Constructing Parsimonious Hybridization Networks with Multiple Phylogenetic Trees using SAT-solver

Vladimir Ulyantsev and Mikhail Melnik

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

Abstract. We present an algorithm for the minimum hybridisation network construction problem that is based on reducing the problem to an instance of boolean satisfiability problem. We created a software tool called PhyloSAT that implements our algorithm and is available for download from https://github.com/ZumZoom/PhyloSAT. Experimental evaluation of our algorithm on biological data shows that our method is as far as we know the fastest exact algorithm for the minimum hybridisation network construction.

Keywords: Hybridisation networks, Boolean satisfiability, TODO

1 Introduction

Phylogenetic network is a powerful tool to model the reticulate evolutionary processes (such as horizontal gene transfer and hybrid specification). Briefly, 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 [5], [6], [7]. 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 [9], [3].

As an 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. Because of the 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], [8] and they usually deal with only two trees. However, recently an exact algorithm PIRN$_C$ has been introduced [11] 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 multiply input trees based on satisfiability (SAT)

solvers. SAT problem is known to be NP-complete [2], but state-of-the-art SAT-solvers running on the 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 [1] as well as to solve problems in other domains such as finite-state machine induction [4], software verification [].

The general outline of our approach is to convert an instance of hybridization network construction problem to an instance of SAT problem (a Boolean formula), solve it using 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 PIRN$_{CH}$ 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.

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, i.e. in-degrees of all nodes, except root, are one, and out-degrees are zero for leaves and two for internal nodes.

A *hybridisation network* is a directed acyclic graph with a single root and leaves bijectively labelled by the set of taxa. If the in-degree of a node is greater than one, such node is called *reticulation node*. In this paper we assume that reticulation nodes have in-degree of two and out-degree of one. We can do this assumption by noting that we can convert a reticulation node with in-degree of three or more to the sequence of reticulation nodes of in-degree two [10]. Other nodes are regular tree nodes and have the same properties as in tree.

Consider the hybridisation network N. Firstly we keep only one of the incoming edges for each reticulation node in N. Secondly we contract edges to remove all the nodes of in-degree one and out-degree one. With this two steps we obtain a tree T' from the network N.

Now suppose we have a phylogenetic tree T and a hybridisation 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 obtained tree T' will be equivalent to T.

A *hybridisation number* is commonly defined as the sum of in-degrees of all edges minus one. Note that under our assumptions $h(N)$ simply equals the number of reticulation nodes.

$$h(N) = \sum_{v != root} (d^-(v) - 1)$$

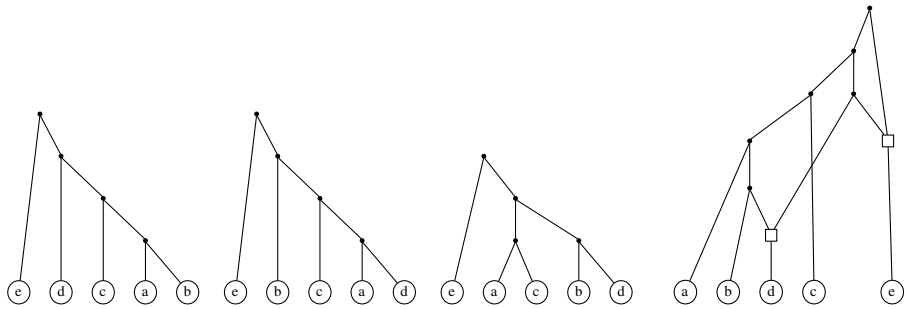

Fig. 1. An illustration of a hybridisation network for three trees with two reticulation nodes.

Suppose we are given a set of K phylogenetic trees $T_1, T_2, ...T_k$ over the same set of taxa. The *minimum hybridisation network* for that set of trees is a network N_{min} that displays each tree and has the lowest hybridisation number possible.

The most parsimonious hybridisation network problem is the following:

Given a set of phylogenetic trees $T_1, T_2, ...T_k$ over the same set of taxa, construct the minimum hybridisation network for that set of trees.

