Hierarchical Clustering Using Constraints

Mariana Kant[1], Maurice LeBon[1], and David Sankoff[2]

[1] Computer Science and Engineering Department,
York University, Toronto, Canada M3J 1P3
mkant@yorku.ca
[2] Department of Mathematics and Statistics,
University of Ottawa, Ottawa, Canada K1N 6N5
sankoff@uottawa.ca

Abstract. We describe a new supertree algorithm that extends the type of information that can be used for phylogenetic inference. Its input is a set of constraints that expresses either the hierarchical relationships in a family of given phylogenies, or/and other relations between clusters of sets of species. The output of the algorithm is a multifurcating rooted supertree which satisfies all constraints. Moreover, if there were contradictions in the set of constraints the corresponding part of the supertree is identified and its set of constraints is displayed such as the user may decide to modify or keep it. Our algorithm is not affected by the order in which the input phylogenies or other constraints are presented. We apply our method to a number of data sets.

1 Introduction

Supertree construction has taken on increasing importance with the widespread adoption by biologists of the "Tree of Life" endeavour (e.g., [1,2,3,4]). The input to the supertree problem is a family $\mathcal{T}$ of rooted trees with overlapping leaf sets. The object is to build T, a supertree compatible with all trees in $\mathcal{T}$, namely a rooted tree whose leaf set includes all species of input trees and from which each tree of the family can be derived by a sequence of edge contractions.

Numerous algorithms have been proposed to build supertrees. Gordon [5] presented an algorithm for building a "strict consensus supertree" of two rooted binary trees. The time complexity of his algorithm is $O(p^3)$, where p is the maximum between the number of leaves of the two trees. Semple and Steel [6] proposed MinCutSupertree algorithm to build a rooted supertree for a family of rooted weighted binary trees. Berry and Nicholas [7] propose MergeTrees, an algorithm for constructing a supertree compatible with two binary trees by grafting "specific subtrees" or "specific leaves" of one tree onto the other. For $|\mathcal{T}| \geq 3$, they apply MergeTrees repeatedly to pairs of trees, each time reducing $|\mathcal{T}|$ by 1.

In this paper we reformulate the supertree problem in a general way, but so that it allows a unique solution by means of an efficient algorithm, which we detail. The input is a set of constraints, which may or may not be derived from trees, on sets of elements. There may be any number of constraints, e.g. for $|\mathcal{T}| \geq 3$. We derive the unique rooted supertree compatible (in a strong sense) with the set of

I. Măndoiu, R. Sunderraman, and A. Zelikovsky (Eds.): ISBRA 2008, LNBI 4983, pp. 2–13, 2008.

constraints. Rather than weaken the notion of compatibility, we allow multifurcating trees (cf. [8]), both for input and output, where contradictory constraints are handled by multifurcations. The output is independent of the order in which the constraints are presented in the input. The algorithm runs in time proportional to the number of distinct leaves across all constraints and proportional to the number of constraints. The algorithm identifies contradictions in the set of constraints, allowing the user to intervene and decide which constraints to retain or discard from the set, depending on the degree of resolution desired.

We illustrate our procedures with data on primate phylogeny.

2 Definitions

In this paper we use "tree" to denote a multifurcating rooted tree, as exemplified in Figure 1. Following the notation and terminology of [7], a tree T has a leaf set $L(T)$ in bijection with a label set. Each internal node (including the root) has at least two children. An edge between two internal nodes is an internal edge. A leaf $x, x \in L(T)$ is a descendant of an internal node u, if the path from x to the root passes through u. For a node u in T, we write $S(u)$ for the subtree rooted at u, i.e., u and all its descendant nodes, and $L(u)$ the leaves of this subtree.

