

# Constructing a Smallest Refining Galled Phylogenetic Network

Trinh N.D. Huynh, Jesper Jansson, Nguyen Bao Nguyen, and Wing-Kin Sung

School of Computing, National University of Singapore, 3 Science Drive 2,  
Singapore 117543. E-mail: {huynhngo, jansson, nguyenba, ksung}@comp.nus.edu.sg

**Abstract.** Reticulation events occur frequently in many types of species. Therefore, to develop accurate methods for reconstructing phylogenetic networks in order to describe evolutionary history in the presence of reticulation events is important. Previous work has suggested that constructing phylogenetic networks by merging gene trees is a biologically meaningful approach. This paper presents two new efficient algorithms for inferring a phylogenetic network from a set  $\mathcal{T}$  of gene trees of arbitrary degrees. The first algorithm solves the open problem of constructing a refining galled network for  $\mathcal{T}$  (if one exists) with no restriction on the number of hybrid nodes; in fact, it outputs the smallest possible solution. In comparison, the previously best method (SpNet) can only construct networks having a single hybrid node. For cases where there exists no refining galled network for  $\mathcal{T}$ , our second algorithm identifies a minimum subset of the species set to be removed so that the resulting trees can be combined into a galled network. Based on our two algorithms, we propose two general methods named RGNet and RGNet+. Through simulations, we show that our methods outperform the other existing methods neighbor-joining, NeighborNet, and SpNet.

## 1 Introduction

A phylogenetic network is a generalization of a phylogenetic tree that allows internal nodes to have more than one parent. Phylogenetic networks are used to describe the evolutionary history of species when traditional tree-based models are known to be insufficient due to the occurrence of reticulation events such as hybrid speciation or horizontal gene transfer [2, 6, 8, 9, 11] that tend to occur frequently in certain types of organisms [6, 9]. Phylogenetic networks are also used in order to visualize several conflicting phylogenetic trees at the same time to represent ambiguity and to make it easier to identify parts of the trees that agree [1, 3, 4], which is helpful because different trees constructed from different datasets often contain parts that contradict each other and because many tree construction methods (e.g., bootstrapping) produce collections of trees rather than a single tree. Hence, development of reliable and efficient methods for constructing phylogenetic networks is crucial in the study of phylogenetics.

Several phylogenetic network reconstruction methods have been proposed recently. The different methods use different types of input data. Bryant and

Moulton [1] proposed a method called NeighborNet to construct a phylogenetic network from a given distance matrix for the species. Gusfield *et al.* [2] and Wang *et al.* [11] showed how to construct a *galled* phylogenetic network (defined below) given character-based data. Jansson, Nguyen and Sung [5] considered how to infer a phylogenetic network which is consistent with a given set of rooted triplets. Huson *et al.* [4] and Nakhleh *et al.* [9] proposed to infer phylogenetic networks and galled phylogenetic networks, respectively, by combining a given set of gene trees, obtained, e.g., via applying maximum likelihood to sequence data. This paper follows the approach of Huson *et al.* and Nakhleh *et al.* since combining gene trees into a phylogenetic network is a promising direction. In addition, this approach is biologically sound, as stated below.

Maddison [7] observed that if a phylogenetic network for a set of species contains a single hybrid node (i.e., a node with indegree greater than one) then each gene present in all of the species must evolve according to one of the two trees embedded in the network. (Maddison also described how to construct such a phylogenetic network from two given gene trees and hypothesized that this method can be extended to construct phylogenetic networks containing more than just one hybrid node.) Based on this observation, Nakhleh, Warnow, and Linder [9] considered the following general approach to reconstructing a phylogenetic network for a set  $L$  of species from two given gene datasets for  $L$ :

For each of the two gene datasets, infer a gene tree; next, if the two trees are identical then return that tree, else find a phylogenetic network with as few hybrid nodes as possible that contains both trees.

In particular, Nakhleh *et al.* proposed two efficient algorithms for reconstructing a structurally restricted phylogenetic network from two given gene trees, corresponding to the last step above.

The first algorithm of Nakhleh *et al.* [9] constructs a galled phylogenetic network, if one exists, having the minimum number  $m$  of hybrid nodes that induces the two given binary phylogenetic trees on a leaf set  $L$ . It runs in  $O(mn)$  time, where  $n = |L|$ . The algorithm does not work when there exists no galled network which induces both trees in the input, for example due to errors in the estimated gene trees. The second algorithm of Nakhleh *et al.* [9] is designed to handle this issue. It assumes that the input is two (not necessarily binary) phylogenetic trees  $t_1$  and  $t_2$ , obtained by first inferring a set of “good” trees for each of the two gene datasets (e.g., by using maximum parsimony or maximum likelihood) and then taking the strict consensus of each set. The algorithm outputs in  $O(n)$  time a galled phylogenetic network  $N$  with a single hybrid node (if one exists) that *refines*  $t_1$  and  $t_2$ , meaning that  $N$  contains as induced trees two binary phylogenetic trees  $T_1$  and  $T_2$  such that  $t_1$  and  $t_2$  are contractions of  $T_1$  and  $T_2$ .

The method SpNet (short for “Species Network”) proposed in [9] is a method for reconstructing phylogenetic networks from sequence datasets that uses maximum likelihood to infer the gene trees and then applies the algorithm above. The simulations studies in [9] indicate that SpNet performs very well compared to other existing methods (e.g., neighbor-joining [10] and NeighborNet [1]). However, their algorithm which SpNet is based on can only construct a phylogenetic

network with *one* hybrid node (if such a network exists), which is a severe restriction. The case that is more likely to occur in practice, i.e., to construct a galled phylogenetic network having more than one hybrid node, was left as an important open problem.

We present a simple and efficient algorithm that solves the main open problem of Nakhleh *et al.* [9].<sup>1</sup> It takes as input two phylogenetic trees of arbitrary degree with the same leaf set, and outputs a galled phylogenetic network (if one exists) which refines both of them. In fact, whenever such a network exists, our algorithm returns a refining galled phylogenetic network having the minimum possible number of hybrid nodes. Moreover, our algorithm can easily be extended to more than two input trees, whereas both algorithms of Nakhleh *et al.* will only work for exactly two input trees. We term the corresponding computational problem *the smallest refining galled network (SRGN) problem*. We also give an algorithm for situations where there exists no refining galled phylogenetic network for the given set of trees and we instead need to identify a largest possible subset  $L'$  of the leaf set such that if the input trees are topologically restricted to leaves in  $L'$ , then they can be combined into a refining galled network (for example by using our algorithm for the SRGN problem mentioned above). We call this new problem *the maximum galled network-compatibility (MGNC) problem*. Our algorithm for the MGNC problem runs in polynomial time as long as the number of input trees is bounded by a constant and their maximum degree is at most logarithmic in the number of leaves. Finally, we apply our two proposed main algorithms for the SRGN problem and the MGNC problem to obtain two general methods for inferring galled networks from gene sequence datasets which we name RGNNet and RGNNet+. We show by practical simulation studies that RGNNet outperforms the other existing methods neighbor-joining, NeighborNet, and SpNet, while RGNNet+ can be used even when the data does not fit into a galled network.

