BMC Evolutionary Biology

Methodology article

Drawing explicit phylogenetic networks and their integration into SplitsTree

Tobias H Kloepper* and Daniel H Huson

Address: Center for Bioinformatics ZBIT, Tübingen University, Sand 14, 72076 Tübingen, Germany

Email: Tobias H Kloepper* - kloepper@informatik.uni-tuebingen.de; Daniel H Huson - huson@informatik.uni-tuebingen.de

* Corresponding author

Published: 24 January 2008

BMC Evolutionary Biology 2008, **8**:22 doi:10.1186/1471-2148-8-22

Received: 16 November 2007
Accepted: 24 January 2008

This article is available from: http://www.biomedcentral.com/1471-2148/8/22

Abstract

Background: SplitsTree provides a framework for the calculation of phylogenetic trees and networks. It contains a wide variety of methods for the import/export, calculation and visualization of phylogenetic information. The software is developed in Java and implements a command line tool as well as a graphical user interface.

Results: In this article, we present solutions to two important problems in the field of phylogenetic networks. The first problem is the visualization of explicit phylogenetic networks. To solve this, we present a modified version of the equal angle algorithm that naturally integrates reticulations into the layout process and thus leads to an appealing visualization of these networks. The second problem is the availability of explicit phylogenetic network methods for the general user. To advance the usage of explicit phylogenetic networks by biologists further, we present an extension to the SplitsTree framework that integrates these networks. By addressing these two problems, SplitsTree is among the first programs that incorporates *implicit* and *explicit* network methods together with standard phylogenetic tree methods in a graphical user interface environment.

Conclusion: In this article, we presented an extension of SplitsTree 4 that incorporates explicit phylogenetic networks. The extension provides a set of core classes to handle explicit phylogenetic networks and a visualization of these networks.

Background

Phylogenetic networks are graphs used for representing phylogenetic relationships between different taxa, and are usually employed when a tree representation does not suffice. There are many different types of phylogenetic networks and it is useful to distinguish between two main classes: *implicit* phylogenetic networks that provide tools to visualize and analyze incompatible phylogenetic signals, such as split networks [1,2], and *explicit* phylogenetic networks that provide explicit scenarios of reticulate evolution, such as hybridization networks [3-7], HGT networks [8] and recombination networks [9-19].

The software currently available for the calculation and analysis of explicit phylogenetic networks consists of a spread of basic implementations of algorithms developed to solve the computational task [6,14,16-18,20]. Most of the software is command line driven and an appealing visualization of the results is often lacking. It is essential to have a tool that allows both broad usage of the methods available to biologists, and better and further development of new methods.

SplitsTree is an application developed in our research group, originally aiming at the phylogenetic analysis of sets of splits. The newest version of SplitsTree [2] incorporates a variety of methods for the calculation, visualization and interpretation of phylogenetic trees and implicit phylogenetic networks. Two main advantages of SplitsTree are the graphical user interface (GUI) and the integration of algorithms via an interface driven class loader (plugins). In this article we present an extension to SplitsTree that enables the program to handle explicit phylogenetic networks. The extension solves two important problems: an efficient integration of explicit phylogenetic networks, and visualizing these networks.

Results and Discussion

A *tree* $T = (V, E)$ is a connected acyclic graph with vertex set V and edge set E. A vertex of degree one is called a *leaf* of T and the set of all leaves is called the *leaf set* of T. A *rooted tree* $T = (V, E, \varrho)$ is a tree (V, E) that has exactly one distinguished vertex called the *root*, denoted ϱ. A rooted tree T has a natural ordering where $v \leq v'$, if v lies on the path from the root to v'. If $v \leq v'$, we say that v is an *ancestor* of v' and v' is a *descendant* of v. For any set of vertices V, a vertex v is called *minimal* with respect to V if for all v' in V, it holds that $v \leq v'$. For any edge e, we use $\alpha(e)$ and $\beta(e)$ to denote the source and target of e. A *rooted phylogenetic X-tree* is a pair (T, ν), where $T = (V, E, \varrho)$ is a rooted tree and $\nu : X \to V$ is a bijection from X to the leaf set of T. See [21] for more details.