Definition 1 (Restriction of a tree). *The restriction of a tree T to a set of leaves X, itself a tree denoted $T|X$, is the smallest induced graph of T connecting leaves with labels in $X \cap L(T)$, where each degree two (non-root) node x as well as its two incident edges (u, x) and (v, x) are replaced by a single edge (u, v) to make the tree homeomorphically irreducible. If $\mathcal{T}$ is a collection of trees, then the collection of subtrees $\mathcal{T}|X := \{T|X : T \in \mathcal{T}\}$.*

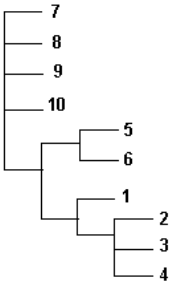

Fig. 1. Diagram of the tree T'

Definition 2 (Tree isomorphism and inclusion). *Two rooted trees T, T' are isomorphic, denoted $T = T'$, if and only if there is a graph isomorphism $T \to T'$ preserving leaf labels (and the root). Tree T is homeomorphically included in T' if and only if $T = T'|L(T)$.*

Definition 3 (Tree refinement). *If $L(T) = L(T')$, the tree T refines the tree T', written $T \ni T'$, if T can be transformed into T' by collapsing some of its internal edges (collapsing an edge means removing it and merging its extremities).*

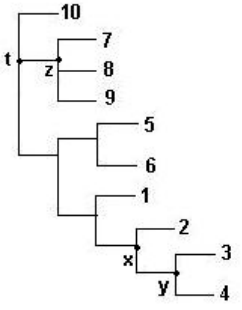

Fig. 2. Diagram of the tree T. T refines the tree T' in Figure 1.

The tree T in Figure 2 refines the tree T' in Figure 1. T' may be obtained from T by collapsing the edges (x, y) and (z, t).

Definition 4 (Tree compatibility). *Let T, T' be trees with leaf sets L, L', respectively. We say that T displays T' if $T|L' \supset T'$. Given a collection of trees $\mathcal{T}$, if there is a tree T that displays every tree in $\mathcal{T}$, we say that T displays $\mathcal{T}$ and that the collection $\mathcal{T}$ is compatible.*

Our algorithms will make use a definition of a set-theoretic definition of a tree [9,10,11].

Definition 5 (Tree-like family). *Let S be a set of n elements. $\mathcal{T}_S$, a tree-like family on the set S is a family of nonempty subsets of S such that:*

1. $S \in \mathcal{T}_S$,
2. $\{e\} \in \mathcal{T}_S$ *for all* $e \in S$,
3. $\forall A, B \in \mathcal{T}_S, A \cap B \in \{A, B, \emptyset\}$.

Example 1. *Given $S = \{1, 2, 3, 4, 5, 6\}$, the collection $\mathcal{T}_S = \{S, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}\{6\}, \{1, 2, 3\}, \{4, 5, 6\}, \{4, 5\}\}$ is a tree-like family.*

It is clear that $\mathcal{T}_S$ is a partially ordered set under set inclusion. Moreover, for every set $\{e\}$ the collection $A_{\{e\}} = \{A : A \in \mathcal{T}_S \text{ and } \{e\} \subseteq A\}$ is a totally ordered set.

For a tree-like family $\mathcal{T}_S$, we consider the graph $G_S = (E, V)$, such that for each set $A \in \mathcal{T}_S$ there is a corresponding node $v_A \in V$; for every pair of nodes v_A, v_B, with $A \subseteq B$ and no set C in the family with $A \subseteq C \subset B$, there is an edge between v_A and v_B.

Example 2. *The graph in Figure 3 is the graph associated with the tree-like family in Example 1.*

Property 1. *The graph G_S associated to a tree-like family is a tree.*

Proof: Let v_S be the root of the tree. For every leaf $v_{\{e\}}$ there is an unique path between the root v_S and $v_{\{e\}}$, namely the path corresponding to all sets in the collection $A_{\{e\}}$. Suppose there is a cycle between two distinct nodes v_A and v_B. This means that $A \subseteq B$ and also $B \subseteq A$. Hence, $A = B$, implying that v_A and v_B are not distinct, a contradiction. □

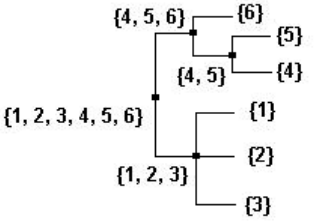

Fig. 3. Graph associated to a tree-like family. Also depicted are the sets associated to internal nodes.