## 1.1 Problem definitions

A *phylogenetic tree* is a rooted, unordered tree whose leaves are distinctly labeled. A *phylogenetic network* is a generalization of a binary phylogenetic tree formally defined as a rooted, connected, directed acyclic graph in which: (1) exactly one node has indegree 0 (the *root*), and all other nodes have indegree 1 or 2; (2) all nodes with indegree 2 (referred to as *hybrid nodes*) have outdegree 1, and all other nodes have outdegree 0 or 2; and (3) all nodes with outdegree 0 (the *leaves*) are distinctly labeled. For any phylogenetic network  $N$ , let  $rn(N)$  be the

---

<sup>1</sup> An alternative approach to solving this problem may be by constructing the set  $\mathcal{R}$  of all rooted triplets that are consistent with at least one of the two given trees  $T_1$  and  $T_2$  and then applying the algorithm of [5]. However, note that the algorithm in [5] requires at least one rooted triplet to be specified for each  $\{a, b, c\} \subseteq L$ , so this method might not work when  $T_1$  and  $T_2$  are non-binary. Furthermore, the running time would be  $O(kn^3)$  which is impractical for large  $n$ . Also note that an arbitrary galled network  $N$  which is consistent with  $\mathcal{R}$  does not always induce  $T_1$  and  $T_2$ , so extra care would need to be taken to ensure the correctness of this approach.

number of hybrid nodes in  $N$ . Next, let  $\mathcal{U}(N)$  be the undirected graph obtained from  $N$  by replacing each directed edge by an undirected edge.  $N$  is said to be a *galled phylogenetic network* (or *galled network*, for short) if all cycles in  $\mathcal{U}(N)$  are node-disjoint. Galled networks are an important type of phylogenetic networks suitable for describing evolutionary history when the frequency of reticulation events is moderate (see [2] for a discussion), and are also known in the literature as *topologies with independent recombination events* [11], *galled-trees* [2], *gt-networks* [9], and *level-1 phylogenetic networks* [5].

Let  $N$  be a phylogenetic network and let  $T$  be a phylogenetic tree.  $T$  is said to be an *induced tree of  $N$*  (symmetrically,  $N$  is said to *induce  $T$* ) if  $T$  can be obtained from  $N$  by deleting a set of edges and then, for every node with outdegree 1 and indegree less than 2, contracting its outgoing edge.

Let  $T$  and  $t$  be two (not necessarily binary) phylogenetic trees.  $t$  is called a *contraction* of  $T$  if  $t$  can be obtained from  $T$  by performing a series of edge contractions. In this case,  $T$  is also said to *refine  $t$* . A phylogenetic network  $N$  *refines* a phylogenetic tree  $t$  if  $N$  induces a binary phylogenetic tree  $T$  such that  $T$  refines  $t$ . Two phylogenetic trees  $t_1$  and  $t_2$  are called *tree-compatible* (or just *compatible*) if there exists a phylogenetic tree which refines both  $t_1$  and  $t_2$ , or *galled network-compatible* if there exists a galled network which refines  $t_1$  and  $t_2$ .

For any phylogenetic network  $N$  with a leaf set  $L$  and a subset  $L' \subseteq L$ , the *topological restriction* of  $N$  to  $L'$ , denoted by  $N|L'$ , is defined as the phylogenetic network obtained by first deleting all nodes which are not on any directed path from the root to a leaf in  $L'$  along with their incident edges, and then, for every node with outdegree 1 and indegree less than 2, contracting its outgoing edge (any resulting set of multiple edges between two nodes is replaced by a single edge). Given a set  $\mathcal{N}$  of phylogenetic networks with a leaf set  $L$  and a subset  $L' \subseteq L$ , we let  $\mathcal{N}|L'$  denote the set  $\{N|L' : N \in \mathcal{N}\}$ .

We define the *smallest refining galled network (SRGN) problem* as follows: Given a set  $\mathcal{T} = \{t_1, t_2, \dots, t_k\}$  of phylogenetic trees of arbitrary degree having a leaf set  $L$ , construct a galled phylogenetic network  $N$  with leaf set  $L$  (if one exists) that refines every  $t_i \in \mathcal{T}$  and minimizes  $rn(N)$ . The *maximum galled network-compatibility (MGNC) problem* is: Given a set  $\mathcal{T} = \{t_1, t_2, \dots, t_k\}$  of phylogenetic trees of arbitrary degree having a leaf set  $L$ , compute a maximum subset  $L'$  of the leaf set  $L$  such that  $\mathcal{T}|L'$  has a refining galled network. In the rest of the paper, we let  $n$  and  $k$  denote the cardinality of  $L$  and  $\mathcal{T}$ , respectively, in the problem definitions above and let  $d$  denote the maximum degree (i.e., the maximum number of children of any node) of all trees in  $\mathcal{T}$ .

## 1.2 Our contributions

In this paper, we first present a polynomial-time algorithm for the SRGN problem restricted to two input trees. Thus, we are able to solve the open problem posed in [9]. The running time of our new algorithm is  $O(n^2)$ . Next, we show how to extend our algorithm from 2 to  $k$  input trees to run in  $O(k^2n^2)$  time.

When a set of phylogenetic trees cannot be combined into a galled phylogenetic network which refines each of them, it is useful to remove as few leaves as

possible from the leaf set of the trees so that the resulting trees admit a solution. Therefore, we also consider the optimization problem MGNC. We give an algorithm that solves the MGNC problem in  $O(2^{3kd}n^{2k})$  time.

Based on our algorithm for the SRGN problem, we propose a new method for inferring a galled phylogenetic network from gene sequence datasets which we name RGNNet. We combine RGNNet with our algorithm for the MGNC problem to obtain an even more general method named RGNNet+, and demonstrate the usefulness of our methods by evaluating and comparing their accuracy to those of several existing methods through extensive simulation studies.

## 2 Terminology and notation

Let  $N$  be a phylogenetic network with a hybrid node  $h$ . Every ancestor  $s$  of  $h$  such that  $h$  can be reached using two disjoint directed paths starting at the children of  $s$  is called a *split node of  $h$* . If  $s$  is a split node of  $h$  then any path starting at  $s$  and ending at  $h$  is called a *merge path of  $h$* . From the above, it follows that in a galled network, each split node corresponds to exactly one hybrid node, and each hybrid node has exactly one split node.

