{t,v,u}$ is true iff regular node $u$ corresponds to the first node on the path to the root of the tree $t$ starting from $v$. Note that because of edge contraction node $v$ may not exist in tree $t$, but nevertheless it lies on the some path in the network that after contraction will become an edge in tree $t$, so $u$ will correspond to the end of that edge. We will still say that $u$ is parent of $v$ considering tree $t$ oblivious to the inexistence of $v$ in $t$.

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 $up$. Note that in case of $x$ literals we have restriction that at most one node from tree corresponds to the network node, and that at most one node from network corresponds to the tree node. Because of dummy roots we know that roots of input trees are bijectively mapped to the root of the network so we have $x_{t,root_t,root}$ set to true for every tree. Also consider an obvious observation that $x_{t,s,v}$ implies $used_{t,v}$.

Furthermore if we know that node $v$ is leaf and its parent $u$ corresponds to some node $tv$ from tree $t$, then we can conclude that $up_{t,v,u}$ is true i.e. $u$ is direct parent of $v$ in tree $t$. Same observation can be done for non-leaves, i.e. if we know that $v$ corresponds to $tv$ and other node $u$ corresponds to $tu$ and $tu$ is parent of $tv$ then $u$ is direct parent of $v$ considering tree $t$. On the other hand if we know that $v$ corresponds to $tv$ and we know that $u$ is direct parent of $v$ considering tree $t$, then $u$ should correspond to parent of $tv$. We also should take care about numeration so we add constraint that says that if node $v$ has greater number than node $u$ then their corresponding nodes from tree can not have reversed order of numbers, i.e. $tu > tv : x_{t,tv,v} \rightarrow \neg x_{t,tu,u}$.

Next we add constraints that connect child-parent relations from trees with indirect child-parent relations in the network. First of all if $u$ is a direct parent of node $v$ and $u$ is used to display tree $t$ then $u$ is also the parent of $v$ considering tree $t$. And vice versa if we know that $u$ is 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 $up$ variables between $v$ and $u$ because they will have the same parent considering tree $t$. We do this with the following constraints:

$$\forall t \in T, v \in V \cup L, u \in PP(v), u \in V, w \in PP(u)$$
$$(parent_{v,u} \wedge \neg used_{t,u} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$$
$$(parent_{v,u} \wedge \neg used_{t,u} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$$

We also add clauses to forbid incorrect numeration, if node $v$ has reticulation parent $u$ then there can not exist node $w$ that its number is less than number of $v$ and $up_{t,u,w}$ is true. And for all $w$ with numbers greater than $v$ we add clauses to share information of $up$ variables, we do this the same way as in previous observation.

Before adding similar constraints for child-parent relations of reticulation nodes we add some restrictions on $rused$ literals. First of all if for any reticulation node $v$ its child is regular node we should use $v$ for displaying tree. If its child is reticulation node then we shouldn't use node $v$ unless its child is used. We also should forbid usage of node $v$ when its direction as parent of $u$ differs from direction specified in $u$, and we should use node $v$ if its direction matches direction specified in $u$ and $u$ itself is used.

Now we consider reticulation node $v$ and its parent $u$. If $u$ is reticulation node then we should share information about their parent considering tree $t$ but only when direction of parent $u$ matches direction specified in $v$. If $u$ is regular node and $u$ is used for displaying then $u$ is direct parent of $v$ considering tree $t$. On the other hand if $u$ is not used then we should share information about their parent considering tree $t$ also only when direction of $u$ matches direction specified in $v$.

The next step formids meaningless values of $used$ literals. Consider reticulation node $v$ and one of its parents $u$. $u$ can not be used for displaying current tree if its direction differs from direction specified in $v$ or if $v$ is not used for displaying itself.

We finish tree mapping by adding heuristic constraints. Notice that number of node in network can not be less than the size of the sub-tree of corresponding node $tv$ in tree $t$ and also it can not be greater than size of the tree minus depth of $tv$. Also if node $tv_1$ in tree $t_1$ and node $tv_2$ in tree $t_2$ have disjoint sets of taxa in their subtrees then they can not be mapped to the same node in network.