Definition 6. *The tree-like family associated to the tree T, the set $\mathcal{T}_{L(T)}$ where* $\mathcal{T}_{L(T)} = \{L(T)\} \cup \{\{v\} : v \in L(T)\} \cup \{\{L(u)\} : u$ internal node of $T\}$.

Example 3. *The family* $\mathcal{T}_{L(T)} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \{1, \ldots, 10\}, \{3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}, \{5, 6\}, \{1, \ldots, 6\}, \{7, 8, 9\}\}$ *is the tree-like family associated to the tree T presented in Figure 2.*

Definition 7 (Constraints). *Let π denote a partition of a set S. We call every element of π, a block. The partition π satisfies the constraint $A \subseteq B$, where B is a subset of S, iff A is included in a block of π. Let C be a set of constraints. The partition π satisfies C if it satisfies every constraint in C. Given a tree T, and u, v, two internal nodes where u is a child of v, the constraint associated to the internal edge between u and v is $L(u) \subseteq L(v)$. We denote C_T the set of constraints for all internal edges of T.*

Example 4. *Given the set $S = \{1, 2, 3, 4, 5, 6\}$ and $\pi = \{\{1, 2, 3\}, \{4, 5, 6\}\}$ a partition on S. π satisfies the set of constraints $\{\{1, 2, 3\} \subseteq S, \{4, 5, 6\} \subseteq S, \{4, 5\} \subseteq \{4, 5, 6\}\}$.*

Example 5. *In Figure 2, the constraint $\{3, 4\} \subseteq \{2, 3, 4\}$ is associated with the edge connecting nodes x and y, and $\{7, 8, 9\} \subseteq \{1, 2, \ldots, 10\}$ to the edge connecting z and t. The set $C_T = \{\{3, 4\} \subseteq \{2, 3, 4\}, \{2, 3, 4\} \subseteq \{1, 2, 3, 4\}, \{5, 6\} \subseteq \{1, \ldots, 6\}, \{1, \ldots, 4\} \subseteq \{1, \ldots, 6\}, \{7, 8, 9\} \subseteq \{1, \ldots, 10\}, \{1, \ldots, 6\} \subseteq \{1, \ldots, 10\}\}$ is the set of constraints associated with T.*

Definition 8 (π_1 refines π_2). *For π_1, π_2 two partitions of the same set S, we say that π_1 refines π_2, denoted $\pi_1 \leq \pi_2$, iff for every pair A, B, with A a block of π_1 and B a block of π_2, the intersection set $A \cap B$ is in $\{A, \emptyset\}$.*

Example 6. *The partition $\pi_1 = \{\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}, \{9, 10\}\}$ refines the partition $\pi_2 = \{\{1, 2, 3, 4\}, \{5, 6\}, \{7, 8, 9, 10\}\}$.*

Definition 9 (π_2 embedded in π_1). *For two partitions π_1 and π_2 we say that π_2 is embedded in π_1 if π_2 is a partition of a block of π_1.*

Example 7. *The partition $\pi_1 = \{\{1, 2\}, \{3, 4\}\}$ is embedded in the partition $\pi_2 = \{\{1, 2, 3, 4\}, \{5, 6\}, \{7, 8, 9, 10\}\}$.*

Example 8. *Let S be the set with elements $\{1,2,3\}$ and C the set of constraints $\{\{1,2\} \subseteq \{1,2,3\}, \{1,3\} \subseteq \{1,2,3\}\}$. There is a contradiction between the two constraints: elements 1 and 2 cannot form a subcluster of $\{1,2,3\}$ at the same time as 1 and 3 form a subcluster of $\{1,2,3\}$. In this case the algorithm TREE-LIKE presented below can only return the partition containing single block $\{1,2,3\}$.*

The algorithm COMP presented in this paper is an extension of the procedure with the same name in [8]. Given a set of leaves S and a set of constraints C on S, the procedure COMP(S, C, π) iteratively builds π, a partition of S satisfying C. The same paper contains a procedure TREE$(S, C, \mathcal{T}_S)$ which constructs a family of embedded partitions $\pi_1, \pi_2 \ldots, \pi_k$ and returns $\mathcal{T}_S = \bigcup_{i=1}^{k} \pi_i$, where $\mathcal{T}_S$ is a tree-like family on S. Our algorithm TREE-LIKE below is an extension of procedure TREE.