Definition 1 *Let X be a set of taxa. A* rooted reticulate network $N = N(V, E, \nu)$ *on X is a connected, directed acyclic graph with vertex set V, edge set E and vertex labeling* $\nu : X \to V$*, such that:*

1. *there exists precisely one distinguished vertex* ϱ *called the* root;

2. *every vertex* $v \in V$ *is either a* tree vertex, $v \in V_T$*, that has exactly one ancestor, or a* reticulation vertex $r \in V_R$ *that has exactly two ancestors;*

3. *every edge is either a* tree edge *leading to a vertex of indegree one or a* reticulation edge *leading to a vertex of indegree two; and*

4. *the set of* leaves *L (vertices with no descendants) consists only of tree vertices and is labeled by the set of taxa X, i.e.* ν *maps X bijectively onto L.*

It follows from these definitions that each reticulation vertex (or *reticulation*, for short) $r \in V_R$ is contained in one or more cycles of the form $C = (r, p(r), w_1, e_1, ..., e_{k-1}, w_k, q(r), r)$, with $w_i \in V$ and $e_i \in E \setminus \{p(r), q(r)\}$ for all i. (Note that additionally, r can also be contained in one or more cycles that do not contain $p(r)$ and $q(r)$). We say that two reticulations $r, r' \in V_R$ are *dependent* if a cycle that contains both r and r' exists.

In graph theory, a *two-connected component* of a graph G is any maximal subgraph G' with the property that any two vertices v and w of G' are connected by two paths p and p' that share no vertices except for v and w. For any reticulation vertex r, let p^r and q^r denote the two associated reticulation edges.

Furthermore, let v_p^r and v_q^r denote the two ancestors of r with respect to p^r and q^r. The *lowest single ancestor* $lsa(r)$ of a reticulation r is the minimum of all nodes in V that is connected to r by two paths p and p' that share no vertices except for $lsa(r)$ and r.

Algorithm

One important approach to drawing trees is the equal angle algorithm which was developed by Meacham (see [22]). The equal angle algorithm guarantees that no two edges intersect. Our algorithm for visualizing recombination networks is based on a generalization of the equal angle algorithm. The algorithms adds an ordering step at each vertex, that chooses an optimal ordering of the descending edges, that minimizes the number of crossings between reticulations edges and other edges. It can easily be altered to be used with any drawing algorithm for trees. We will start out with a description of the equal angle algorithm and will then define some basic properties. Finally, we will give solutions to minimize crossing edges in a drawing of a reticulate networks, and the optimal placement of reticulation vertices.

The equal angle algorithm is a recursive algorithm that starts at an internal vertex of a tree. For each subtree connected to the starting vertex, we appoint an angle proportional to the share of leaves it contains. In the next step, we assign to each subtree a sector of the circle of the size of the angle appointed to it and draw the edge to the subtree in the middle of the sector. We place the sector of the subtree in a way that it is centered at the end of the branch and the branch is pointing at the bisector of the angle. We then recurse to the starting vertex of the subtree and assign each newly discovered subtree its proportional share of the angle. Each subtree is than placed in the sector of the starting vertex. The recursion is repeated until we have appointed angles to each branch of the tree. The only modifications for rooted trees are the explicit start point (the root of the tree) and the use of a fraction of the cycle. For a detailed description of the algorithm, see [22].

The rooted equal angle algorithm is not directly applicable to a reticulate network since for each reticulation, we have to decide which of the reticulation edges we want to use for the drawing algorithm and either choice may be suboptimal. The idea behind our approach is to use neither of them. The *influence* of a reticulation upon the graph structure is bounded by the reticulation and its lowest single ancestor, therefore we decided to define an *auxiliary edge* between those two vertices and to use the auxiliary edges for the layout of the graph. When the algorithm reaches a node each descending edge is checked for its status (being either a tree-edge, an auxiliary-edge or a reticulation-edge) and only tree- and auxiliary-edges are incorporated into the process.

Through these modifications to the rooted equal angle algorithm, it is possible to visualize reticulate networks, but these visualizations are not very satisfying. To obtain an improved method, we will address two key problems. The first problem is the crossing of reticulation edges: even though it can not always be avoided, the number of such events should be minimized. The second problem is that the auxiliary edges are artifical edges and their optimal edge length must be determined. In the following, we will show solutions to these two problems.

