On building and comparing trees Application to supertrees in phylogenetics

Vincent Berry

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ DE MONTPELLIER II

— SCIENCES ET TECHNIQUES DU LANGUEDOC —

Mémoire

pour obtenir le diplôme

d'Habilitation à Diriger des Recherches

Spécialité :	**Informatique**
Formation Doctorale :	**Informatique**
École Doctorale :	**Information, Structures, Systèmes**

On building and comparing trees

Application to supertrees in phylogenetics

par

Vincent BERRY

Présentée et soutenue publiquement le 8 décembre 2008 devant le Jury composé de :

Alain Denise, Professeur, Université Paris-Sud	Rapporteur
Manolo Gouy, Directeur de Recherches, CNRS	Rapporteur
Mike Steel, Professeur, Université de Canterbury (NZ)	Rapporteur
Alain Guénoche, Directeur de Recherches, CNRS	Examinateur
Olivier Gascuel, Directeur de Recherches, CNRS	Examinateur
Jean-Claude König, Professeur, Université de Montpellier 2	Examinateur
Gilles Caraux, Professeur, ENSAM	Invité

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ DE MONTPELLIER II

— SCIENCES ET TECHNIQUES DU LANGUEDOC —

Mémoire

pour obtenir le diplôme

d'Habilitation à Diriger des Recherches

Spécialité :	**Informatique**
Formation Doctorale :	**Informatique**
École Doctorale :	**Information, Structures, Systèmes**

On building and comparing trees

Application to supertrees in phylogenetics

par

Vincent BERRY

Présentée et soutenue publiquement le 8 décembre 2008 devant le Jury composé de :

Alain Denise, Professeur, Université Paris-Sud	Rapporteur
Manolo Gouy, Directeur de Recherches, CNRS	Rapporteur
Mike Steel, Professeur, Université de Canterbury (NZ)	Rapporteur
Alain Guénoche, Directeur de Recherches, CNRS	Examinateur
Olivier Gascuel, Directeur de Recherches, CNRS	Examinateur
Jean-Claude König, Professeur, Université de Montpellier 2	Examinateur
Gilles Caraux, Professeur, ENSAM	Invité

Université de Montpellier II (Université des Sciences et Techniques du Languedoc)
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)

Référence
BERRY, Vincent « *On building and comparing trees* ».
Habilitation à diriger des recherches, Université de Montpellier II, 8 décembre 2008.

Thanks

The work presented in this manuscript would never have been possible without a number of people that I wish here to thank wholeheartedly.

Christophe Paul provided me with varying musical CDs to wait for the very few times he was available for discussing ongoing and future research ideas. Nonetheless, I owe him the best investigation sessions I did at the Lirmm.

Just next, come David Bryant and Charles Semple. I really enjoyed working with these NZ colleagues, that is, once the cultural and accentual frontier was crossed! I thank them for very stimulating long-term collaborations.

V. Ranwez was an invaluable colleague over the last years, with which I take a great pleasure to discuss ideas. His calm and patience in sorting out wild ideas is of great help. I sincerely rejoice to continue working with him in the next coming years on a common research project.

I thank E.J.P. Douzery for numerous discussions on practical data. I also thank him for leading me and my programs into the jungle of phylogenetic softwares and users, where no established format or conception can survive long.

F. Chevenet showed me how discussing research can invariably end with some fine wine to sip. G. Caraux also brought some bottles along, but more than that, a useful and constant moral support.

V. Lefort provided useful technical support for conceiving web pages where some softwares we developed are available and can also be executed online.

O. Gascuel provided me both with a clear vision of what bioinformatics is and should be, and with an invaluable financial support to me and some PhD students when I was in between two funded projects. I also thank him for his advices in the handling of this manuscript.

I thank M. Steel, A. Denise and M. Gouy for having looked into the depths of this manuscript and writing thorough reviews on my research work.

I have a big debt toward PhD students I supervised, namely Alexis, Sylvain, Céline and Philippe. I really wish you will all obtain a permanent position to enjoy the pleasures of research.

Last, but not least, I'd like to have a few words for my family. This work cost them a non-negligible number of sacrifices and would never have been possible without their full support.

Abstract

The research work presented in this manuscript is of algorithmic kind: it is mainly composed of polynomial, fixed parameter and approximation algorithms, while hardness results are also mentioned.

This work is about building and comparing labelled trees. These objects find application in different areas, but notoriously in phylogenetics, where they represent evolutionary relationships of organisms or sequences.

Most of this work can be considered as investigating solutions to so-called *supertree* problems. Supertrees are large trees built by a dynamic programming approach from smaller trees. For instance, the latter are gene trees from which a comprehensive tree on many living species is to be built, such as the *Tree of Life*.

First definitions are introduced, then a part of the manuscript is dedicated to quartet tree building methods. The next part details tree comparison methods, mainly variants of the maximum agreement subtree method. Next follows a part on supertree problems in all generality. The manuscript ends with a report of the research plan for the next few years.

Several journal papers illustrating the material described in this manuscript are adjoined in appendix.

Contents

Part I

Introduction to the manuscript

Chapter 1

Is that all supertree stuff?

The work presented in this manuscript is the result of ten years of research on the construction and comparison of trees with application to phylogenetics, the bioinformatics field that aims at elucidating evolutionary relationships between living organisms. The words *Computational Biology* are well suited to describing the kind of work presented in this manuscript. In other words, obtained results are always based on a non-negligible number of theoretical arguments. However, these results were almost always obtained with the aim of answering questions arising as a result of problems encountered in phylogenetics. The practical side prevailed throughout this material, being both the motivation and the final step, via tests on real or simulated data.

As I look back at this work, almost every part of it seems to fall under the *supertree* category. Supertrees are trees on a large set of species that are built from smaller trees by combining the topological traits of these source trees. The first part of my research concerned quartet methods (presented in part II of this manuscript) which go almost beyond the limits of the “supertree" term since their input consists of source trees on four taxa only. The second part (presented in Part III) was motivated by an extension of the maximum agreement subtree consensus method (MAST) to the context of supertrees where input trees usually have different leaf sets. In the process, we obtained results on the original MAST problem but the basic motivation remained supertrees. The most recent part of my activity (presented in part IV) has been focused on tackling problems of supertrees originating from the systematic biology community. The sizes of parts detailing past works differ due to the different amounts of work (e.g. in number of years) they represent.