It has been shown that even for the case of $k = 2$ construction of such a network is NP-complete problem [2]. As far as we know there exists only one algorithm for the construction of the most parsimonious hybridisation network [11].

3 Algorithm

The main idea of the algorithm is to look over possible values of hybridisation number and to construct and solve a boolean formula that represents a hybridisation network with fixed hybridisation number. Before doing this search we do some pre-processing and consequently some post-processing after the network is found.

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 complexity of the problem. We do this according to the rules from [1]. To define those 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 that has all the taxa from A in its descendants.

1. **Sub-tree reduction rule**
   Replace any sub-tree that appears identically in all input trees by a leaf with new label.

2. **Cluster reduction rule**
   For each cluster A replace the sub-trees containing it by a leaf with new label and add a new task for processing that consists of deleted sub-trees $T'_1, T'_2, ...T'_k$ with leaf set A.

After we split the task into a set of less complex tasks, we add a dummy root to each tree of each task along with the new dummy leaf for tree consistency. This is done to ensure that all the trees in the input share 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 hybridisation number

To solve a subtask we need to find the lowest hybridisation number k such that there exists a hybridisation network with this hybridisation 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 that k. We decrease k until the solver can't satisfy the formula, and that 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 k until the solver will be able to satisfy the formula, or we can use binary search. But in our experiments the most time-consuming computations were with the value of $k_{optimal} - 1$ and consequently results of upwards method were very poor. Binary search results were close to downwards method, but in some cases when binary search tried to satisfy formulas with values of k less than $k_{optimal}$ its results also were poor. This can be explained by an observation that it is much easier for solver to produce an answer if formula is satisfiable than if it is not. In case of unsatisfiable formula solver should check every possible answer to ensure that there is no solution. Obvious method for reducing searching time is to use different heuristics for finding close starting upper and lower bounds of hybridisation number. Possible candidates are PIRN$_{CH}$, RIATA-HGT and MURPAR. We do not consider such optimisations in this paper.

3.3 Encoding boolean formula

Having a set of trees T over a set of taxa N and fixed hybridisation number k we will construct boolean formula that will be satisfiable iff there exists a hybridisation network that displays each tree in T and its hybridisation number equals to k. To do this we first notice that a network over the set of taxa of size n with hybridisation number k will have $2 * (n + k) - 1$ nodes. As we add dummy root and 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 all leaves have numbers in range $[0, n]$, all 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 leave or regular node is less than number of its parent. This is done to avoid consideration of isomoprhic networks during SAT solving. Such numeration 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 nodes that can be indirect parents 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 boolean formula.

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

1. $\forall v \in V, u \in PC(v) : l_{v,u}(r_{v,u})$
   $l(r)$ is true iff regular node v has node u as left (right) child.
2. $\forall v \in L \cup V/root, u \in PP(v) : p_{v,u}$
   p is true iff u is parent of regular node v.
3. $\forall v \in R, u \in PP(v) : lp_{v,u}(rp_{v,u})$
   $lp(rp)$ is true iff u is left (right) parent of reticulation node v.
4. $\forall v \in R, u \in PC(v) : ch_{v,u}$
   ch is true iff u is child of reticulation node v.

This takes $O((n+k)^2)$ literals for specifying 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 (child or something) and at most one parent. This can be expressed in the following way (for example for parents of regular node v):
$(\bigvee_{u \in PP(v)} p_{v,u}) \wedge (\bigwedge_{i,j \in PP(v); i<j} (p_{v,i} \rightarrow \neg p_{v,j}))$.

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

Network will be consistent if for every pair of nodes parent relation implies child relaton and vice versa. So we add constraints that connect parent literals with children literals for all the types of nodes.

And the last step in network construction is to care about 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. Thus as we need $O(n^2)$ clauses for each node to represent uniquness of its parents and children and $O(n)$ clauses for each node to represent parents-children relation, we finally get $O(n^3)$ clauses in total to represent network structure.