Minimizing crossing edges

An edge crossing another one is an undesirable event in drawing a graph. It is well known that solving this problem is, in general, computationally hard [23]. The equal angle algorithm ensures that we only have to deal with reticulation edges crossing other edges. Furthermore, the construction of the auxiliary edges implies that edges that can be crossed by the reticulation edges are descendent edges of the lowest single ancestor of the reticulation. The optimization starts at the root of the networks and optimizes the arrangement of the directly descending vertices. It then continues the optimization iteratively at each directly descending vertex in the order given and keeps going until it has optimized all placements.

Let V_v^T be the set of tree vertices directly below a vertex v and let V_v^I be the set of reticulation vertices connected to v by auxiliary edges. We say that a *tree path* $p(v, v')$ from a vertex v to a vertex v' exists if v' is a descendant of v and every edge in $p(v, v')$ is either a tree- or auxiliary-edge. Furthermore, we say that a reticulation r is *easily reachable* from a vertex v if a tree path $p(v, v_p^r)$ exists. Finally, let R_v be the set of all reticulations that are easily reachable from the vertex v.

The set R_v can be divided into those reticulations r for which $v = lsa(r)$, which we will again denote by R_v^G; v is a descendant of $lsa(r)$, denoted by R_v^D; and v is an ancestor of $lsa(r)$, denoted by R_v^A. If v is the root, R_v^D is empty. The set R_v^D can be divided further. Since for a reticulation r in R_v^D, the nodes directly below $lsa(r)$ have been previously sorted, we can denote the set $\bar{R}_v^D$ as containing those r in R_v^D for which r is sorted less than the directly descending node of $lsa(r)$ leading via a tree path to v.

The aim of our optimization is to find a linear arrangement of the vertices in $V_v^T \cup V_v^I$ such that the number of reticulation edges, in the subtrees of the vertices in $V_v^T \cup V_v^I$, intersecting with tree edges is minimized. We define the *optimal linear arrangement graph OLA*^{v} (V, E) of a vertex v as one that contains a vertex representative for any vertex in $V_v^T \cup V_v^I$. We add a weighted edge between any two vertices (v_i, v_k) in V and set the weight w_{ik} of the edge to $\left| R_{v_i}^D \cap R_{v_k}^D \right|$. More formally written:

Problem 1 *With*

$$x_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ takes position } k, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall i, k$$

minimize

$$\sum_{(i,j)\in E} w_{ij} x_{ik} x_{jl} \left| k - l \right| + \sum_{i \in V} \left| R_{v_i}^D \cap \bar{R}_n^D \right| x_{ik} \left| k \right| + \sum_{i \in V} \left| R_{v_i}^D \cap \left(R_n^D \setminus \bar{R}_n^D \right) \right| x_{ik} \left| \left| V \right| - k \right|$$

$$\textit{subject to} \quad \sum_{i \in (V)} x_{ik} = 1 \quad \textit{and} \quad \sum_{k \in \{1,\ldots,|V|\}} x_{ik} = 1, \quad \forall i, k$$

The optimal linear arrangement problem is known to be *hard* [24]. Nevertheless, this arrangement problem is in general much smaller than the complexity of minimizing all crossing edges at once. Interestingly, a couple of additional restrictions exist that we may apply to the ordering, leading to a "greedy" solution that works well in most cases. One restriction that we can place upon the structure is that for any reticulation r, the position in the ordering should be between v_p^r and v_q^r. Consequently, we should place v_p^r and v_q^r before we place r.

Another restriction we can place is a consequence from the dependency of the reticulations upon each other. For any pair of reticulation *r*, *r*' in R_v^G we say that *r* is less than *r*' if and only if a tree path $p(r,\ v_p^{r'})$ exists. To meet the first restriction we have to place *r* before we can place *r*'. The graph that can be constructed from the relations between the reticulations must be cycle free, since the reticulation network is cycle free. Consequently, we can use a standard topological sorting algorithm to obtain a linear ordering $Ord_l(R_v^G)$ for the reticulations in R_v^G.

The optimization algorithm iterates through the ordering and at each reticulation *r* it first places v_p^r and v_q^r, if necessary, and then *r*. If all reticulation are placed, the algorithm places all descending tree edges that have not yet been placed. At each placement, the algorithm positions the vertex at the position that minimizes the score given in Problem 1. After all nodes have been placed in the linear arrangement, the result is returned to the main method. An Example of the optimization procedure can be seen in Figure 1.