In all these works, the emphasis is on reliability, i.e. inference of trees whose parts are strongly supported by data. This reliability is usually enforced by combinatorial arguments.

The manuscript ends with a description of ongoing research and a short description of the research I will conduct on multigene families in the next four years, through the ANR funding we just obtained.

Chapter 2

Basic notions and definitions

This chapter first explains basic but important terms used throughout this manuscript. Then a number of basic definitions from computer science and phylogenetics are recalled. Those relevant to several parts of the manuscript are presented here, while those specific to a part are detailed in due place. Some definitions are standard in the fields but reviewing them allows us to see associated notations used in this manuscript. For additional definitions and properties on objects considered here, I refer the reader to the excellent book of C. Semple and M. Steel [SS03]. Lastly, the basic terms of parameterized complexity are explained, as several results detailed in the manuscript are affiliated with this way of dealing with NP-hard problems.

2.1 Allowing computer science and phylogenetics to communicate

Phylogenetics and computer science sometimes have different words to denote the same notions. I indicate here the most frequent synonyms that are used in this manuscript.

Words that biologists might not know.

- the **degree** of a node in a tree is the number of its neighbors, while the degree of a tree is the maximum degree observed for its nodes.
- an **edge** is a link between two adjacent nodes in a tree (or in a graph), which is sometimes referred to as **branch** in phylogenetics.
- Computer scientists usually formalize considered tasks to be fulfilled in a **problem** statement, indicating the input and the sought output. Each possible input for the given problem is also called an **instance**. Computer scientist like to classify problems in different **classes** depending on the intrinsic hardness of the problems. The well-known P and NP classes

are well-known examples but a large number of classes exist, some specifically related to approximation and parameterized versions of usual problems. A small dozen of problem classes are mentioned in this manuscript. The reader is referred to computer science manuals on complexity theory for their definition.

- a **reduction** from one problem to another problem is a transformation of any instance of the first problem into an instance of the second problem. When this transformation is done in polynomial time, this highlights that the second problem is at least as hard as the first one.

Words that computer scientists might not know.

- a **taxon** (plural **taxa**) is a studied object that can be anything like a species, a subspecies, a genre, family name, or even a molecule.
- a **phylogeny** or **phylogenetic tree**, is a tree displaying the evolutionary relationships of several taxa. Internal nodes of the tree represent ancestral taxa while its leaves represent contemporary or extinct taxa. The association of taxa to leaves – and sometimes internal nodes – of a phylogeny is indicated by labels given to these nodes, with each label representing a different taxa.
- a **clade** in a rooted tree is a set of leaves under the same internal node. It represents all taxa belonging to the same taxonomic group, e.g. mammals.
- a **multifurcation**, or **multifurcating node**, in a tree is an internal node whose degree is larger than that of a node in a binary tree, i.e. larger then 3.
- a **fully resolved** tree is a binary tree. This stems from the usual hypothesis that evolution is mainly a binary process. A **partially resolved** tree is then a tree containing at least one multifurcation.

2.2 Supertrees

Definition and applications. It often happens that several phylogenies with different but overlapping taxa sets have to be combined within a single phylogeny, representing a summary of these **source** phylogenies. The resulting tree is called a **supertree** as it is built from trees and usually contains more taxa than each input tree.

Supertree methods are thus tree-building methods that take trees as input, as opposed to distance or character-based methods that work from primary data. But what is the interest in an indirect approach when a tree could be inferred directly from a combination of the primary data? Well, depending on the ways the data was collected, supertrees are sometimes the easiest way to obtain a large phylogeny, and sometimes they even are the only possible approach. For instance,

one might want to obtain a large phylogeny for a group of species by recycling data used by previous authors. However, these data can sometimes be hard to find (e.g. lost on an old computer that was used at the time of the study[1]) and the only usable vestiges of the study are the phylogenies visible in figures included in the paper, that are, in the best cases, referenced in a database such as TreeBASE [SBB+93]. Another situation where supertrees are the incontrovertible solution is when the primary datasets to be combined are heterogeneous, e.g. morphological traits, molecular sequences of different sorts, distances between genomes.

Supertree methods can also be of use in broad-scale studies based on molecular data where several genes are considered as primary data but span different sets of species. This means that gene sequences are unavailable for some taxa. Such a situation arises because the given genes have not yet been sequenced for all the studied species (due to sequencing cost or to the difficulty of sequencing some species) or because the gene no longer exists for these species. In this situation, there are two possible approaches, one is the supermatrix approach that concatenates for each species all available sequences, putting specific signs such as question marks when the sequence is missing. The obtained sequences for all species are then combined into a large matrix that is analyzed by a classical tree-building method such as maximum likelihood. The other approach is to infer a gene tree for each gene and then to combine the gene trees in a comprehensive tree through a supertree method. Depending on the percentage of missing data, a supermatrix or supertree approach is best suited.

There are many different applications of supertree methods in phylogenetics, including the construction of extensive taxonomies for large groups of species such as mammals [BECJ+07] and the construction of the *Tree of Life* [SPH98, BE04], for which supertrees are currently the most promising technique – if not the only hope! – thanks to their *divide-and-conquer* strategy. Indeed, the supertree approach is an example of such a strategy: the computational effort is first focused on small quantities of data to obtain intermediary objects (source trees) that are then combined within a larger one (the supertree).

Supertree methods. The goal of a supertree is to combine the topological information of the source trees into a single comprehensive picture, and there are several ways of doing this as the information present in different source trees can be incompatible. Thus, an important issue concerning supertree methods is how they handle topological conflicts, i.e. different arrangements of the same labels (i.e. taxa) among source trees. Such conflicts can, for instance, arise from long branch attraction, from model misspecification in the inference of individual source trees, or from contradictory phylogenetic signals among the primary datasets, e.g. due to paralogy or horizontal gene transfers (HGT). Supertree methods fall into three categories depending on the way they handle conflicts.