3 Algorithms

We use Card(A) for the cardinality of set A.

Algorithm 1

Algorithm: COMP(*S, C,π)*
Input: A set of leaves *S*, and a set of constraints *C* on *S*.
Output: π, a partition of *S* that satisfies the set of constraints *C*.
begin
1. *count* ← 1
5. Let π^0=*{S₁, ..., Sₖ}* , *k=Card(S)*, be the initial partition of *S*, where S_i= *{s_i}, 1≤ i ≤k*, consists of a single element (leaf) of *S*.
3. $\pi \leftarrow \pi^0$
4. **repeat** while *count* ≠0
5. *count* ← 0
6. **for each** constraint *A⊆B* in *C* **do**
7. Find all S_t in π such that $A \cap S_t \neq \emptyset$.
8. **if** there are at least two such S_t , **then**
($A \cap S_t \neq \emptyset$ for only one S_t means that $A \subseteq S_t$ and the constraint is already satisfied.)
9. *count* ← 1
10. $S_{new} \leftarrow \bigcup_{\substack{t=1,\\ S_t \cap A \neq \emptyset}}^{|\pi|} S_t$ // A ⊆ S_{new}
11. Delete from π all S_t with $A \cap S_t \neq \emptyset$.
// they will be replaced by their union.
12. Add S_{new} to π
13. **if** *Card(π)* =1 **then**
14. **if** *Card (S)* >1 **then** // there is a contradiction in *C*
15. **return** π
16. **end** (if)
17. **end**(if)
18. **end** (if)
19. **end** (for)
20. **end** (repeat)
21. **return** π
end

π returned by COMP(S, C, π) is a partition of S. The algorithm starts with π, a family of single element blocks, which is clearly a partition of S. During steps 6-12 one or more blocks of π are deleted and their union is added to the family as a new block S_{new}. Clearly $A \subseteq S_{\text{new}}$ and the number of blocks in the partition diminishes by at least one at each pass of steps 6-12. Consequently, π returned by algorithm COMP(S, C, π) is a partition which satisfies the constraint set C. If π has only one block and the set of constraints is not empty then the set of constraints is satisfied but it contains at least one contradiction.

Algorithm 2

Algorithm: TREE-LIKE(S, C, $\mathcal{T}_S$)
Input: A set of leaves S, and a set of constraints C on S.
Output: $\mathcal{T}_s$, a family of embedded partitions, such that each partition satisfies its corresponding constraints of C.
begin
1. $\mathcal{T}_s \leftarrow \emptyset$
Let $\pi^0=\{S_1, ..., S_k\}$, $k=Card(S)$, be the initial partition of S,
where $S_i=\{s_i\}$, $1 \le i \le k$, consists of a single element (leaf) of S.
2. **if** $C=\emptyset$, **then** $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \pi^0$ // partition having each block with cardinal 1
3. **return** $\mathcal{T}_s$
else
4. Print the elements of S
5. Print the elements of C.
6. COMP(S, C, π)
Suppose $\pi = \{S_1, ..., S_m\}$, $m \ge 1$
7. $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \pi$
8. **if** $|\pi|=1$ **then**
9. Print "Possible error: The partition has only one block"
10. **return** $\mathcal{T}_s$
else
11. **for** i=1 **to** m **do**
12. $C_{Si} \leftarrow \{A \subseteq B: A \subseteq B \in C$ and $B \subseteq S_i\}$
(C_{Si} is the subset of C concerning only the elements of S_i.
so that Card(C_{Si}) ≤ Card(C).)
13. TREE_LIKE(S_i, C_{Si}, $\mathcal{T}_i$)
14. $\mathcal{T}_s \leftarrow \mathcal{T}_s \cup \mathcal{T}_i$
15. **end** (for)
16. **end** (if)
17. **end** (if)
18. **return** $\mathcal{T}_s$
end.