Optimal placement for reticulation vertices

Having calculated the angle and optimal arrangement for each edge, we have to place the vertices. Tree vertices can be placed in the same way as in the standard equal angle algorithm. But since auxiliary edges do not come with a given length, we have to calculate an optimal placement for each of the reticulation vertices. Such a placement has to incorporate the conditions of the equal angle algorithm, otherwise we might face unnecessary crossings between edges. Note that there are two cases for which we have to consider different placement methods. In the first case, we have a reticulation *r* where the nodes v_p^r and v_q^r are both different from *lsa*(*r*), and in the second case, one of them is equal to *lsa*(*r*).

In both cases, we place the reticulation vertex *r* on the bisector of the sector assigned to its auxiliary edge. In the first case, the distance between *r* and *lsa*(*r*) should be larger than the minimum distance between *lsa*(*r*) and the line $l(v_p, v_q)$, indicating that *r* is a descendant of v_p and v_q. In other words, we assume the angles $v_q v_p r$ and $v_p v_q r$ are positive. In the second case, we assume that v_q is equal to *lsa*(*r*). We first calculate the point on the bisector r_t that has the same distance to *lsa*(*r*) as v_p and than ensure that the angle between $r_t v_p r$ is positive. We added an option to the algorithm so that the user can specify the (maximum) value of this angle; the standard value is 15°. An example of the drawing algorithm can be seen in Figure 2.

Implementation

We started to integrate explicit phylogenetic networks into SplitsTree in our RECOMB 2005 article [6]. Originally, such methods were squeezed into the existing data structures within SplitsTree. SplitsTree itself is built around a group of core classes, each one representing a different type of information. The standard file format of SplitsTree

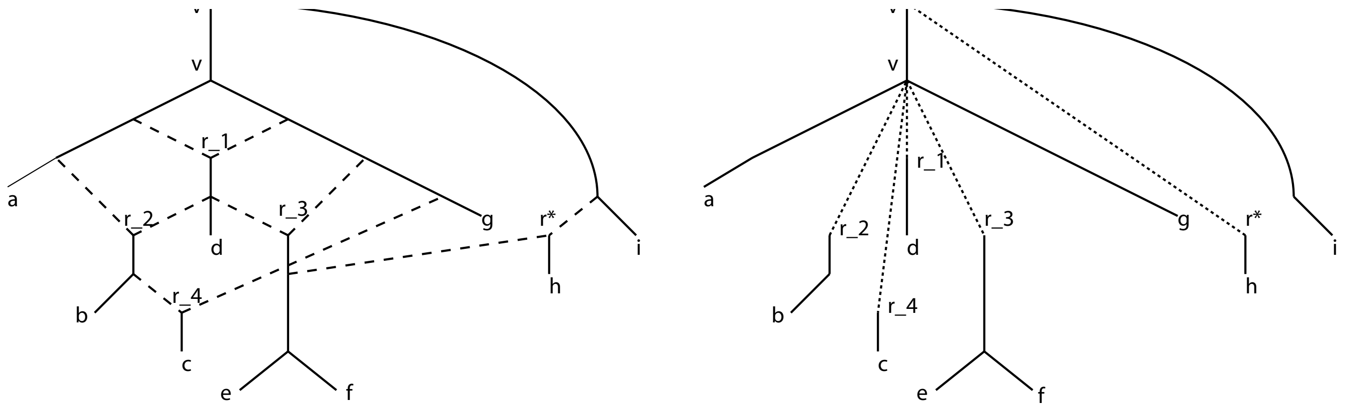

Figure 1

Example of the layout optimization. The figure on the left side shows an explicit phylogenetic network. The reticulation edges of the network are shown as dashed lines. Removing the reticulation edges and integrating the auxiliary edges into the network leads to a tree structure, as shown on the right-hand side of the figure (auxiliary edges are drawn as dashed lines). The set of easily reachable edges R_v of the node *v* contains the reticulations *r*1, *r*2, *r*3, *r*4 and *r**. The set R_v^D only contains *r** and is equal to $\bar{R}_v^D$. The placement of *r** has cost 1 since the reticulation edges only cross the edge that leads to leaf *g*. The placement of *r*4 has cost 2, since the reticulation edges to the right crosses *r*1 and *r*3.