Again the most expensive clauses are clauses that represent uniqueness of variables $up$ 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. Choosing the most appropriate solver is not considered in this paper,

**Table 2.** Clauses to map trees to network

| Clauses | Range |
|---|---|
| $up_{t,v,u_1} \vee ... \vee up_{t,v,u_k}$ | $v \in V \cup L \cup R, u_1...u_k \in PU(v)$ |
| $up_{t,v,u} \rightarrow \neg up_{t,v,w}$ | $v \in V \cup L \cup R, u, w \in PU(v)$ |
| $x_{t,t_v,v_1} \vee ... \vee x_{t,t_v,v_k}$ | $t \in T, t_v \in V(t), v_1...v_k \in V$ |
| $x_{t,t_v,v} \rightarrow \neg x_{t,t_v,w}$ | $t \in T, t_v \in V(t), v, w \in V$ |
| $x_{t,t_v,v} \rightarrow \neg x_{t,t_w,v}$ | $t \in T, t_v, t_w \in V(t), v \in V$ |
| $x_{t,tv,v} \rightarrow used_{t,v}$ | $t \in T, v \in V, tv \in V(t)$ |
| $x_{t,root_t,root}$ | $t \in T, root_t = root(t)$ |
| $x_{t,tu,u} \rightarrow up_{t,v,u}$ | $t \in T, v \in L, u \in PP(v), tu \in V(t)$ |
| $(x_{t,tv,v} \wedge x_{t,tu,u}) \rightarrow up_{t,v,u}$ | $t \in T, v \in V, u \in PP(v), tv \in V(t), tu = p(tv)$ |
| $(x_{t,tv,v} \wedge up_{t,v,u}) \rightarrow x_{t,tu,u}$ | $t \in T, v \in V, u \in PP(v), tv \in V(t), tu = p(tv)$ |
| $x_{t,tv,v} \rightarrow \neg x_{t,tu,u}$ | $t \in T, v \in V, u \in V, tv \in V(t), tu = p(tv), u < v$ |
| $(parent_{v,u} \wedge used_{t,u}) \rightarrow up_{t,v,u}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in V$ |
| $(parent_{v,u} \wedge up_{t,v,u}) \rightarrow used_{t,u}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in V$ |
| $(parent_{v,u} \wedge \neg used_{t,u} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in V, w \in PP(u)$ |
| $(parent_{v,u} \wedge \neg used_{t,u} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in V, w \in PP(u)$ |
| $parent_{v,u} \rightarrow \neg up_{t,u,w}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in R, w \in PU(u), w \leq v$ |
| $(parent_{v,u} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in R, w \in PU(u), w > v$ |
| $(parent_{v,u} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$ | $t \in T, v \in V \cup L, u \in PP(v), u \in R, w \in PU(u), w > v$ |
| $ch_{v,u} \rightarrow rused_{t,v}$ | $t \in T, v \in R, u \in PC(v), u \in V \cup L$ |
| $\neg rused_{t,u} \rightarrow \neg rused_{t,v}$ | $t \in T, v \in R, u \in PC(v), u \in R$ |
| $(lp_{t,u,v} \wedge \neg dir_{t,u}) \rightarrow \neg rused_{t,v}$ | $t \in T, v \in R, u \in PC(v), u \in R$ |
| $(rp_{t,u,v} \wedge dir_{t,u}) \rightarrow \neg rused_{t,v}$ | $t \in T, v \in R, u \in PC(v), u \in R$ |
| $(lp_{t,u,v} \wedge dir_{t,u} \wedge rused_{t,u}) \rightarrow rused_{t,v}$ | $t \in T, v \in R, u \in PC(v), u \in R$ |
| $(rp_{t,u,v} \wedge \neg dir_{t,u} \wedge rused_{t,u}) \rightarrow rused_{t,v}$ | $t \in T, v \in R, u \in PC(v), u \in R$ |
| $(lp_{v,u} \wedge dir_{t,v} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$ | $t \in T, v \in R, u \in PP(v), u \in R, w \in PU(u)$ |
| $(lp_{v,u} \wedge dir_{t,v} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$ | $t \in T, v \in R, u \in PP(v), u \in R, w \in PU(u)$ |
| $(rp_{v,u} \wedge \neg dir_{t,v} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$ | $t \in T, v \in R, u \in PP(v), u \in R, w \in PU(u)$ |
| $(rp_{v,u} \wedge \neg dir_{t,v} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$ | $t \in T, v \in R, u \in PP(v), u \in R, w \in PU(u)$ |
| $(lp_{v,u} \wedge dir_{t,v} \wedge used_{t,u}) \rightarrow up_{t,v,u}$ | $t \in T, v \in R, u \in PP(v), u \in V$ |
| $(rp_{v,u} \wedge \neg dir_{t,v} \wedge used_{t,u}) \rightarrow up_{t,v,u}$ | $t \in T, v \in R, u \in PP(v), u \in V$ |
| $(lp_{v,u} \wedge dir_{t,v} \wedge \neg used_{t,u} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$ | $t \in T, v \in R, u \in PP(v), u \in V, w \in PU(u)$ |
| $(lp_{v,u} \wedge dir_{t,v} \wedge \neg used_{t,u} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$ | $t \in T, v \in R, u \in PP(v), u \in V, w \in PU(u)$ |
| $(rp_{v,u} \wedge \neg dir_{t,v} \wedge \neg used_{t,u} \wedge up_{t,u,w}) \rightarrow up_{t,v,w}$ | $t \in T, v \in R, u \in PP(v), u \in V, w \in PU(u)$ |
| $(rp_{v,u} \wedge \neg dir_{t,v} \wedge \neg used_{t,u} \wedge up_{t,v,w}) \rightarrow up_{t,u,w}$ | $t \in T, v \in R, u \in PP(v), u \in V, w \in PU(u)$ |
| $(\neg dir_{t,v} \wedge lp_{v,u}) \rightarrow \neg used_{t,u}$ | $t \in T, v \in R, u \in PP(v), u \in V$ |
| $(dir_{t,v} \wedge rp_{v,u}) \rightarrow \neg used_{t,u}$ | $t \in T, v \in R, u \in PP(v), u \in V$ |
| $(lp_{v,u} \wedge \neg rused_{t,v}) \rightarrow \neg used_{t,u}$ | $t \in T, v \in R, u \in PP(v), u \in V$ |
| $(rp_{v,u} \wedge \neg rused_{t,v}) \rightarrow \neg used_{t,u}$ | $t \in T, v \in R, u \in PP(v), u \in V$ |
| $\neg x_{t,tv,v}$ | $t \in T, v \in V, tv \in V(t), tv < size(subtree(tv))$ |
| $\neg x_{t,tv,v}$ | $t \in T, v \in V, tv \in V(t), tv > size(t) - depth(tv)$ |
| $\neg x_{t_1,tv_1,v} \vee \neg x_{t_2,tv_2,v}$ | $t_1, t_2 \in T, v \in V, tv_1 \in V(t_1), tv_2 \in V(t_2)$ and subtrees of $t1$ and $t2$ have disjoint sets of taxa |