Suppose TREE-LIKE$(S_p, C_{S_p}, \mathcal{T}_p)$ and TREE-LIKE$(S_k, C_{S_k}, \mathcal{T}_k)$ are two distinct calls of TREE-LIKE during the execution of TREE-LIKE$(S, C, \mathcal{T}_S)$. The following situations may occur:

1. $S_p \cap S_k = \emptyset$, meaning that the two calls are made directly or recursively for distinct values of i in the for loop of TREE-LIKE$(S, C, \mathcal{T}_S)$;
2. $S_p \cap S_k \neq \emptyset$. Without loss of generality we suppose TREE-LIKE$(S_p, C_{S_p}, \mathcal{T}_p)$ calls directly or recursively TREE-LIKE$(S_k, C_{S_k}, \mathcal{T}_k)$. Hence, the partition $\mathcal{T}_k$ is embedded in the partition $\mathcal{T}_p$.

If there are no contradictions in the set of constraints then at each recursive call of TREE-LIKE the input contains at least one constraint less than the caller

TREE-LIKE. Hence, the last call on the recursive stack of executions is one with the constraints set empty. Consequently, the last partition returned is the partition where each block has cardinality 1.

If there are contradictions (Example 8) in the set of constraints, the algorithm will return a partition with a single block S' and will not continue the recursion. A message saying "Possible error. The partition has only one block" is displayed. To complete the algorithm, we also consider a subsequent partition for S' with single elements blocks.

Property 2. *$\mathcal{T}_S \cup \{S\}$ is a tree-like family.*

Proof: S belongs to the family. Every single leaf set is in the family. Consider A, B two distinct sets in the family that are blocks in two partitions built during the execution of the algorithm. Then $A \cap B \in \{A, B, \emptyset\}$.

4 Implementation

The algorithms COMP(S, C, π) and TREE-LIKE$(S, C, \mathcal{T}_S)$ were implemented using Java. We use a memory bit to represent each element of S and consequently we realized all operations on sets using the Bitwise operators of the BitSet Class. The result is a speed up of the running time of the program.[1]

Let k be the cardinal of the set of constraints C. The running time of the algorithm COMP(S, C, π) is $O(k)$ The set of constraints C is usually scanned only once. If there are many scans to be done, then the number of repetitions is no greater than k.

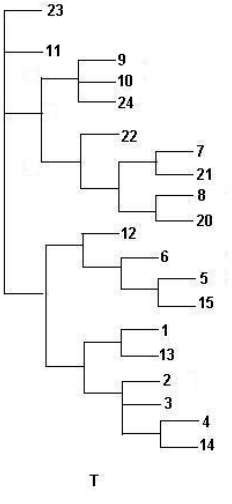

Fig. 4. T, supertree compatible with trees T_1, T_2 and T_3

[1] The program may be obtained from `mkant@yorku.ca`

The algorithm TREE-LIKE$(S, C, \mathcal{T}_S)$ is recursive. The total numbers of calls equals the number of internal nodes in the final supertree. Let p be the cardinality of the set of leaves in S. The worst-case running time of TREE-LIKE is then $O(kp)$. In the present implementation, at each recursive call of TREE-LIKE the program outputs the corresponding set of elements and constraints. Hence, the user may identify the contradictions in the set of constraints and choose which constraints to retain and which ones to discard.