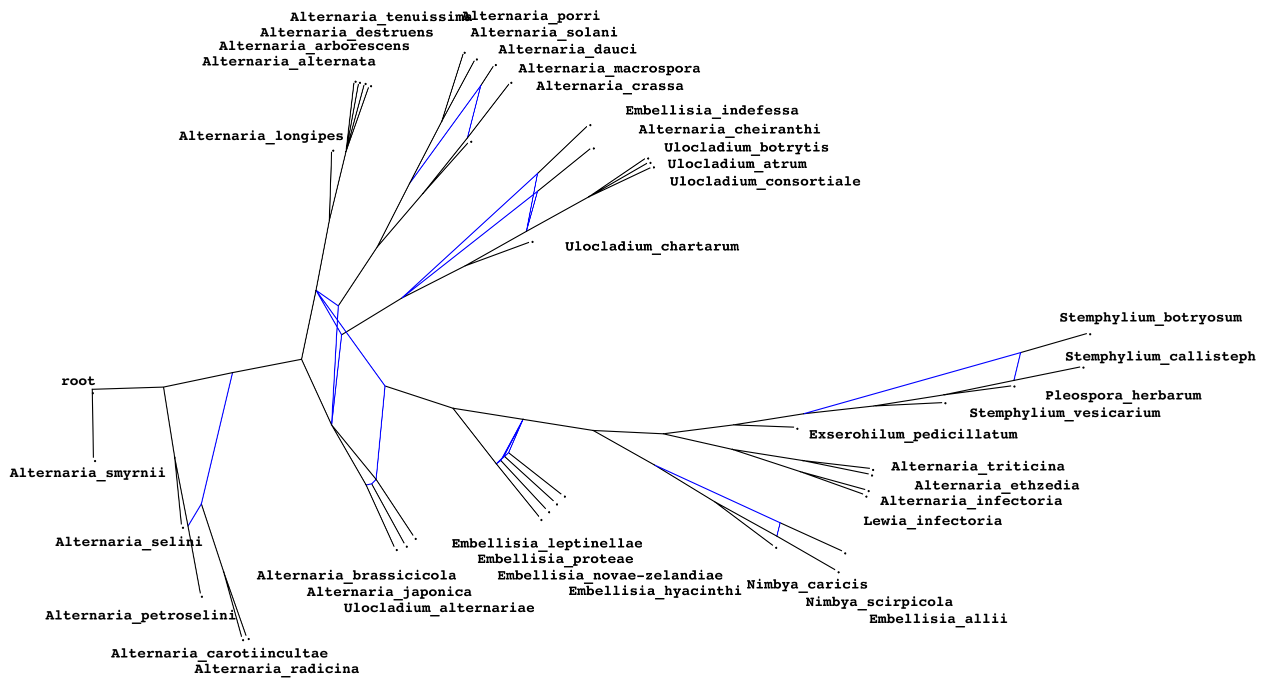

Figure 2
Example of the drawing algorithm. The figure shows the drawing of a reticulation network that we recently published [20] which is based on three gene trees described in [28].

is the *Nexus* [25] file format and each core class has its own Nexus representation. Consequently, developing a Nexus representation of explicit phylogenetic networks is essential for the integration of these into SplitsTree.

To build a Nexus representation for an explicit phylogenetic network, one needs to find an efficient way to present it as a string. We decided to use a version of the *extended Newick* (eNewick) [26] format. In general, the eNewick format allows labels to be present up to two times within the network. A label is allowed to appear once as a leaf and once as an internal label. Whenever a label occurs twice, the leaf is identified with the internal vertex, thus providing a network with vertices of indegree two. A lot of research has lately been focused on proving some interesting decomposition theorems [6,16,20] for explicit phylogenetic networks. The general motivation of these theorems is that the calculation of a reticulate network, with a minimal number of reticulation events, from some given information is hard [27]. The idea is to decompose each network into its two-connected components and to calculate the minimal solutions of each two-connected component separately.

Following the idea of decomposing explicit phylogenetic networks, each two-connected component may have several solutions and the possible combinations of these solutions grows exponentially, which is a problem if the number of two-connected components is large. Consequently, we decided that the Nexus representation of the network needs to reflect the two-connected components within it.