Given a galled network  $N$ , we write  $A(N)$  to denote the set of leaf labels in  $N$  and  $s(N)$  to denote the set of children of the root of  $N$ . Given a node  $u$  in  $N$ , we write  $child(u)$  to denote the set of children of  $u$  and  $N[u]$  to denote the subnetwork of  $N$  rooted at  $u$ , i.e., the minimal subgraph of  $N$  which includes all nodes and directed edges of  $N$  reachable from  $u$ .  $N[u]$  is said to be *attached* to a merge path  $P$  in  $N$  if  $u$  does not belong to  $P$  but  $u$  is a child of a node belonging to  $P$ . If  $N'$  is a subnetwork of  $N$  then  $N \setminus N'$  is the network obtained by removing  $N'$  (and all incident edges) from  $N$ , and then, for every node with outdegree 1 and indegree less than 2, contracting its outgoing edge.

Given a tree  $T$  and a node  $v$  in  $T$ , for any nonempty subset  $A \subseteq child(v)$ , we write  $T[v, A]$  to denote the subtree obtained from  $T[v]$  by removing all leaves that are not reachable from any node in  $A$  and all incident edges. We call  $T[v, A]$  a *restricted subtree rooted at  $v$  in  $T$* . Finally,  $T[A]$  is short for  $T[v, A]$  if  $v$  is the root of  $N$ .

## 3 Solving the SRGN problem

This section solves the SRGN problem in  $O(k^2n^2)$  time. For explanation of the algorithm, we solve the problem when  $k = 2$ . Some technical lemmas which are needed to prove our main result can be found in Section 4 and Section 5.

**Definition 1.** *Given two trees  $T_1$  and  $T_2$  with leaf set  $L$ ,  $T_1$  and  $T_2$  admit a leaf set bipartition  $(X, Y)$  of  $L$  if, for  $i = 1, 2$ , there is a partition  $(A_i, B_i)$  of  $s(T_i)$  such that  $X = A(T_i[A_i])$  and  $Y = A(T_i[B_i])$ . (See Figure 3(b) for an example.)*

**Definition 2.** *Given two trees  $T_1$  and  $T_2$  with leaf set  $L$  and  $u$  be a node in  $T_2$ . We say that  $T_1$  is side compatible with  $(T_2, u)$  if either (1)  $u$  is the root of  $T_2$ ; or*

**Algorithm** *BuildGalledNetwork*

**Input:** Two trees  $T_1$  and  $T_2$  leaf-labeled by  $L$ .

**Output:** A SRGN  $N$  for  $T_1$  and  $T_2$ , if exists.

- 1 **if**  $T_1$  and  $T_2$  admits a leaf set bipartition  $(X, Y)$  **then**
- 1.1 Let  $N_X = \text{BuildGalledNetwork}(T_1|X, T_2|X)$  and  $N_Y = \text{BuildGalledNetwork}(T_1|Y, T_2|Y)$ .
- 1.2 If any of  $N_X$  and  $N_Y$  is null, return null.
- 1.3 Return  $N$  obtained by attaching both  $N_X$  and  $N_Y$  to a common root.
- elseif**  $T_1$  and  $T_2$  admits a leaf set tripartition  $(X, Y, Z)$  **then**
- 1.4 For  $i = 1, 2$ , let  $u_i$  be the root of  $T_i|Y$  in  $T_i$ .
- 1.5 Let  $N_X = \text{BuildSide}(T_1|X, T_2|X, u_2)$ ,  $N_Z = \text{BuildSide}(T_2|Z, T_1|Z, u_1)$ ,  $N_Y = \text{BuildGalledNetwork}(T_1|Y, T_2|Y)$ .
- 1.6 If any of  $N_X, N_Y, N_Z$  is null, return null.
- 1.7 Let  $N'$  be the network formed by attaching  $N_X$  and  $N_Z$  to a common root.
- 1.8 Let  $w_X$  and  $w_Z$  be the nodes in  $N'$ , where  $\Lambda(N'[w_X]) = \Lambda(T_2[u_2]) \cap X$  and  $\Lambda(N'[w_Z]) = \Lambda(T_1[u_1]) \cap Z$ , and let  $v_X$  and  $v_Z$  be the parent of  $w_X$  and  $w_Z$ , respectively.
- 1.9 For  $i = X, Z$ , create a new node  $q_i$  by subdividing the edge  $(v_i, w_i)$ .
- 1.10 Create a new node  $h$ , make  $h$  a child of both  $q_X$  and  $q_Z$ , and add an edge from  $h$  to the root of  $N_Y$ .
- 1.11 Let  $N$  be the resulting network and return  $N$ .
- endif**

**End** *BuildGalledNetwork*

**Fig. 1.** Framework for constructing phylogenetic network.

**Algorithm** *BuildSide*

**Input:** Two trees  $T_1$  and  $T_2$  and a node  $u$  in  $T_2$ , where  $T_1$  is side compatible with  $(T_2, u)$ .

**Output:** A SRGN  $N$  for  $T_1$  and  $T_2$ , where there is no split node on the path from the root of  $N$  to a node  $v$ , excluding  $v$ , where  $\Lambda(N[v]) = \Lambda(T_2[u])$ .

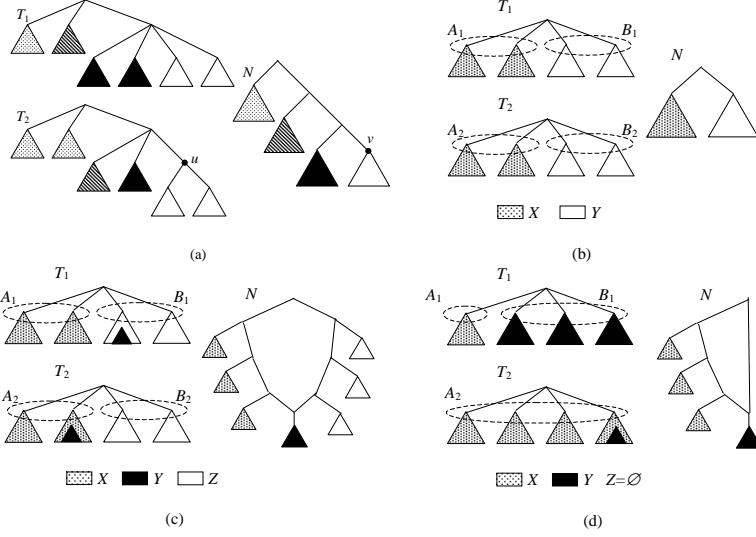
- 1 If  $u$  is the root of  $T_2$ , return  $\text{BuildGalledNetwork}(T_1, T_2)$ .
- 2 Find leaf set bipartition  $(X, Y)$  admitted by  $T_1$  and  $T_2$  where  $T_2|X$  contains  $u$ .
- 3 If there is no such partition, return null.
- 4 Let  $N_1 = \text{BuildSide}(T_1|X, T_2|X, u)$  and  $N_2 = \text{BuildGalledNetwork}(T_1|Y, T_2|Y)$ .
- 5 If any of  $N_1$  and  $N_2$  is null, return null.
- 6 Return  $N$  obtained by attaching both  $N_1$  and  $N_2$  to a common root.