The first suite of methods are just able to detect conflicts among source trees, returning an

[1]Let me mention here that this is not pure rhetoric: I really obtained this answer once from a biologist whose primary data interested me! But computer scientists should laugh aloud here when considering the very small number of experimental past studies for which they are still able to provide the full results (not only those summarized in published papers) and programs used to obtain these results.

“incompatible" message in such cases. The pioneering methods that belong to this category are BUILD [ASSU81] and the strict consensus supertree [Gor86] (we should maybe also cite the SQM quartet method here [ESSW97]). Although they are important milestones, these methods appear “*of limited use. As most systematics know, phylogenies usually conflict with one another*” [BE04, p4].

The second suite of methods handle conflicts among input trees in a liberal way: they apply a *voting* procedure. In order to extract their main phylogenetic signal, source trees are asked to vote on various parts of the phylogeny to be inferred, with the most supported candidates being elected and composing the output supertree. Voting methods are said to *resolve* conflicts [TW03]: for each conflict, they use some optimization criteria to make a decision in favor of one of the topological alternatives. Most conflicts among input trees are expected to be resolved because relationships displayed by the supertree are guided by source topologies on the basis of the weight of evidence. This strategy is best suited for studies that aim at extracting congruent phylogenetic signals from a set of source trees with moderately to strongly supported clades, and then plays the role of a meta-analytic synthesis of the input trees. This can be applied for instance to gene trees, where the main phylogenetic signal can be contrasted in some source trees by alternative signals, i.e. slightly different evolutionary histories due to duplication/loss or HGT events. A large number of voting methods have been proposed, that can mainly be presented on the basis of the reconstruction principle they adopt:

- The most widespread voting method is Matrix Representation with Parsimony (MRP) whereby clades of each source tree are encoded as binary characters of a matrix that is then analyzed with the maximum parsimony criterion to obtain the supertree [Bau92, Rag92]. Analyzing this binary encoding of source topological information with other tree-building criteria leads to MRP variants such as Matrix Representation with Flipping [CEFBS02, CDE$^+$03] and Matrix Representation with Compatibility [RR04].

- Other voting methods, such as MinCut (MC) [SS00] and ModifiedMinCut (MMC) [Pag02], extend the BUILD algorithm [ASSU81], i.e. encode rooted source trees in a graph on the basis of the rooted triples they contain. The graph is then progressively decomposed to get the clades of the supertree in a top-down way. When conflicts hinder the decomposition, the graph is cut by removing the least number of supported relationships.

  The MC (and hence MMC) method are, in some sense, extensions of the Adams consensus [Ada72] as shown by [SS00]. Therefore, multifurcations in the output supertree are not to be interpreted in the usual way: subtrees hanging from a multifurcating node are not considered as monophyletic (any fully resolved supertree where they are interleaved is acceptable), the important point is that the internal structure of each subtree is respected. Under this interpretation, a multifurcation represents a much wider range of fully-resolved phylogenies than with the usual interpretation and is harder to interprete in a phylogenetic context[2].

[2]In particular, this means that simulation studies on supertree methods that use the Robinson and Foulds distance to evaluate the performance of MC or MMC are misleading.

- Quartet methods proposed in the computer science field in the 90s (detailed in Part. II of this manuscript) can be considered as supertree methods. However, they have mainly been designed to deal with a single dataset, e.g. sequences, from which all possible quartets can be computed. By contrast, usual supertree methods combine input trees that often have little taxonomic overlap. Thus, in the supertree context, quartet methods are required to build trees from incomplete to sparse quartet sets. This rules out a number of quartet methods detailed in Part. II. Piaggio *et al* proposed a quartet method specifically dedicated to such a situation [PTBE04]. Their method refines a method by J. Willson, related to quartet cleaning methods detailed in Chap. 4 of this manuscript.

- Average Consensus [LC97] and Super Distance Matrix [CBDG06] methods implement the voting approach in an alternative way. They average the initial distance matrices, converted from source characters or valued topologies, into a *superdistance* matrix; a tree-building distance-based approach is then used to infer a supertree from the matrix.

By combining the topological information of different source trees, any supertree method is expected to generate *novel* clades, i.e. clades not present in any input tree alone [Pur95]. When source trees conflict, it is possible that some novel clades contradict some source trees. Unfortunately, voting methods can infer novel clades that are contradicted by each and every source tree [GP02, Gol05, CSW06]. The frequency of this extreme phenomenon is still debated, [BE03] reporting on the basis of simulations that it is not very frequent for MRP, while [Gol05] shows selected case studies where "*this situation is clearly not very unlikely*".

The third suite of methods handle conflicts among input trees in a conservative way. They adopt a *veto* philosophy: the phylogenetic information of every source topology is to be respected, and the supertree is not allowed to contain any clade that a source tree would vote against. These methods *remove* conflicts [TW03] because they either propose multifurcations in the supertree [GP02], or prune taxa whose position varies in the source trees [BN04]. In this framework, the supertree should not retain a single branching pattern for a subset of taxa when topological alternatives are present in the source trees. The full agreement required by veto methods provides an unambiguous phylogenetic framework that is, for instance, well suited for taxonomic revisions. More specifically, such a conservative approach may be applied to automatically build or update parts of the Tree of Life (`http://tolweb.org`). In this application, each source tree can be thought of as representing the current state of knowledge on a taxonomic group (but possibly also contains taxa from neighboring groups). As all input trees are assumed to be equally reliable, in case of a conflict, it is not a desirable feature that the supertree chooses one of the topological alternatives. We rather expect it to indicate the incongruences (by including multifurcations, excluding taxa or outputting extra information). Unanimity among source trees as displayed by veto supertrees would also seem useful for other applications such as molecular dating of speciation events or character evolution investigations. Several supertree methods have been proposed over the years [Gor86, GP02], including those discussed in chapters 7 and 9 of this manuscript.

For a wide panorama of research on supertrees as of 2004, I refer the reader to the well-known book of O. Bininda-Emonds [BE04].

2.3 Notations and basic definitions on trees

Trees and collections of trees.