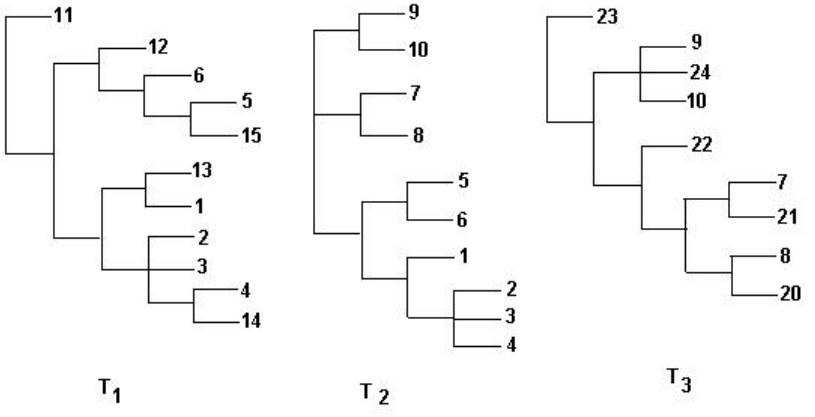

Fig. 5. Trees T_1, T_2 and T_3

5 Applications

5.1 Supertree for a Family of Trees with Refinement

Consider the trees T_1, T_2 and T_3 (Figure 5) where $L(T_1)$=$\{1, 2, 3, 4, 5, 6, 11, 12, 13, 14, 15\}$, $L(T_2) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, $L(T_3) = \{7, 8, 9, 10, 20, 21, 22, 23, 24\}$, where $T_1|L(T_1)\cap L(T_2) \ni T_2|L(T_1)\cap L(T_2)$ and $T_3|L(T_2)\cap L(T_3) \ni T_2|L(T_2)\cap L(T_3)$. The three trees (presented in Figure 5) have overlapping sets of leaves. Trees T_1 and T_3 include subtrees which are refinements of subtrees of T_2.

The tree T in Figure 4 is the supertree compatible with T_1, T_2 and T_3. It refines all of them: $T|L(T_2) \ni T2, T|L(T_1) \ni T_1$, and $T|L(T_3) \ni T_3$.

5.2 Supertree for a Family of Trees with Contradiction

Koop et al. [12] computed the independent phylogenies for the $\epsilon-, \gamma-, \eta-, \delta-$, and $\beta-$globin genes of groups of primates depicted in Figure 6. We obtained the supertree in Figure 7 for this family of phylogenies.

In the $\beta-$globin phylogeny {Chimpanzee, Gorilla} $\subseteq$ {Chimpanzee, Gorilla, Human} while in the $\eta-$globin phylogeny {Chimpanzee, Human}$\subseteq${Chimpanzee, Gorilla, Human}. This is a contradiction. Hence, the supertree computed by TREE-LIKE for the five phylogenies presents an internal node for the set {Chimpanzee, Gorilla, Human} with three children (Figure 7 (a), internal node x). To resolve this, we could choose, for example, to retain the constraint from

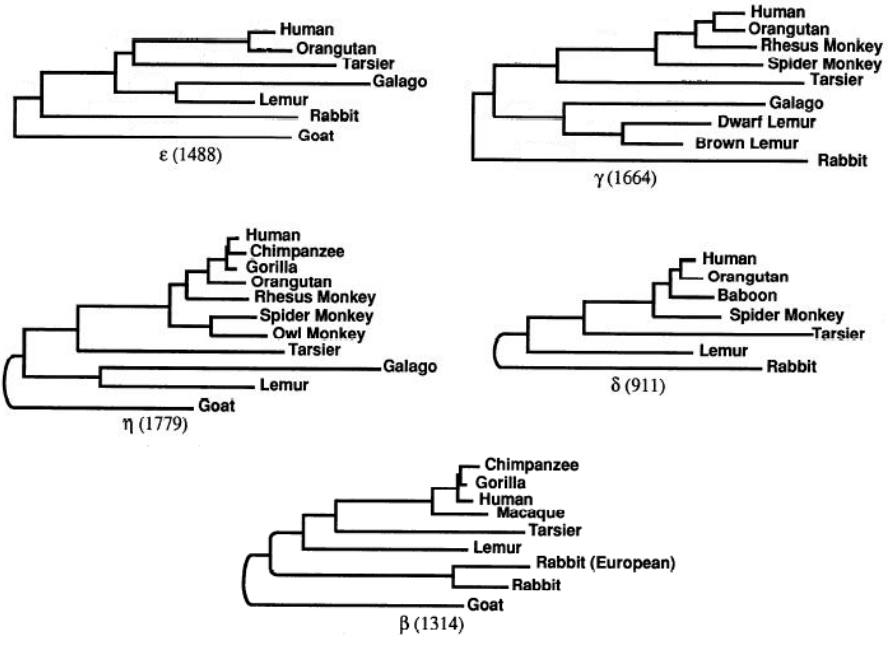

Fig. 6. Phylogenies computed for $\epsilon-, \gamma-, \eta-, \delta-$, and $\beta-$globin genes of groups of primates. From [12]. Used with permission.