Note that any reticulate network contains either a two-connected component or a tree like element, that contains the root. We call this particular element the *root component*. The two-connected components are called *netted components*, and for each netted component, a number of solutions may exist. Any connected component that is not a two-connected component is a *tree component*. Each tree component may appear more than once within the possible configurations. The possible combinations of these three basics elements is left to the user.

We now describe a Nexus notation for explicit phylogenetic networks, the schematic of this notation is shown in Figure 3. In general, one needs to save the components

```
Begin Reticulate;
        Dimensions
                NTax = number-of-taxa
                NRootComponents = number-of-root-components
                NNettedComponents = number-of-netted-components
                NTreeComponents = number-of-tree-components;
        [Format
                [ActiveRoot = position-of-active-root-component;]
                [ActiveNettedComponents = positions-of-active-netted-components;]
                [ShowLabels =[Internal][TreeComponents][NettedComponents];]
        ]
        TreeComponents
                [name = tree-component-specification;]
                [name = tree-component-specification;]
                ...
        NettedComponents
                [nettedComponent name=
                        [name = netted-component-specification;]
                        [name = netted-component-specification;]
                        ...
                ]
                [nettedComponent name=
                        [name = netted-component-specification;]
                        [name = netted-component-specification;]
                        ...
                ]
                ...
        RootComponents
                [name = root-component-specification;]
                [name = root-component-specification;]
                ...
End;
```

Figure 3
Reticulate Nexus Block Schematic. Shown is a schematic of the Reticulate nexus block as it is implemented in SplitsTree. The block is divided into three parts: the *Dimensions* contains all information about the dimensions of the reticulate network; the *Format* is an optional element that describes the configuration of the reticulate network; and the *TreeComponents*, *NettedComponents* and *RootComponents* contain the string representation of the reticulate network.

containing the root in the *RootComponents* section. Any such string should either be formated in standard eNewick, in Newick format where any two leaves with the same label are labeled with the name of a tree component, or in Newick format where at least one leaf is labeled with the name of a netted component.

The *NettedComponents* section contains a list of all two-connected components. Each one must be identified by a unique name and there must be at least one string representation given for each. Any such string must either be formated in eNewick, or in Newick format where any two leaves with the same label are labeled with the name of a tree component.

Finally, the *TreeComponents* section contains a list of uniquely named strings in Newick format, where leaves can be labeled with the name of netted components.

Conclusion

In this article we presented a new algorithm for the visualization of explicit phylogenetic networks. The algorithm is a generalization of the well known equal angle algorithm and can be used to adapt most known phylogenetic tree drawing algorithm to the task of drawing reticulate networks. Moreover, we have described a datastructure and file format for representing reticulate networks in a way that reflects the structural properties of the networks.

Our implementation of these results in the popular SplitsTree software will make them accessible to biologists and other researchers that are interested in using such networks.

Availability and requirements

• **Project name:** Drawing Phylogenetic Networks

• **Project home page:** http://www.SplitsTree.org

• **Operation system(s):** Platform independent

• **Programming language:** Java

The extensions to SplitsTree 4 are freely available for users of the application. SplitsTree 4 can be downloaded from the projects home page. Using the application is free.

Authors' contributions

TK designed the algorithm and the integration into SplitsTree. TK and DH implemented the algorithm and the integration. TK and DH wrote the manuscript for the article.

Acknowledgements

We would like to thank Anand Radhakrishnan, C. Nickias Kienle and K. Wiederhold for many helpful discussions. Furthermore we thank anonymous reviewers for their helpful comments and suggestions.

References

1. Bandelt HJ, Dress AWM: **A canonical decomposition theory for metrics on a finite set.** *Advances in Mathematics* 1992, **92:**47-105.
2. Huson DH, Bryant D: **Application of phylogenetic networks in evolutionary studies.** *Molecular Biology and Evolution* 2006, **23:**254-267. [Software available from http://www.splitstree.org].
3. Sang T, Zhong Y: **Testing hybrization hypotheses based on incongruent gene trees.** *System Biol* 2000, **49(3):**422-424.
4. Linder CR, Rieseberg LH: **Reconstructing Patterns of Reticulate Evolution in Plants.** *Am J Bot* 2004, **91(10):**1700-1708.
5. Nakhleh L, Warnow T, Linder CR: **Reconstructing reticulate evolution in species – theory and practice.** *Proceedings of the Eighth International Conference on Research in Computational Molecular Biology (RECOMB)* 2004:337-346.
6. Huson D, Kloepper T, Lockhart P, Steel M: **Reconstruction of Reticulate Networks from Gene Trees.** In *Proceedings of the Ninth International Conference on Research in Computational Molecular Biology (RECOMB) Volume 3500.* LNCS, Springer Verlag; 2005:233-249.
7. Bordewich M, Semple C: **Computing the minimum number of hybridisation events for a consistent evolutionary history.** *Discrete Appl Math* 2007, **155(8):**914-928.
8. Hallett M, Largergren J, Tofigh A: **Simultaneous Identification of Duplications and Lateral Transfers.** *Proceedings of the Eight Inter-*