**End** *BuildSide*

**Fig. 2.** Algorithm BuildSide.

(2) there is a restricted subtree  $t_1$  rooted at the root of  $T_1$  and a restricted subtree  $t_2$  rooted at the root of  $T_2$ , where  $t_2$  does not contain  $u$ , such that  $\Lambda(t_1) = \Lambda(t_2)$  and  $T_1 \setminus t_1$  is side compatible with  $(T_2 \setminus t_2, u)$ . (see Figure 3(a).)

Given that  $T_1$  is side compatible with  $(T_2, u)$ , we have the following properties.



**Fig. 3.** (a) The left figure shows an example of  $T_1$  and  $T_2$  such that  $T_1$  is side compatible with  $(T_2, u)$ . Sets of leaves with the same pattern in both trees are identical. Based on Lemma 2, if  $T_1$  and  $T_2$  are galled network-compatible, then there exists a SRGN  $N$  for  $T_1$  and  $T_2$  having no split node on the path from the root to the parent of  $v$ , where  $\Lambda(N[v]) = \Lambda(T_2[u])$ . (b)  $(X, Y)$  is a leaf set bipartition admitted by  $T_1$  and  $T_2$ . By Lemma 4, there exists a SRGN  $N$  for  $T_1$  and  $T_2$  that satisfies  $(X, Y)$ . (c)  $(X, Y, Z)$  is a leaf set tripartition admitted by  $T_1$  and  $T_2$ . Note that  $Y$  is the set of leaves of some restricted subtrees in both  $T_1$  and  $T_2$ . By Lemma 5, there exists a SRGN  $N$  for  $T_1$  and  $T_2$  that satisfies  $(X, Y, Z)$ . (d) This figure is similar to (c) except that  $Z = \emptyset$ .

**Lemma 1.** *Given that  $T_1$  is side compatible with  $(T_2, u)$ . Then  $T_1$  and  $T_2$  admit a leaf set bipartition. Furthermore, for any leaf set bipartition  $(X, Y)$  admitted by  $T_1$  and  $T_2$ , where  $T_2|X$  contains  $u$ ,  $T_1|X$  is side compatible with  $(T_2|X, u)$ .*

**Lemma 2.** *Given that  $T_1$  is side compatible with  $(T_2, u)$ . Suppose  $T_1$  and  $T_2$  are galled network-compatible, then there exists a SRGN  $N$  for  $T_1$  and  $T_2$ , where there is a node  $v$  in  $N$  such that  $\Lambda(N[v]) = \Lambda(T_2[u])$  and there is no split node on the path from the root of  $N$  to  $v$ , excluding  $v$ .*

*Proof.* An algorithm for building such an  $N$  is shown in Figure 2. That is  $N = \text{BuildSide}(T_1, T_2, u)$ . Its correctness comes from Lemma 1 and Lemma 7.  $\square$

**Definition 3.** *Given two trees  $T_1$  and  $T_2$  with leaf set  $L$ .  $T_1$  and  $T_2$  admit a leaf set tripartition of  $L$  into  $(X, Y, Z)$  if there exist a partition  $(A_1, B_1)$  of  $s(T_1)$ , a partition  $(A_2, B_2)$  of  $s(T_2)$ , a restricted subtree  $t_1$  rooted at some node  $u_1$  in  $T_1$ , where  $\Lambda(t_1) \subseteq \Lambda(T_1[B_1])$ , and a restricted subtree  $t_2$  rooted at some node  $u_2$  in  $T_2$ , where  $\Lambda(t_2) \subseteq \Lambda(T_2[A_2])$ , such that (1)  $Y = \Lambda(t_1) = \Lambda(t_2)$ ,  $X = \Lambda(T_1[A_1]) = \Lambda(T_2[A_2] \setminus t_2)$  and  $Z = \Lambda(T_1[B_1] \setminus t_1) = \Lambda(T_2[B_2])$ ; (2)  $T_1[A_1]$  is side compatible with  $(T_2[A_2] \setminus t_2, u_2)$ ; and (3)  $T_2[B_2]$  is side compatible with  $(T_1[B_1] \setminus t_1, u_1)$ . See Figure 3(c)(d) for an example.*

**Definition 4.** Consider a galled network  $N$  leaf-labeled by  $L$ .  $N$  satisfies a partition  $(X, Y)$  of  $L$  if the root of  $N$  is a non-split node and  $X$  and  $Y$  are the sets of leaves in the two subnetworks attached to the root (see Figure 3(b)).  $N$  satisfies a partition  $(X, Y, Z)$  of  $L$  if the root of  $N$  is a split node,  $Y$  is the set of leaves of the subnetwork attached to the corresponding hybrid node, and  $X$  and  $Z$  are the set of leaves of the subnetworks attached to the two corresponding merge path (see Figure 3(c)(d)).

**Definition 5.** Given a galled network  $N$  satisfying a leaf set partition  $(X, Y, Z)$ . We say that  $N$  is non-skew if both  $X$  and  $Z$  are nonempty, otherwise it is skew. (See Figure 3(c)(d) for examples.)

Below three lemmas describe the properties of both leaf set bipartition and leaf set tripartition.

**Lemma 3.** If  $T_1$  and  $T_2$  admit neither a leaf set bipartition nor a leaf set tripartition, then there is no refining galled network for  $T_1$  and  $T_2$ .

**Lemma 4.** Consider two trees  $T_1$  and  $T_2$  leaf-labeled by  $L$ . A leaf set bipartition  $(X, Y)$  admitted by  $T_1$  and  $T_2$ , if exists, can be computed in  $O(n)$  time. Furthermore, if  $T_1$  and  $T_2$  are galled network-compatible, then there is a SRGN for  $T_1$  and  $T_2$  that satisfies  $(X, Y)$ .

*Proof.* Follows from Lemmas 7 and 8. □

**Lemma 5.** Consider two trees  $T_1$  and  $T_2$  which do not admit any leaf set bipartition. A leaf set tripartition  $(X, Y, Z)$  admitted by  $T_1$  and  $T_2$ , if exists, can be computed in  $O(n)$  time. Furthermore, if  $T_1$  and  $T_2$  are galled network-compatible, there is a SRGN for  $T_1$  and  $T_2$  that satisfies  $(X, Y, Z)$ .

*Proof.* Follows from Lemmas 10, 13. □

Based on the above lemmas, a SRGN for  $T_1$  and  $T_2$  can be computed by Algorithm BuildGalledNetwork as shown in Figure 1. If  $T_1$  and  $T_2$  admit a leaf set bipartition  $(X, Y)$ , the algorithm first builds recursively the SRGN  $N_X$  and  $N_Y$  for  $(T_1|X, T_2|X)$  and  $(T_1|Y, T_2|Y)$  respectively. Then, based on Lemma 4, we return the network formed by attaching  $N_X$  and  $N_Y$  by a common root.