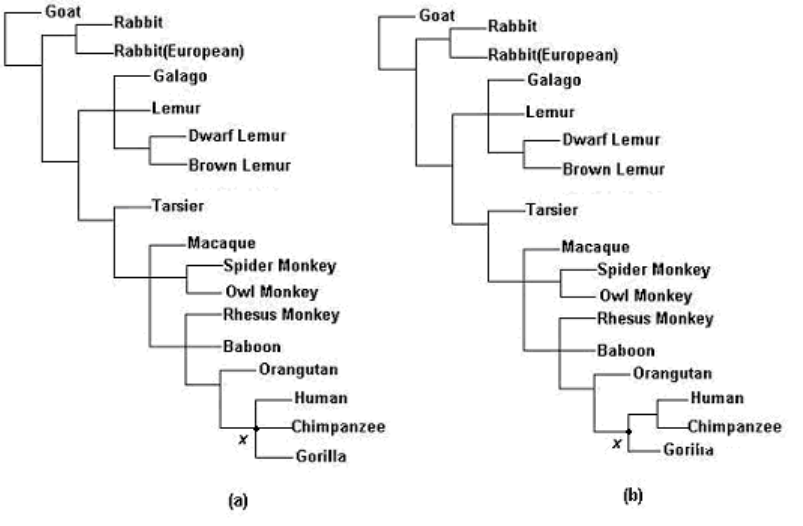

Fig. 7. Supertree for the phylogenies for $\epsilon-, \gamma-, \eta-, \delta-$, and $\beta-$globin genes of primates

the $\eta-$globin phylogeny and to discard the other. The output supertree would then have, from node x, a branch for Gorilla and another for the group Chimpanzee and Human (Figure 7 (b)).

5.3 Supertree for a Set of Constraints on Clusters

Consider the set of elements $L^* = \{1, \ldots, 14\}$ and the family of constraints $C^* = \{\{1,2\} \subseteq \{1,2,3\}, \{1,2,3\} \subseteq \{1,2,3,4,5,6\}, \{7,8\} \subseteq \{1,2,3,4,5,6,7,8\}, \{1,2,3,4,5,6,7,8\} \subseteq \{1,2,3,4,5,6,7,8,9,10\}, \{11,12\} \subseteq \{11,12,13,14\}, \{11,12,13,14\} \subseteq \{1,2,3,4,5,6,7,8,9,10,11,12,13,14\}\}$; $A \subseteq B$ meaning that either the elements in subset A are closer each other then the elements in B, or that the cluster A is a distinct subcluster of the cluster B of L^*.

The supertree T^* corresponding to the set of constraints C^* is presented in Figure 8(a). The set of constraints identifies $\{1,2,3,4,5,6,7,8\}$ as a subcluster of $\{1,2,3,4,5,6,7,8,9,10\}, \{1,2,3\}$ as subcluster of $\{1,\ldots 6\}, \{7,8\}$ as subcluster of $\{1,\ldots,8\}$. There is no specific information for elements 4, 5, and 6. Consequently, those three elements are presented as children of internal node x, Figure 8(a). Adding the constraint $\{4,5,6\} \subseteq \{1,2,3,4,5,6,7,8,9,10\}$ induces the algorithm to build the supertree in Figure 8(b).

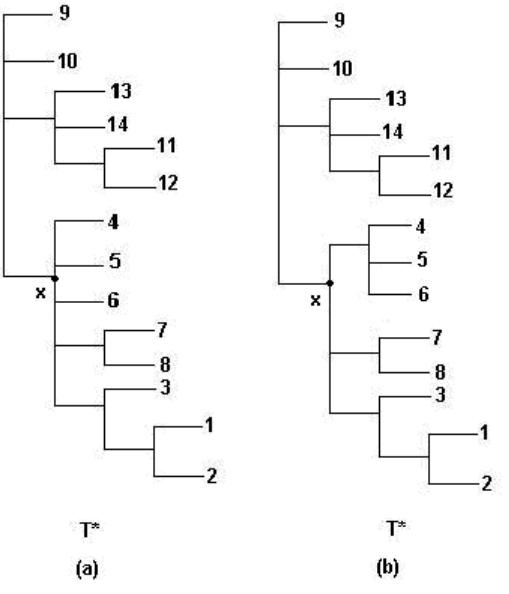

Fig. 8. Supertree obtained for a set of constraints on clusters

6 Conclusion

The input of our algorithm may be constraints derived from cluster analyses and/or from a family of small rooted trees with overlapping leaf sets or simply