national Conference on Research in Computational Molecular Biology (RECOMB) 2004:347-356.

9. Hudson RR: **Properties of the Neutral Allele Model with intergenic Recombination.** *Theoretical Population Biology* 1983, **23:**183-201.
10. Hein J: **Reconstructing evolution of sequences subject to recombination using parsimony.** *Math Biosci* 1990:185-200.
11. Griffiths RC, Marjoram P: **Ancestral Inference from Samples of DNA Sequences with Recombination.** *J Computational Biology* 1996, **3:**479-502.
12. Gusfield D, Eddhu S, Langley C: **Efficient reconstruction of phylogenetic networks with constrained recombination.** *Proceedings of the IEEE Computer Society Conference on Bioinformatics* 2003:363.
13. Song Y, Hein J: **Parsimonious Reconstruction of Sequence Evolution and Haplotype Blocks: Finding the Minimum Number of Recombination Events.** *Proceedings of the IEEE Computer Society Conference on Bioinformatics* 2003:287-302. [Proceedings of the Workshop on Algorithms in Bioinformatics].
14. D Gusfield SE, Langley C: **The Fine Structure of Galls in Phylogenetic Networks.** *INFORMS J. of Computing Special Issue on Computational Biology* 2004, **16(4):**459-469.
15. Song Y, Hein J: **On the minimum Number of Recombination Events in the Evolutionary History of DNA Sequences.** *J Math Biol* 2004, **48:**160-186.
16. Gusfield D, Bansal V: **A Fundamental Decomposition Theory for Phylogenetic Networks and Incompatible Characters.** *Proceedings of the Ninth International Conference on Research in Computational Molecular Biology (RECOMB)* 2005:217-232.
17. Huson D, Kloepper T: **Computing Recombination Networks from Binary Sequences.** *Bioinformatics* 2005, **21(suppl. 2):**ii159-ii165. [ECCB].
18. Song Y, Hein J: **Constructing Minimal Ancestral Recombination Graphs.** *J Comp Biol* 2005, **12:**147-169.
19. Lyngsø RB, Song YS, Hein J: **Minimum Recombination Histories by Branch and Bound.** *WABI* 2005:239-250.
20. Huson DH, Kloepper TH: **Beyond Galled Trees – Decomposition and Computation of Galled Networks.** 2007. [Accepted to RECOMB2007].
21. Semple C, Steel MA: *Phylogenetics* Oxford University Press; 2003.
22. Felsenstein J: *Inferring Phylogenies* Sinauer Associates, Inc; 2004.
23. MR Garey DSJ, Stockmeyer L: **Some simplified NP-complete graph problems.** *Theoretical Computer Science* 1976:237-267.
24. Garey M, Johnson D: **Crossing Number is NP-Complete.** *SIAM J Alg Discr Math* 1983:312-316.
25. Maddison D, Swofford D, Maddison W: **NEXUS: an extendible file format for systematic information.** *System Bio* 1997, **46(4):**590-621.
26. Morin MM, Moret BME: **NetGen: generating phylogenetic networks with diploid hybrids.** *Bioinformatics* 2006, **22(15):**1921-1923.
27. Wang L, Zhang K, Zhang L: **Perfect Phylogenetic Networks with Recombination.** *Journal of Computational Biology* 2001, **8:**69-78.
28. Pryor BM, Bigelow DM: **Molecular characterization of Embellisia and Nimbya species and their relationship to Alternaria, Ulocladium and Stemphylium.** *Mycologia* 2003, **95(6):**1141-1154.