Otherwise, the algorithm checks if  $T_1$  and  $T_2$  admit a leaf set tripartition  $(X, Y, Z)$ . If yes, it builds a SRGN  $N$  for  $T_1$  and  $T_2$  that satisfies  $(X, Y, Z)$  by following Step 1.4-1.11. It does this by recursively building a SRGN  $N_X$  for  $(T_1|X, T_2|X)$ , a SRGN  $N_Z$  for  $(T_1|Z, T_2|Z)$ , and a SRGN  $N_Y$  for  $(T_1|Y, T_2|Y)$ .  $N_X$ ,  $N_Y$ , and  $N_Z$  are combined into  $N$  such that each of  $N_X$  and  $N_Z$  constitutes a merge path from the root of  $N$  to the corresponding hybrid node and  $N_Y$  is the subnetwork rooted at the hybrid node. Since  $N$  is a galled network,  $N_X$  and  $N_Z$  must be constructed such that there is no split node on both merge paths. Thus, by Lemma 2, the algorithm calls BuildSide to construct  $N_X$  and  $N_Z$ .

Otherwise, by Lemma 3, there is no galled network refining  $T_1$  and  $T_2$ .

Note that in Step 1.4-1.11, the algorithm assumes both  $X$  and  $Z$  are nonempty. But a slight technical change can be made to remove this assumption.

In summary, we have the following theorem.

**Theorem 1.** *Given two trees  $T_1$  and  $T_2$ , a SRGN for  $T_1$  and  $T_2$  can be computed by Algorithm `BuildGalledNetwork` in  $O(n^2)$  time.*

The algorithm can be extended to more than two trees.

**Theorem 2.** *The SRGN problem can be solved in  $O(k^2n^2)$  time.*

## 4 Computing the leaf set bipartition of $T_1$ and $T_2$

This section is devoted to prove Lemma 4.

### 4.1 Relationship between leaf set bipartition and SRGN

Suppose  $T_1$  and  $T_2$  are galled network-compatible. This section shows that if there exists a leaf set bipartition  $(X, Y)$  admitted by  $T_1$  and  $T_2$ , then there is a SRGN for  $T_1$  and  $T_2$  that satisfies  $(X, Y)$ .

**Lemma 6.** *Let  $N$  be a SRGN for  $T_1$  and  $T_2$ . For any hybrid node  $h$  of  $N$ , we have  $\Lambda(N[h]) \subseteq \Lambda(T_1[a])$  for some  $a \in s(T_1)$  or  $\Lambda(N[h]) \subseteq \Lambda(T_2[b])$  for some  $b \in s(T_2)$ .*

**Lemma 7.** *Suppose  $T_1$  and  $T_2$  are galled network-compatible. Assume  $T_1$  and  $T_2$  admit a leaf set bipartition  $(X, Y)$ . Then, we can construct a SRGN  $N$  for  $T_1$  and  $T_2$  that satisfies  $(X, Y)$ .*

*Proof.* Let  $N_X$  and  $N_Y$  be SRGN for  $(T_1|X, T_2|X)$  and  $(T_1|Y, T_2|Y)$ , respectively. Let  $N$  be the network formed by attaching  $N_X$  and  $N_Y$  to a common root, then  $N$  satisfies  $(X, Y)$ . By contrary, assume there is a SRGN  $N^*$  for  $T_1$  and  $T_2$  and  $rn(N^*) < rn(N)$ . For every hybrid node  $h \in N^*$ , by Lemma 6, we conclude that either  $\Lambda(N^*[h]) \subseteq X$  or  $\Lambda(N^*[h]) \subseteq Y$ . Hence, the set of hybrid nodes in  $N^*|X$  and  $N^*|Y$  should be disjoint. In other word,  $rn(N^*|X) + rn(N^*|Y) \leq rn(N^*)$ . As  $rn(N_X) \leq rn(N^*|X)$  and  $rn(N_Y) \leq rn(N^*|Y)$ , we have  $rn(N) = rn(N_X) + rn(N_Y) \leq rn(N^*)$ . Thus, we arrived at contradiction and the lemma follows.  $\square$

### 4.2 Linear time algorithm for computing a leaf set bipartition

Next, we describe how to compute a leaf-set bipartition admitted by  $T_1$  and  $T_2$  in  $O(n)$  time. Our computation is based on the bipartite graph  $G(T_1, T_2)$  where  $s(T_1)$  and  $s(T_2)$  are the vertex set on the left and on the right, respectively. In addition,  $(u, v)$  is an edge in  $G(T_1, T_2)$  if and only if  $u \in s(T_1)$ ,  $v \in s(T_2)$  and  $\Lambda(T_1[u]) \cap \Lambda(T_2[v]) \neq \emptyset$ . Note that  $G(T_1, T_2)$  can be constructed in  $O(n)$  time. A partition  $(X, Y)$  is a leaf set bipartition admitted by  $T_1$  and  $T_2$  if and only if  $G(T_1, T_2)$  can be divided into two disjoint subgraphs  $G_1$  and  $G_2$  such that  $X = \cup_{(u,v) \in E(G_1)} \Lambda(T_1[u])$  and  $Y = L \setminus X$ . Thus we get the following lemma.

**Lemma 8.** *A leaf set bipartition admitted by  $T_1$  and  $T_2$ , if exists, can be computed in  $O(n)$  time.*

## 5 Computing the leaf set tripartition of $T_1$ and $T_2$

This section is devoted to prove Lemma 5.

### 5.1 Relationship between leaf set tripartition and SRGN

Given  $T_1$  and  $T_2$ . For any leaf set  $L'$ , let  $rn^*(L')$  denote the number of hybrid nodes in a SRGN for  $T_1|L'$  and  $T_2|L'$ .

**Lemma 9.** *Suppose  $T_1$  and  $T_2$  with leaf set  $L$  do not admit any leaf set bipartition and are galled network-compatible. If  $T_1$  and  $T_2$  admit a leaf set tripartition  $(X, Y, Z)$ , then  $rn^*(X) + rn^*(Y) + rn^*(Z) + 1 \leq rn^*(L)$ .*

**Lemma 10.** *Suppose  $T_1$  and  $T_2$  with leaf set  $L$  do not admit any leaf set bipartition and are galled network-compatible. If  $T_1$  and  $T_2$  admit a leaf set tripartition  $(X, Y, Z)$ , then there is a SRGN  $N$  for  $T_1$  and  $T_2$  that satisfies  $(X, Y, Z)$ .*

*Proof.* We construct  $N$  by following Steps 1.4-1.11 shown in Figure 1. Then  $N$  satisfies  $(X, Y, Z)$  and is a galled network. Let  $N'$  be the network obtained from  $N$  by removing the edge  $(q_X, h)$  and let  $N''$  be the network obtained from  $N$  by removing the edge  $(q_Z, h)$ . Then  $N'$  refines  $T_1$  and  $N''$  refines  $T_2$ , which implies  $N$  refines both  $T_1$  and  $T_2$ . By the construction of  $N$ , we have  $rn(N) = rn^*(X) + rn^*(Y) + rn^*(Z) + 1$ . From Lemma 9, we conclude  $rn(N) = rn^*(L)$ .  $\square$

### 5.2 Linear time algorithm for computing a leaf set tripartition

To find leaf set tripartition for  $T_1$  and  $T_2$ , we examine the graph  $G(T_1, T_2)$  again.