from previous biological knowledge. It outputs a multifurcating supertree, which is the unique tree compatible in a strong sense with all the constraints, and which does not depend on any ordering of these constraints, either in the input or in preprocessing. At each recursive call of TREE-LIKE the program outputs the set of elements and the set of associated constraints. This information identifies contradictory constraints and helps the users decide if they must be retained, further resolving the tree, or not.

Semple and Steel [6] have written about desirable properties of supertrees. Among these are:

1. the method runs in polynomial time
2. the resulting supertree displays all rooted binary supertrees shared by all of the trees in the family
3. if the family is compatible, the resulting supertree displays each of the trees in the family
4. (a) the resulting supertree is independent of the order in which the members of the family are listed
   (b) if we rename all species and then apply the method to this new collection of input trees, the resulting supertree is the one obtained by applying the method to the original collection of trees, but with the species renamed as before.
5. the method allows a possible weighting of the trees in the family.

Of these properties, our method satisfies 1-4. If there is no contradiction among trees, property 5 is moot. If there is a contradiction, our provision for manual intervention is a way of assigning priorities among the input trees, though it is not a pre-assigned numerical weight.

Daniel and Semple [13] have used the notion of "semilabeling" in order to build supertrees that are based on incompatible trees. While this may well be advantageous from one point of view, it nevertheless has the disadvantage that the supertree can no longer display all the trees in the family on which it is based.

Acknowledgments

Research supported in part by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) to MK and to DS. DS holds the Canada Research Chair in Mathematical Genomics.

References

1. Cracraft, J., Donoghue, M.: Assembling the Tree of Life. Oxford University Press, Oxford (2004)
2. Bininda-Emonds, O.R.P.: Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life, vol. 4. Kluwer Academic, Dordrecht (2004)

3. Hall, B.G.: Phylogenetic Trees Made Easy: A How-to Manual. 3 edn., Sinauer Associates, Inc (2007)
4. Bininda-Emonds, O.R.P., Gittleman, J.L., Steel, M.A.: The (Super) Tree of Life: Procedures, Problems, and Prospects. Annual Review of Ecology and Systematics 33, 265–289 (2002)
5. Gordon, A.D.: Consensus Supertrees: The Synthesis of Rooted Trees Containing Overlapping Sets of Labeled Leaves. Journal of Classification 3, 335–348 (1986)
6. Semple, C., Steel, M.: A Supertree Method for Rooted Trees. Discrete Applied Mathematics 105, 147–158 (2000)
7. Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. Journal of Discrete Algorithms 5, 564–591 (2007)
8. Constantinescu (Kant), M., Sankoff, D.: Tree Enumeration Modulo a Consensus. Journal of Classification 3, 349–356 (1986)
9. Margush, T., McMorris, F.R.: Consensus n-Trees. Bulletin of Mathematical Biology 43, 239–244 (1981)
10. Kant, M.: The Synthesis of Two Compatible Rooted Trees in a Rooted Supertree by an Algorithm on Sets. In: Proceedings of Fifth International Conference on Computing and Information (1993)
11. Day, W.H.E., Johnson, D.S., Sankoff, D.: The Computational Complexity of Inferring Rooted Phylogenies by Parsimony. Mathematical Biosciences 81, 33–42 (1986)
12. Koop, B.F., Tagle, D.A., Goodman, M., Slightom, J.L.: A molecular view of primate phylogeny and important systematic and evolutionary questions. Molecular Biology and Evolution 6, 580–612 (1989)
13. Daniel, P., Semple, C.: A class of general supertree methods for nested taxa. SIAM Journal of Discrete Mathematics 19, 463–480 (2005)