Definition 2.1 *Let T be a singularly labeled tree, $L(T)$ denotes the leaf set in T, as well as the label set of its leaves. The **size** of a tree T is its number of leaves and is denoted $|T|$.*

*A **collection** is a set of trees that are either all rooted or all unrooted. Given a collection $\mathcal{T}$, $L(\mathcal{T})$ denotes the set of labels in $\mathcal{T}$, i.e. $\cup_{T\in\mathcal{T}}L(T)$. The size of the collection, denoted $|\mathcal{T}|$, is the number of trees it contains.*

A **star** tree on L is the trivial tree containing only one internal node linked to leaves bijectively labeled with elements of L.

Let u be a node in a rooted tree R, $S(u)$ denotes the complete subtree of R rooted at u. Given two nodes u and v in a rooted tree R, $u < v$ means that u is a proper ancestor of v and $u \leq v$ means that u is either a proper ancestor of v, or v itself.

The following relationships between nodes of a tree are extremely common and useful in designing algorithms on rooted trees. In phylogenetics, it is also referred to as the most recent common ancestor relationship.

Definition 2.2 *Given a rooted tree R, the **least common ancestor** (or **lca**) of a set of leaves $L \subseteq L(R)$ is the single node u such that $u \leq \ell$ for all $\ell \in L$, and $v < u$ for any other node v that is also an ancestor of every leaf in L. The lca of L in R is denoted $lca_R(L)$. More particularly, the lca of any pair $\{\ell,\ell'\} \subseteq L(R)$ is denoted $lca_R(\ell,\ell')$.*

Tree restrictions.

Definition 2.3 *Given a set L of labels and a tree T, the **restriction** of T to L, denoted $T|L$, is the tree obtained in the following way: take the smallest induced subgraph of T connecting leaves with labels in $L\cap L(T)$, then remove any degree two (non-root) node to make the tree homeomorphically irreducible. If $\mathcal{T}$ is a collection of trees, then define $\mathcal{T}|L = \{T|L : T \in \mathcal{T}\}$.*

See trees U, U' in Figure 2.1 for an example. Note that for any tree T and any two label sets L,L', $(T|L)|L' = T|(L\cap L') = (T|L')|L$.

Isomorphism and refinement relationships between trees.

Definition 2.4 *Two trees T, T' are **isomorphic**, denoted $T = T'$, if and only if there is a graph isomorphism $T \mapsto T'$ preserving leaf labels (and the root if both trees are rooted). Given two trees T, T', T is **homeomorphically included** in T' if and only if $T = T'|L(T)$.*

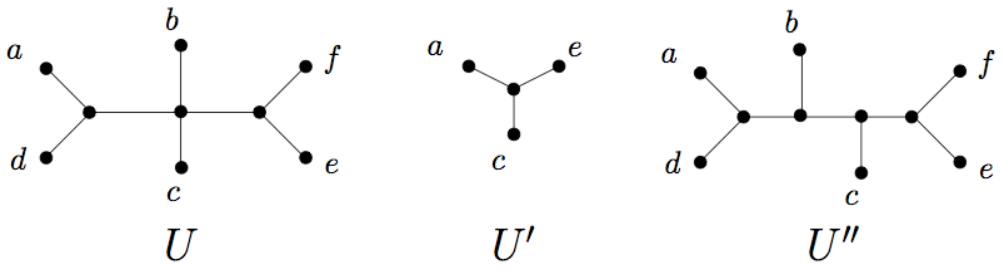

Figure 2.1: Three unrooted trees to illustrate the restriction and refinement relationships. A tree U, a tree U' such that $U' = U|\{a,c,e\}$ and a tree U'' such that $U'' \trianglerighteq U$.

Definition 2.5 *A tree T **refines** a tree T', and we write $T \trianglerighteq T'$, whenever T can be transformed into T' by collapsing some of its internal edges (*collapsing *an edge means removing it and merging its extremities). See Figure 2.1 for an example. More generally, a tree T refines a collection $\mathcal{T}$, denoted $T \trianglerighteq \mathcal{T}$, whenever T refines all T_i's in $\mathcal{T}$.*

When considering a set of trees with different leaf sets, the preceding definition can be extended [SS03]:

Definition 2.6 *Let T be a tree with leaf set L, let L' be a subset of L and T' be a tree with leaf set L'. We say T **displays** T' whenever $T|L' \trianglerighteq T'$. A tree T displaying any tree T' in a collection $\mathcal{T}$ is said to display $\mathcal{T}$.*

Note that definitions 2.4, 2.3 and 2.5 apply in the case where $\mathcal{T}$ is a set of rooted trees, or alternatively a set of unrooted trees.

Isomorphism and compatibility issues between rooted trees can also be expressed in terms of ancestor relationships. The following statements are directly derived from the definitions given previously and are implicitly or explicitly used in a number of works (for instance [GW02, PT04]).

Observation 2.1 *Let R and R' be two rooted trees on the same leaf set L.*
The following two statements are equivalent:
(i) *R and R' are isomorphic*
(ii) *$\forall \ell, \ell', \ell'' \in L$, the two following equations hold*

$$lca_R(\ell,\ell') < lca_R(\ell,\ell'') \iff lca_{R'}(\ell,\ell') < lca_{R'}(\ell,\ell'') \tag{2.1}$$

$$\text{and} \quad lca_R(\ell,\ell') = lca_R(\ell,\ell'') \iff lca_{R'}(\ell,\ell') = lca_{R'}(\ell,\ell'') \tag{2.2}$$

Moreover, the following two statements are equivalent:
(iii) *R refines R'*
(iv) *$\forall \ell, \ell', \ell'' \in L$, the following holds:*

$$lca_{R'}(\ell,\ell') < lca_{R'}(\ell,\ell'') \Rightarrow lca_R(\ell,\ell') < lca_R(\ell,\ell'') \tag{2.3}$$

Compatibility.

Compatibility is the notion used to state whether objects of some kind can be combined together in a given structure. The tree, rooted or unrooted, is usually the structure examined in phylogenetics, but different kinds of networks have also been considered [BD92, HNR$^+$98, Gam] as they enable us to model recombination events such as hybridization or lateral gene transfers. In this manuscript, we will only consider the compatibility in tree structures.

The compatibility of different kinds of objects can be considered: bipartitions, unrooted or rooted trees with the same or different label sets, quartets, etc. Below we formally define compatibility for the kinds of objects that will be considered in the next chapters. We first consider the compatibility of bipartitions in unrooted trees.