**Lemma 11.** *Suppose  $T_1$  and  $T_2$  did not admit any leaf set bipartition while there exists a non-skew network refining  $T_1$  and  $T_2$  that satisfies  $(X, Y, Z)$ . Then,  $G(T_1, T_2)$  contains an edge  $(u_1, u_2)$  such that  $Y = \Lambda(T_1[u_1]) \cap \Lambda(T_2[u_2])$ . Also,  $G(T_1, T_2) - \{(u_1, u_2)\}$  consists of two disjoint nontrivial star graphs  $G'$  and  $G''$  (where a star graph is a connected graph with at most one node whose degree is larger than 1) such that  $X = \cup_{(u,v) \in G'} \Lambda(T_1[u])$  and  $Z = \cup_{(u,v) \in G''} \Lambda(T_1[u])$ .*

**Lemma 12.** *Given three disjoint sets  $X, Y, Z$  such that  $L = X \cup Y \cup Z$ . We can check if  $(X, Y, Z)$  is a leaf set tripartition admitted by  $T_1$  and  $T_2$  in  $O(n)$  time.*

**Lemma 13.** *Suppose  $T_1$  and  $T_2$  did not admit any leaf set bipartition. We can compute a leaf set tripartition admitted by  $T_1$  and  $T_2$ , if one exists, in  $O(n)$  time.*

*Proof.* When there exist a non-skew network refining  $T_1$  and  $T_2$ , by Lemma 11, there exists an edge  $(u, v) \in G(T_1, T_2)$  such that its deletion divides  $G(T_1, T_2)$  into two star graphs.  $(u, v)$  corresponds to a set  $Y = \Lambda(T_1(u)) \cap \Lambda(T_2(v))$ . As its deletion divides  $G$  into two star graphs, we know that  $T_1|L - Y$  and  $T_2|L - Y$  admit a leaf set bipartition  $(X, Z)$ . By Lemma 12, we can determine if  $(X, Y, Z)$  is a leaf set tripartition in  $O(n)$  time. Note that every  $G(T_1, T_2)$  contains at most two edges  $(u, v)$  such that  $G(T_1, T_2) - \{(u, v)\}$  consists of two disjoint nontrivial star graphs. Hence, a tripartition for  $T_1$  and  $T_2$  can be computed in  $O(n)$  time.

By using a similar idea, we can compute a leaf set tripartition in  $O(n)$  time when there exists a skew network refining  $T_1$  and  $T_2$ .  $\square$

## 6 RGNNet: A new technique for inferring galled networks

In this section, we describe a method we call RGNNet (short for Refining Galled Network) for inferring galled networks from sequence datasets and compare its performance to other methods. RGNNet is based on the approach proposed by Nakhleh *et al.* [9] called SpNet, but unlike that approach, RGNNet is capable of inferring networks with more than one hybrid node. Given two gene datasets of a set of taxa, RGNNet tries to infer a smallest galled phylogenetic network for the set of taxa by utilizing our algorithm for the SRGN problem as follows:

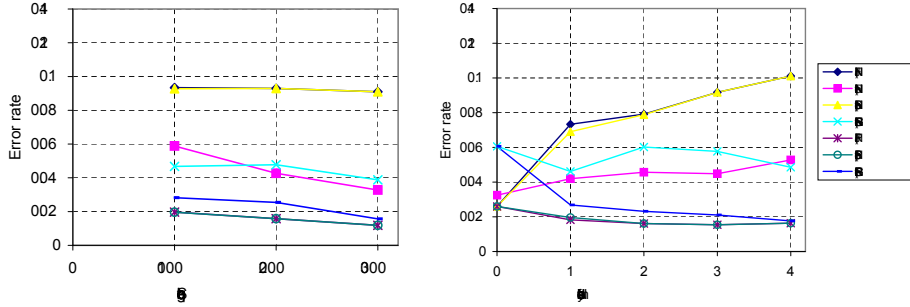
- **Step 1:** For each gene dataset, use maximum likelihood to construct the best two trees for the dataset,
- **Step 2:** For each dataset, compute the strict consensus of the two best trees, thus producing the trees  $t_1$  and  $t_2$ , and
- **Step 3:** If  $t_1$  and  $t_2$  are compatible, combine the datasets and analyze the combined dataset using neighbor-joining (NJ) and return a tree. Else, apply our algorithm for the SRGN problem to  $t_1$  and  $t_2$ . If possible, return a galled network with minimum reticulations that refines  $t_1$  and  $t_2$ ; if no such network exists, we try to *reroot*  $t_1$  and  $t_2$  (described below) to get  $t'_1$  and  $t'_2$ , respectively, and apply our algorithm again. If we still cannot get any SRGN for  $t'_1$  and  $t'_2$ , we apply NJ to the concatenated dataset and return a tree.

*Rerooting the strict consensus trees.* Our algorithm assumes rooted trees and networks. So we use an outgroup to obtain rooted estimates of gene trees inferred by maximum likelihood method (in Step 1). However, the estimates are sometimes rooted incorrectly. This makes the inference of phylogenetic networks impossible. To overcome this problem, whenever there do not exist any SRGN for  $t_1$  and  $t_2$  (in Step 3), we try to reroot  $t_1$  and  $t_2$ . To reroot both trees, we find an edge  $(u_1, v_1)$  in  $t_1$  and  $(u_2, v_2)$  in  $t_2$ , where  $u_i$  is the parent of  $v_i$  for  $i = 1, 2$ , such that  $A(t_1[v_1]) = A(t_2) \setminus A(t_2[v_2])$  and the absolute difference between  $|A(t_1[v_1])|$  and  $|A(t_2) \setminus A(t_2[v_2])|$  is as small as possible. Then, if they exist, for  $i = 1, 2$ , we create a new node  $p_i$  in  $t_i$  by subdividing the edge  $(u_i, v_i)$  and reroot  $t_i$  at  $p_i$ .

### 6.1 Experimental evaluation

To evaluate and compare the performance of the four methods neighbor-joining, NeighborNet, SpNet, and RGNNet for phylogenetic reconstruction, we have carried out extensive simulations. For the simulations, we have used the same methodology as Nakhleh *et al.* [9], as discussed below.

**Experimental settings** We used the model from [8] to generate random networks. Within each generated galled network, we produced two induced gene trees and simulated sequence evolution on these trees under the GTR+  $\Gamma$ +I (gamma distributed rates, with invariable sites) model of evolution, using the settings of [12]. The two separate sequence datasets then were used to run SpNet and RGNNet, and the combined sequence dataset was given to NeighborNet and neighbor-joining. As in [9], we used split-based false positive and false negative rates to measure the error rates of the methods.