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 recover 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 the single hybridization network corresponding to the original task.

## 4 Experiments

To test the performance of our algorithm we evaluated it 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 the time limit of 1000 seconds. For comparison we also ran PIRN$_C$ and PIRN$_{CH}$ on the same test cases.

9 out of 57 test cases were not solved even by heuristic algorithms in time. Our algorithm was able to produce 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. On all the test cases running time of PIRN$_C$ was worse than running time of our algorithm. Even PIRN$_{CH}$ did not solve 2 of these 28 test cases in time and its running time was better than running time of our algorithm only on 5 test cases and significantly worse on 2 test cases. Twelve more test cases were not solved by PIRN$_C$ and our algorithm did not complete computation in time, but 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 our algorithm, 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 our algorithm 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 and we excluded them from consideration.

Experiments showed that in some cases our algorithm is able to find an optimal network but then it spends a lot of time trying to find the network with 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 heuristic lower bound is equal to the actual minimal hybridisation number. Besides we found that it also costs a lot of time to build a network with a high hybridization number when the minimal hybridization number is low. Thus close upper bound of minimal hybridization number will also be very useful and will save lots of computational time.

## 5 Conclusion

We have proposed an algorithm for construction an exact parsimonious hybridization network from multiply trees. Experiments have shown that our method

outperforms the PIRN$_C$ algorithm in all cases and performs reasonably well comparing to the heuristic PIRN$_{CH}$. However in cases of high hybridization numbers search and construction of optimal network is still a very challenging problem. In future we plan to use existing heuristics and estimations on lower and upper bounds on hybridization number to limit searching bounds and thus reduce running time of our algorithm.

## Acknowledgements

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)