Definition 2.7 *A set B of bipartitions on the same label set X is **compatible** (or* tree-compatible*) if and only if there is an unrooted tree T with $L(T) = X$ such that for each bipartition $\sigma_1|\overline{\sigma}_1 \in B$ there is an edge in T that, when removed, splits T into two trees T' and T" with $L(T') = \sigma_1$ and $L(T") = \overline{\sigma}_1$.*

P. Buneman gave the following well-known characterization of such a set:

Lemma 2.1 ([Bun71])

- *Two bipartitions $b_1 = \sigma_1|\overline{\sigma}_1$, $b_2 = \sigma_2|\overline{\sigma}_2$ are compatible if and only if at least one of $\sigma_1 \cap \sigma_2$, $\sigma_1 \cap \overline{\sigma}_2$, $\overline{\sigma}_1 \cap \sigma_2$, $\overline{\sigma}_1 \cap \overline{\sigma}_2$ is empty.*
- *A set B of bipartitions is **compatible** if and only if every pair of bipartitions $b_1, b_2 \in B$ are compatible (i.e. we only need to check the compatibility of subsets of two elements to decide on the compatibility of the whole set).*

Now, consider the case of collections of trees:

Definition 2.8

- *A collection $\mathcal{T}$ of leaf-labeled trees on the same label set L is said to be **compatible** if and only if there is a leaf-labeled tree T on L that refines every tree T' in $\mathcal{T}$.*
- *A collection $\mathcal{T}$ of leaf-labeled trees with different label sets is said to be **compatible** if and only if there is a tree T on $L(\mathcal{T})$ that displays any tree T' in $\mathcal{T}$.*

2.4 Building stones of trees

In phylogenetics, it is well-known that trees of arbitrary size can be described by subtrees of small bounded size. This decomposition is used a number of times in this manuscript, with a whole part being even devoted to them. Tirples and fans are the building stones of rooted trees.

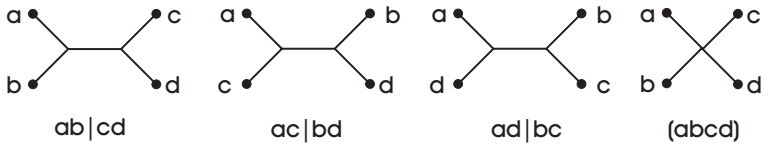

Figure 2.2: The four possible quartet topologies for $\{a,b,c,d\}$.

Definition 2.9 *A **rooted triple** (or simply* triple *or* resolved triple*) is a binary rooted tree on three leaves.*

*A **fan** (also called* unresolved triple*) is a rooted tree on three leaves with only one internal node. On three given distinct leaves a,b and c, there are three possible rooted triples, denoted $ab|c$, respectively $ac|b$, respectively $bc|a$, depending on their innermost grouping of two leaves (ab, respectively ac, respectively bc). The only one possible fan on this set of leaves is denoted (a,b,c).*

Let R be a rooted tree. For any set $\{a,b,c\}$ of three leaves in $L(R)$, $R|\{a,b,c\}$ is either a rooted triple or a fan. We define $\mathsf{rt}(R)$, respectively $\mathsf{f}(R)$, as the set of rooted triples, respectively fans, of tree R induced by the 3-leaf subsets of $L(R)$. Given a collection $\mathcal{T}$ of rooted trees, $\mathsf{rt}(\mathcal{T})$, resp. $\mathsf{f}(\mathcal{T})$ denotes the union of the set of rooted triples, resp. fan sets, of trees in $\mathcal{T}$.

The basic building stones of unrooted trees are quartet and stars on four leaves:

Definition 2.10 *A **quartet** is a binary unrooted tree on four leaves. Alternatively, an unrooted tree on four leaves can be a star tree. Given four distinct leaves a, b, c and d, there are three possible quartets, respectively denoted $ab|cd$ (corresponding to the binary tree where the path from a to b does not intersect the path from c to d), $ac|bd$ and $ad|bc$, and only one possible star denoted (a,b,c,d).*

Let U be an unrooted tree. For any set q of four leaves appearing in U, $U|q$ is either a quartet or a star. We define $\mathsf{q}(U)$, respectively $\mathsf{s}(U)$, as the set of quartets, respectively stars, of U induced by 4-leaf subsets of $L(U)$. Given a collection $\mathcal{T}$ of unrooted trees, $\mathsf{q}(\mathcal{T})$, resp. $\mathsf{s}(\mathcal{T})$ denotes the union of the set of quartets, resp. 4-leaf star sets, of trees in $\mathcal{T}$.

Figure 2.2 shows the quartets and fan corresponding to a set of four leaves.

As for bipartitions, tree compatibility can also be defined for partial objects such as triples and quartets[3]. But a stronger notion also exists in this case, indicating when such a set completely fits to a tree:

Definition 2.11 *A triple set R, resp. a set quartet set Q, is*

- ***compatible** (or* tree-consistent*) if and only if there is a tree T such that $R \subseteq \mathsf{rt}(T)$, resp. $Q \subseteq \mathsf{q}(T)$.*
- ***tree-like** if and only if there is a tree T such that $R = \mathsf{rt}(T)$, resp. $Q = \mathsf{q}(T)$.*

[3]they are *partial* objects in the sense that they each indicate a relationship on only three or four elements of a set of arbitrary size.

2.5 Parameterized complexity

When facing an NP-hard problem, there are two main alternatives to obtain practical algorithms: proposing polynomial time approximation algorithms, or either exact algorithms whose running time is fast for some instances of the considered problem. The latter solution is usually investigated by distinguishing a given parameter characterizing the instances, and trying to obtain an exact algorithm whose running time is practical for instances having small values for this parameter. A number of results reported in this manuscript fall into this category.

Parameterized complexity theory [DF99, FG06] formalized this behaviour and proposed various difficulty classes sorting computational problems according to the kinds of parameterized results that can be obtained. More precisely, given a computational problem, let n denote the size of an instance and let p be a parameter describing these instances. For example, when an instance is composed of molecular sequences, n is the total number of characters composing the input sequences and p can be the size of the alphabet (i.e. 2 for binary characters encoding purines/pyrimidines, 4 for DNA sequences, 20 for proteins). Parameterized complexity theory makes a distinction between: (i) a problem solvable in $O(2^p n)$ time, (ii) a problem solvable in $O(n^p)$ time, and (iii) a problem which is NP-hard for any value of p larger than some constant. In case (i), the corresponding algorithm remains practical for large n values, provided that p is small. In case (ii), the algorithm is still practical for the smallest p values. In case (iii), the problem is not easier for any instance when considering parameter p.