**Fig. 4.** FN and FP error rates of neighbor-joining (NJ), NeighborNet(NNet), SpNet, and RGNet on 40-taxon galled model networks. The left graph shows the error rates as a function of concatenated sequence length (c.s.l) on model networks with expected diameter (e.d) 0.5 and 3 hybrid nodes. The right graph shows the error rates as a function of the number of hybrid nodes in model networks with e.d=0.2 and c.s.l=2000. For clarity, the results is shown without the FP rates of NeighborNet.

Topological accuracy Given a phylogenetic tree  $T$  leaf-label by a set  $L$  of taxa. Each edge  $e$  in  $T$  induces a *split*  $\{A(e), B(e)\}$  on  $L$ , where  $A(e)$  is the set of taxa which are descendants of  $e$ , and  $B(e)$  is the set containing the rest of the taxa. We define the set of splits of  $T$ , denoted by  $C(T)$ , as the set of all splits induced by edges in  $T$ . Generalizing the above, let  $N$  be a phylogenetic network whose set of induced trees is referred to as  $\mathcal{T}(N)$ . The set of splits of  $N$ , denoted by  $C(N)$ , is defined as  $C(N) = \bigcup_{T \in \mathcal{T}(N)} C(T)$ . Given a model network  $N_1$  and an inferred network  $N_2$ , the *false positive rate* and *false negative rate* are defined as:  $FP(N_1, N_2) = \frac{|C(N_2) - C(N_1)|}{|C(N_1)|}$  and  $FN(N_1, N_2) = \frac{|C(N_1) - C(N_2)|}{|C(N_1)|}$ .

**Experimental results** We have done extensive experiments to evaluate the performances of the four methods. We focus here on some of the experimental results on 40-taxon galled model networks, shown in Figure 4. The performance of SpNet and neighbor-joining (NJ) are essentially identical, which is expected since SpNet uses NJ whenever it cannot infer a one-hybrid network [9].

As indicated by the figures, the FN error rates of RGNet are comparable to those of NeighborNet and are better than those of NJ and SpNet. Furthermore, the FN rates of RGNet are stable as the number of hybrid nodes increases, while the FN rates of NJ and SpNet increase significantly as the number of hybrid nodes increases. In all cases, NeighborNet shows very poor false positive rates which are always more than 70%, while NJ, SpNet and RGNet show very good false positive rates (almost always less than 3%). The FP rates of RGNet are marginally higher than those of NJ and SpNet. To summarize, RGNet outperforms the other three methods in the combined view of both FN and FP rates.

## 7 Solving the MGNC problem

In this section, we solve the MGNC problem when  $k = 2$ . Given an instance  $\mathcal{T}$ , we call the optimal solution network  $N$  for the problem a *maximum compatible*

galled network (MCGN) for  $\mathcal{T}$ , and  $\Lambda(N)$  a *maximum compatible set* (MCS). We present here an algorithm to compute the MCGN. Extension to  $k > 2$  is straightforward.

### 7.1 The algorithm

Let  $T_1$  and  $T_2$  be two input trees. For any restricted subtrees  $T_1[u_1, A_1]$  and  $T_2[u_2, A_2]$  of  $T_1$  and  $T_2$ , respectively, we define the function  $MCS(T_1[u_1, A_1], T_2[u_2, A_2])$  to be the cardinality of the MCS of  $T_1[u_1, A_1]$  and  $T_2[u_2, A_2]$ . Furthermore, suppose  $T_1[v_1, V_1]$  is a restricted subtree of  $T_1[u_1, A_1]$ , let  $N$  be a MCGN of  $T_1[u_1, A_1] \setminus T_1[v_1, V_1]$  and  $T_2[u_2, A_2]$  such that there is a node  $v$  in  $N$ , where  $\Lambda(N[v]) \subseteq \Lambda(T_1[v_1] \setminus T_1[v_1, V_1])$ , and any node in  $N$ , which is on the path from the root to  $v$  and excluding  $v$ , should be a non-split node. We define  $MCS^*(T_1[u_1, A_1], T_1[v_1, V_1], T_2[u_2, A_2])$  to be the cardinality of  $\Lambda(N)$ .

Below lemmas show the recursive equations for computing  $MCS(T_1[u_1, A_1], T_2[u_2, A_2])$  and  $MCS^*(T_1[u_1, A_1], T_1[v_1, V_1], T_2[u_2, A_2])$ .

**Lemma 14.** *For any restricted subtrees  $T_1[u_1, A_1]$  and  $T_2[u_2, A_2]$  of  $T_1$  and  $T_2$ , respectively, where  $u$  and  $v$  are non-leaf nodes,  $MCS(T_1[u_1, A_1], T_2[u_2, A_2]) =$*

$$\max \begin{cases} \max\{MCS(T_1[u_1, A_1], T_2[v_2, \text{child}(v_2)]) : v_2 \in A_2\} \\ \max\{MCS(T_1[v_1, \text{child}(v_1)], T_2[u_2, A_2]) : v_1 \in A_1\} \\ MCS_1(T_1[u_1, A_1], T_2[u_2, A_2]) \\ MCS_2(T_1[u_1, A_1], T_2[u_2, A_2]) \end{cases}$$

where

- $MCS_1(T_1[u_1, A_1], T_2[u_2, A_2]) = \max\{MCS(T_1[u_1, B_1], T_2[u_2, B_2]) + MCS(T_1[u_1, A_1 - B_1], T_2[u_2, A_2 - B_2]) : B_i \text{ is some nonempty proper subset of } A_i, \text{ for } i = 1, 2\}$ ,  
and
- $MCS_2(T_1[u_1, A_1], T_2[u_2, A_2]) = \max\{MCS^*(T_1[u_1, B_1], T_1[v_1, V_1], T_2[u_2, A_2 - B_2]) + MCS^*(T_2[u_2, B_2], T_2[v_2, V_2], T_1[u_1, A_1 - B_1]) + MCS(T_1[v_1, V_1], T_2[v_2, V_2]) : B_i \text{ is a subset of } A_i, v_i \text{ is some node in } T_i, V_i \text{ is a nonempty proper subset of child}(v_i) \text{ such that } T_i[v_i, V_i] \text{ is a subtree in } T_i[u_i, B_i], \text{ for } i = 1, 2\}$ .

For the base cases, in which  $u_1$  or  $u_2$  is a leaf,  $MCS(T_1[u_1, A_1], T_2[u_2, A_2])$  equals  $|\Lambda(T_1[u_1, A_1]) \cap \Lambda(T_2[u_2, A_2])|$ .