Mapping trees to network To show that network should contain all the input trees we add literals that represent mapping of tree nodes to the network nodes:

Table 1. Clauses for network structure encoding

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

1. $\forall t \in T, s \in V(t), v \in V : x_{t,s,v}$
   x 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
2. $\forall t \in T, v \in R : dir_{t,v}$
   dir 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. $\forall t \in T, v \in R : rused_{t,v}$
   $rused$ is true iff child of reticulation node v is used to display tree t
4. $\forall t \in T, v \in V : used_{t,v}$
   $used$ is true iff regular node v is used to display tree t
5. $\forall t \in T, v \in V, u \in PU(v) : up_{t,v,u}$
   up 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.

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.

Thus we have extra $O(tn^2)$ literals to match input trees to the network. So 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 if $x_{t,s,v}$ is true then $used_{t,v}$ should be also true.

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 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 restrinctions 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 considreing 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.

Next step is forbidding some meaningless values of $used$ literals. Consider reticulation node v and one of its parents u. u can not be used for displlaying 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 uniquness 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 boolean formula

To solve generated boolean formula we use a SAT-solver CryptoMiniSat 4.2.0. Choosing the most appropriate solver is not considered in current paper, however several experimentations were made by M. Bonet and K. John [1] so it is one of the topics of further research.

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

3.5 Post-processing

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

4 Experiments

To test performance of our algorithm we evaluated it on a grass (Poaceae) dataset provided by the Grass Phylogeny Working Group (Grass Phylogeny Working Group, 2001). 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 time limit equal to 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 heurstic algorithms in time. Our algorithm was able to produce optimal answer for 28 tests. From these 28 test cases PIRN$_C$ was able to solve only 21 and in all of them hybridisation number was less than 6. On all the tests running time of PIRN$_C$ was worse than running time of our algorithm. Even PIRN$_{CH}$ did not solve 2 of these 28 tests in time and its running time was better than running time of our algorithm only on 5 tests and significantly worse on 2 tests. 12 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 sometimes our algorithm is able to find an optimal network but then it spends a bunch of time trying to find network with hybridisation number one less than optimal and that time is higher than reasonable limit. Sometimes we can avoid such behaviour by first finding close lower bound on hybridisation number which will allow not to waste time on useless computation in some cases. Also we found that it also costs a lot of time to build a network with a very high hybridisation number when minimum hybridisation number is not so high. Thus upper bounds on hybridisation number will also be very useful and will save lots of computational time.

5 Discussion

We proposed an algorithm for construction an exact parsimonious hybridisation network from multiply trees. Experiments show that our method outperforms PIRN$_{CH}$ algorithm in all cases and perform reasonably well comparing

to heuristic PIRN$_C$. However in cases of high hybridisation numbers search and construction of optimal network is still very challenging problem. In future we plan to use existing heuristics and estimations on lower and upper bounds on hybridisation number to limit searching bounds and reduce running time of our algorithm.

Acknowledgements

References

1. 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)
2. Bordewich, M., Semple, C.: Computing the minimum number of hybridization events for a consistent evolutionary history. Discrete Applied Mathematics 155(8), 914–928 (2007)
3. Chen, Z.Z., Wang, L.: Hybridnet: a tool for constructing hybridization networks. Bioinformatics 26(22), 2912–2913 (2010)
4. Heule, M.J., Verwer, S.: Exact dfa identification using sat solvers. In: Grammatical Inference: Theoretical Results and Applications, pp. 66–79. Springer (2010)
5. Huson, D.H., Rupp, R., Scornavacca, C.: Phylogenetic networks: concepts, algorithms and applications. Cambridge University Press (2010)
6. Morrison, D.A.: Introduction to phylogenetic networks. RJR Productions (2011)
7. Nakhleh, L.: Evolutionary phylogenetic networks: models and issues. In: Problem solving handbook in computational biology and bioinformatics, pp. 125–158. Springer (2011)
8. 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)
9. Semple, C.: Hybridization networks. Department of Mathematics and Statistics, University of Canterbury (2006)
10. Wu, Y.: Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees. Bioinformatics 26(12), i140–i148 (2010)
11. Wu, Y.: An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. Journal of Computational Biology 20(10), 792–804 (2013)