The *fixed-parameter tractability* concept has been introduced to deal with case (i). A problem is said to be fixed-parameter tractable (fpt) if there is an algorithm that solves the problem on an instance of size n in time $O(f(p)n^c)$, where f is any function of the parameter p, and c is a constant independent of p. In most cases, f has exponential growth. The above definition naturally extends to problems involving a combination of several parameters.

Tools are also available to distinguish problems that specifically fall into case (ii) above. Parameterized complexity classes and parameterized reduction enables one to show that a parameterized problem is unlikely to be fpt. The ground complexity class is that of fpt problems, denoted FPT. The theory defines several other complexity classes, which are conjectured to properly contain the FPT class. A studied problem is shown to be hard for one of these classes by a parameterized reduction from an already classified problem, and rules out the possibility of an fpt algorithm (under some complexity-theoretic assumption). These classes are called W[1], W[2],... and we have FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots$. We refer the reader to [DF99, FG06] for formal definitions of these classes.

Over the last decade, this approach has gained increasing popularity in computer science and in particular in computational biology, where parameters taking small values can often be distinguished. In phylogenetics, a number of results have been recently obtained on the parameterized complexity of NP-hard problems (see [GNT08] for a survey).

Part II

Quartet methods

Chapter 3

Setting the stage

3.1 Introduction to quartet-based methods

The quartet paradygm. Several phylogenetic reconstruction methods relying on quartets have been proposed. The combinatorial justification for such methods is that if all quartets are available and have the topology of the estimated tree, then this tree is uniquely determined and can be efficiently constructed. This remains true in some cases where only a moderate number of quartets are available [Ste92, SS03]. Of course, it is unlikely that we are given a fully correct set of quartets. Anyhow, any quartet set Q can be compared to a general tree T, as Q can be put face to face with $\mathsf{q}(T)$. Quartet-based methods (hereafter referred to as *quartet methods*) proceed from this mere principle. They are designed as a two-step procedure, with the first being to infer quartets separately from primary data (molecular sequences, morphological traits, distances between studied taxa, etc), and the second step being to build a tree whose quartet set corresponds to the input quartets according to some decision or optimization criterion. The quartet paradigm for computing phylogenetic trees is an example of the well-known divide-and-conquer approach: trees are first computed on subproblems of small size, and are then combined to obtain a solution to the full problem. As such, it had its hour of glory in the 1990s. At that time, no smart algorithm like PhyML [GG03] or RaxML [SLM05] was available to compute trees on more than ≈ 40 taxa using computationally intensive criteria such as maximum likelihood applied to sophisticated models of evolution. However, such a task was completely manageable on trees of four taxa and this allowed to infer trees of ≈ 100 taxa while relying on an underlying elaborate criterion.

Tractability of the general problem. As mentioned above, quartet sets obtained from practical data contain several erroneous quartets, whereby a tree consistent with all given quartets may not exist. The natural optimization task is then to seek the tree that satisfies (i.e. contains) a maximum number of input quartets. However, this is not a tractable problem as M. Steel showed in his seminal 1992 paper that deciding whether a given quartet set is compatible is already an NP-hard problem [Ste92]. As a direct consequence, deciding whether there is a tree that satisfies at least x

input quartets is also NP-hard. In 1999, we have shown that the problem remains NP-hard even if the input quartet set is complete [BJK+99]. Several authors have proposed heuristic [CV06] and exact algorithms, either exponential [BdCG+98] or FPT [GN03], for the corresponding optimization problem.

3.2 The rise of quartet methods in phylogenetics

In 1995, the only known papers on quartets were that of H.-J. Bandelt and A. Dress [BD86], describing various relationships between trees and quartets, and that of M Steel [Ste92], who focused on combinatorial aspects of combining quartets into trees, such as knowing the minimum number of quartets needed to identify a tree or showing the NP-hardness result mentioned above. At that time, I started working on quartets with the idea of providing a quartet-based algorithm to reconstruct phylogenetic trees. For this purpose, I focused on the $||^*$ relationship [BD86], which has the interesting property of corresponding to a perfectly tree-like part that exists in any quartet set, even one that is incompatible. In short, computing this relationship amounts to computing the largest tree-like subset of an input quartet set. No algorithm was provided in [BD86] for computing this relationship. Solving this problem had been conjectured to be an NP-hard problem [BG91], and the bad news on the NP-hardness of deciding quartet compatibility [Ste92] only reinforced this opinion. However, tree-likeness is a stricter variant than compatibility, thus there was a slight chance that the problem was tractable. Eventually, I found out that this could be computed not only in polynomial time, but also in linear time, i.e. $O(n^4)$, where n is the number of objects addressed in the input quartet set.

At that time, the only known method for building trees that explicitly referred to quartets was the AddTree algorithm [ST77]. This distance-based agglomerative algorithm only used quartets as an intermediate formalism to recode a distance matrix and then decide which subtrees to group at each agglomerative step. However, the quartet set changed regularly to reflect changes in the distance matrix, whose dimension decreased by one after each agglomerative step. In contrast, *quartet-based* methods take a quartet set as input and, as such, do not change this set during the tree-building process. Note that a specific quartet inference process is sometimes associated with a quartet method.

In 1996-97, no less than three quartet methods originating from independent works appeared at the same time in phylogenetics:

- the **Quartet Puzzling** (QP) method, from a German team [SvH96]. The QP method first estimates the probability that each possible quartet is correct on the basis of the Bayes theorem. Then these values are considered as weights for quartets, with the second phase of the method being to infer a tree on the basis of this complete set of weighted quartets. This heuristic process starts by choosing a random order on taxa. Then it chooses the heaviest quartet for the first four taxa into which it inserts each successive taxon in turn, attaching the new leaf to an edge of the current tree so as to optimize a quartet-based score. Since the quality of the result depends on the ordering of taxa, QP uses a large number of random input

orderings and computes the majority consensus of all trees found. Thus, QP seeks to return a tree in which every edge appears in more than half of the tentative trees, i.e. is reliable to some extent. The first implementation of the method was requiring $O(n^5)$ running time with a high constant (the authors recommend performing 1000 runs), but can be accelarated to $O(n^4)$ as noted by several authors (e.g. [RG01]), though with the same leading constant.