**Lemma 15.** *Consider any restricted subtrees  $T_1[u_1, A_1]$  and  $T_2[u_2, A_2]$  of  $T_1$  and  $T_2$ , respectively. Suppose  $T_1[v_1, V_1]$  is a restricted subtree of  $T_1[u_1, A_1]$ . Then,  $MCS^*(T_1[u_1, A_1], T_1[v_1, V_1], T_2[u_2, A_2])$  equals the maximum of the following three terms.*

- $\max\{MCS^*(T_1[u_1, A_1], T_1[v_1, V_1], T_2[w_2, \text{child}(w_2)]) : w_2 \in A_2\}$ ,
- $MCS^*(T_1[w_1, \text{child}(w_1)], T_1[v_1, V_1], T_2[u_2, A_2])$  where  $w_1 \in A_1$  and  $T_1[v_1, V_1]$  is a subtree in  $T_1[w_1]$ ,
- $\max\{MCS^*(T_1[u_1, B_1], T_1[v_1, V_1], T_2[u_2, B_2]) + MCS(T_1[u_1, A_1 - B_1], T_2[u_2, A_2 - B_2]) : B_1 \text{ and } B_2 \text{ are nonempty proper subset of } A_1 \text{ and } A_2, \text{ respectively, and } T_1[v_1, V_1] \text{ is a subtree in } T_1[u_1, B_1]\}$ .

For base cases, in which  $v_1 = u_1$ , then  $MCS^*(T_1[u_1, A_1], T_1[v_1, V_1], T_2[u_2, A_2]) = MCS(T_1[u_1, A_1 - V_1], T_2[u_2, A_2])$ .

By applying dynamic programming on the above two recursive equations and simple backtracking, we get the following.

**Theorem 3.** *The MCGN of  $T_1$  and  $T_2$  can be computed in  $O(2^{6d}n^4)$  time.*

In general, the above algorithm can be extended to get the following result.

**Theorem 4.** *The MGNC problem can be solved in  $O(2^{3kd}n^{2k})$  time.*

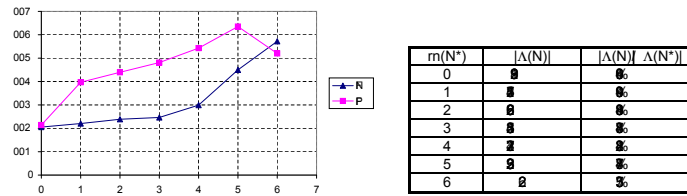
## 8 RGNNet+: Combining RGNNet and our algorithm for the MGNC problem

The simulations in Section 6 demonstrate that RGNNet has superior performance over the other existing methods when the model network is a galled network. However, when the model network is not galled, there is a high chance that RGNNet cannot construct any galled phylogenetic network from the gene tree estimates. In these cases, RGNNet would return a phylogenetic tree by calling neighbor-joining. However, trees are not sufficient to accurately describe the relationships among species that are represented in model networks, leading to poor results in these cases.

To overcome this problem, we can employ our algorithm for the MGNC problem. When RGNNet cannot infer a galled phylogenetic network, we do not call neighbor-joining. Instead, we apply our algorithm for the MGNC problem to remove as few species as possible from the input so that the resulting trees can be merged to a galled network, and then proceed as before. In Section 6, we have shown that RGNNet has very good performance with regards to both the false positive error rate and the false negative error rate. Therefore, the network returned by our approach which combines our algorithms for the MGNC problem and the SRGN problem is highly likely to show the relationships among a subset of species that are represented in the true network. We name this combined approach RGNNet+.

We have done extensive experiments to evaluate RGNNet+. We used the same experimental setting as in Section 6.1. We simulated on general (i.e. not restricted to galled) model networks. For each model network  $N^*$ , we inferred a galled network  $N$  with as many taxa as possible. We used  $FN(N^*|A(N), N)$  and  $FP(N^*|A(N), N)$  to measure the topological accuracy of  $N$ .

Figure 5 shows our experimental results. The FN and FP error rates are less than 7%, which are good. Hence, the inferred networks estimated with high accuracy the relationships among subsets of the species. We also see that on average, the inferred galled networks kept a majority of the species (over 80% when  $rn(N^*) \leq 4$ ). This implies that even when the true networks are not restricted to be galled, the evolutionary relationships among a majority of the species can be represented by a galled network.



**Fig. 5.** Illustrating the performance of RGNNet+. Experiments done on model networks with 30 taxa, 0 to 6 hybrid nodes, concatenated sequence length=2000, and expected diameter=0.5. The table shows the average number of leaves in inferred galled networks as a function of the number of hybrid nodes in model networks. The graph shows the average false positive rate and average false negative rate of inferred galled networks as a function of the number of hybrid nodes in model networks.

## References

1. D. Bryant and V. Moulton. Neighbor-Net: An agglomerative method for the construction of phylogenetic networks. *Molecular Biology and Evolution*, 21(2):255–265, 2004.
2. D. Gusfield, S. Eddhu, and C. Langley. Efficient reconstruction of phylogenetic networks with constrained recombination. In *Proc. of Computational Systems Bioinformatics (CSB2003)*, pages 363–374, 2003.
3. B. Holland and V. Moulton. Consensus networks: A method for visualising incompatibilities in collections of trees. In *Proc. of the 3<sup>rd</sup> Workshop on Algorithms in Bioinformatics (WABI 2003)*, pages 165–176, 2003.
4. D. H. Huson, T. Dezulian, T. Klöpper, and M. Steel. Phylogenetic super-networks from partial trees. In *Proc. of the 4<sup>th</sup> Workshop on Algorithms in Bioinformatics (WABI 2004)*, pages 388–399, 2004.
5. J. Jansson, N. B. Nguyen, and W.-K. Sung. Algorithms for combining rooted triplets into a galled phylogenetic network. In *Proc. of the 16<sup>th</sup> Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, to appear.
6. C. R. Linder, B. M. E. Moret, L. Nakhleh, and T. Warnow. Network (reticulate) evolution: Biology, models, and algorithms. Tutorial presented at the *9<sup>th</sup> Pacific Symposium on Biocomputing (PSB 2004)*, 2004.
7. W. P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.
8. L. Nakhleh, J. Sun, T. Warnow, C. R. Linder, B. M. E. Moret, and A. Tholse. Towards the development of computational tools for evaluating phylogenetic reconstruction methods. In *Proc. of the 8<sup>th</sup> Pacific Symposium on Biocomputing (PSB 2003)*, pages 315–326, 2003.
9. L. Nakhleh, T. Warnow, and C. R. Linder. Reconstructing reticulate evolution in species – theory and practice. In *Proc. of the 8<sup>th</sup> Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004)*, pages 337–346, 2004.
10. N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.
11. L. Wang, K. Zhang, and L. Zhang. Perfect phylogenetic networks with recombination. *Journal of Computational Biology*, 8(1):69–78, 2001.
12. D. Zwickl and D. Hillis. Increased taxon sampling greatly reduces phylogenetic error. *Systematic Biology*, 51(4):588–598, 2002.