- the **Short Quartet Method** (SQM) issuing from an international collaboration at a DIMACS meeting [ESSW97]. The SQM method was rough – it either returned a fully resolved tree or no tree at all, and hence was later shown to perform poorly in simulations – but the paper came with an interesting proof that a simple quartet inference process would lead the SQM method to *converge fast*, i.e. to recover the correct tree with high probability from molecular sequences that grow only polynomially in the number of leaves, for all model trees with edges having bounded mutation probabilities. Intuitively, this nice property is obtained by ensuring that the method uses only "close" relationships (smallest values in the distance matrix) to determine the tree. The running time of SQM is $O(n^4 \log^2 n + n^2 k)$, where k is the length of the molecular sequences from which quartets are inferred.

- the **Q* method** [BG97] I proposed on the basis of the $||^*$ relation of [BD86]. It takes as input a *fully determined* quartet set inferred through any quartet inference process such as maximum likelihood (ML), neighbor joining (NJ), or ordinal quartet method (OQM) [Kea98]. A fully determined quartet set is one where one and only one quartet is available for all $O(n^4)$ combinations of four input taxa. From such a set where quartets are considered as unweighted, it proposes a tree such that the quartets corresponding to each edge are all in the input set. No need to say that with such an heavy combinatorial constraint it usually proposes partially resolved trees. It's running time is linear, i.e. $O(n^4)$. This method is detailed below in section (4.1).

A constant in these first quartet methods was the conservative approach they adopted, maybe starting from the observation that the quartet inference processes usually propose some erroneous quartets. Thus, the first quartet methods all aimed at inferring trees containing only reliable edges. QP is less stringent than SQM, while Q^* pushes the conservatism to an extreme point, while still being more flexible than SQM in its ability to infer some edges even when insufficient quartets are available for other edges.

Between 1997 and 2001, under the name *quartet cleaning* (QC), several works in which I took part proposed new quartet methods relying on combinatorial arguments in the line of the Q^* method but being less stringent, hence inferring more resolved trees. These studies were based on an initial idea of P. Kearney [JKL98] and conducted with many different collaborators (see section 4.2).

Starting from 1998, David Bryant and I also discovered that the Q^* method had two equivalent methods in the world of distance-based methods. The first one is called the **Buneman tree** since it is briefly described in a paragraph of an old work of P. Buneman originating from the classification field [Bun71]. It allows us to compute a set of compatible bipartitions, i.e. an unrooted tree, from a distance matrix. The second one is called **Apresjan clustering**, after the name of the

Russian mathematician D.J. Apresjan and was proposed even earlier [Apr66]. It enables to obtain a collection of compatible clusters, i.e. a rooted tree. The correspondence between distance and quartet worlds was further increased when in 1999 Vince Moulton and Mike Steel investigated an extension of the Buneman tree, called the **refined Buneman method** [MS99], that corresponds to part of the QC methods. This is called being timely! As I heard quite early about this work through a technical report, the same year we were able to propose a faster algorithm to compute the refined Buneman tree [BB99]. Later on, we extended this algorithm to a whole family of clustering methods for which we provided fast algorithms [BB01] (see section 4.2).

Below, we first detail the Q^* method, then QC methods generalizing it, and finally discuss the future of quartets in phylogenetics.

Chapter 4

Contributions to the quartet world

4.1 The Q^* method

Defining Q^*. Given a fully determined quartet set Q on a set S of objects (e.g. taxa), we consider here the problem of finding the maximum tree-like subset of Q, denoted Q^*. The following characterization of Q^* can be obtained from the definition of the $||^*$ relationship given in Bandelt and Dress [BD86]:

Definition 4.1 *Let Q be a quartet set and B^* be the set of bipartitions $b = \sigma|\overline{\sigma}$ such that $\mathsf{q}(b) \subseteq Q$, then $Q^* = \bigcup_{b \in B^*} \mathsf{q}(b)$.*

B^* is a compatible set of bipartitions, due to the fact that $\mathsf{q}(b) \in Q$ is required for any $b \in B^*$ and that Q contains at most one quartet topology for each four-object set. Tree-like subsets of Q on the ground set S bijectively correspond to elements in B^*. As B^* is a set of compatible elements, the tree-like subsets of Q form a lattice having as unique maximum element the set Q^* corresponding to the complete set B^*.

Note however that Q^* is usually not the maximum *compatible* subset of Q, since it is not even a maximal compatible subset of Q: $\forall q \in Q - Q^*$, $\{q\} \cup Q^*$ is compatible. Moreover, we can easily find counter-examples showing that Q^* is not always contained in the maximum subset of compatible quartets.

Computing Q^*. Obtaining B^*, T^*, Q^* form one another can be done in linear time. The algorithm I designed focuses on computin B^* from the input quartet set Q. It first does this for a trivial number of objects, then progressively transforms B^* to account for new objects considered, until all objects in S have been accounted for. More precisely, for the trivial case of four objects, B^* is readily obtained: it contains the four trivial bipartitions on these elements plus possibly the bipartition $\{x,y\}|\{z,t\}$, with $\{x,y,z,t\} = \{1,2,3,4\}$ (when $xy|zt \in Q$). At step i, an object $x_i \in S$ not yet considered is taken into account and the B^* set of this step is initialized with the trivial bipartition

$\{x_i\}|\{x_1,\ldots,x_{i-1}\}$. Then for each bipartition $b=\sigma|\overline{\sigma}$ in the previous set B^*, the algorithm checks whether $b'=\sigma\cup\{x_i\}|\overline{\sigma}$ and $b''=\sigma|\overline{\sigma}\cup\{x_i\}$ qualify to be in the new B^* set. As stated before, this only requires checking whether $(\mathsf{q}(b')-\mathsf{q}(b))\subseteq Q$ and $(\mathsf{q}(b'')-\mathsf{q}(b))\subseteq Q$, respectively. This procedure stops after all objects of S have been processed.

In [BG97] we showed that this algorithm can be implemented to have $O(n^4)$ running time since it is possible to consider only once each quartet in the process: the tree T^* is built together with B^* and information is propagated along its edges according to a postorder and a preorder traversals to detect the position where a new object x_i can be inserted as well as the former edges that have to be collapsed due to missing quartets on x_i.

Convergence rate. Additionally, we showed that, when using the four point method (FPM) to infer quartets of Q, the convergence rate of the method is at worst polynomial when the maximum evolutive distance between two taxa is bounded, which is usually the case in practice. This means that sequences from which quartets are inferred need only to have polynomial length in the number of taxa. More precisely, we have the following result:

Theorem 4.1 ([BG97]) *Under the Cavender-Farris model of evolution, the probability that the Q^* method recovers the entire topology of an estimated tree T is at least*

$$1-n^2e^{-f^2e^{-4d}k/2}$$

where f is the length of the smallest edge in T. Equivalently, if we suppose k characters evolve along the edges of a phylogeny T under the Cavender-Farris model, then $T^*=T$ with probability at least $1-\varepsilon$ $(\varepsilon>0)$ if

$$k>\frac{2\ln(\frac{n^2}{\varepsilon})e^{4d}}{f^2}\quad .$$

Thus, the difficulty comes from short edges of the estimated tree. This is explained by the fact that edges can be found only when the ancestral sequence evolving from one endpoint of the edge to the other one is prone to substitutions – the number of such events is proportional to the amount of time the sequence evolved along the edge, i.e. to the length of the edge. When substitutions actually occurre during this laps of time, the presence of the edge can be suspected by comparing sequences observed for the leaves of the tree (namely, sequences belonging to the subtree originating from one endpoint of the edge share common traits that sequences from the symmetric subtree do not have).

Since the inference of the different edges is independent for the Q^* method, we can obtain the following result:

Theorem 4.2 ([BG97]) *Under the Cavender-Farris model of evolution, the probability that the Q^* method recovers an edge e is at least*

$$1-n^2e^{-l(e)^2e^{-4d}k/2}.$$

The theorem extends to more general stochastic evolution models, such as the generalized Jukes Cantor model [TN84].

Generalizing the problem. When the set Q of considered quartets is not complete, it can be shown that deciding whether Q is tree-like is solvable in polynomial time, even if Q contains several resolutions for the same quartet. However, what is the complexity of obtaining the largest tree-like quartet subset in such a case? In other words, does the problem considered in this section remain solvable in polynomial time?

4.2 Quartet cleaning methods

As already mentioned, the quartet inference step in quartet methods is not error prone. As a result, the input set Q of quartet topologies sometimes contain quartets differing from those present in the estimated tree T. A quartet $\{a,b,c,d\}$ is a **quartet error** if $ab|cd \in Q$ but $ab|cd \notin \mathsf{q}(T)$. When the quartet set Q is only a rough estimation of $\mathsf{q}(T)$, quartet methods will infer a tree T' that is unlikely to be a good estimate of T. In fact, several early quartet methods are very sensitive to quartet errors. For instance, a single quartet error can prevent the Q^* and SQM methods to recover the (complete) true evolutionary tree. Hence, at the time these methods were proposed, it was of prior importance to develop procedures to improve the accuracy of quartet topologies inferred from primary data.

There are two approaches for improving the quartet accuracy. The first is to develop better quartet topology inference methods, which requires skills on models of sequence evolution and statistics to propose for instance more accurate character-based methods than those previously used. [Wil99a, LWT99] followed this way but – as far as I know – the efficiency of their methods has not yet been experimentally reported. The second alternative is to develop methods that detect and correct quartet errors in Q by comparing different subsets of quartets. This process is called *quartet cleaning* and have been investigated since the late 90s, through the impulsion of P. Kearney.

In, [BJK$^+$99] we presented two polynomial-time quartet cleaning algorithms. One of them was proven to be optimal in its ability to correct quartet errors whereas the other is shown to be robust to the distribution of quartet errors in the true evolutionary tree. We also performed an extensive simulation showing that i) quartet errors are common, regardless of the quartet topology inference method used, hence establishing the need for quartet cleaning algorithms; ii) the quartet cleaning algorithms we presented are very effective in detecting and correcting quartet errors, in most studied cases dramatically improving the accuracy of Q. Some details are given below on all these results.

Definitions Define the quartet error $\{a,b,c,d\}$ to be **across edge** e if e is on the joining path of $\{a,b,c,d\}$ in T. Similarly, define the quartet error $\{a,b,c,d\}$ to be **across vertex** v if v is on the joining path of $\{a,b,c,d\}$ in T. These definitions permit the assignment of quartet errors in Q to

edges/vertices of T.

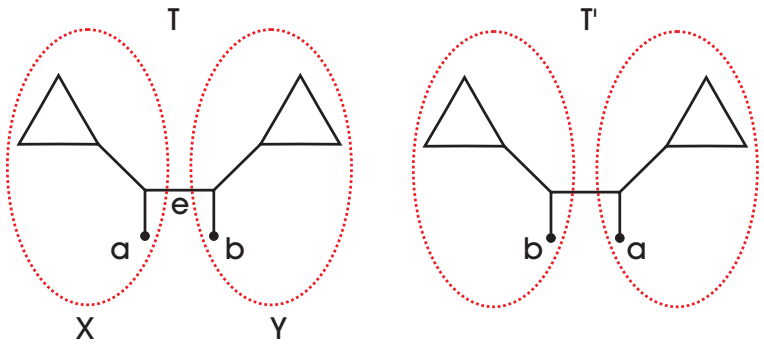

Figure 4.1: An extreme case of subtrees surrounding an edge e. This leads to establishing cleaning bounds.

Let $e = (X,Y)$ be the bipartition induced in T as depicted in Figure 4.1. Observe that $\mathsf{q}(T)$ and $\mathsf{q}(T')$ differ by quartets of the form $ax|by$ where $x \in X$ and $y \in Y$. It follows that $|\mathsf{q}(T) - \mathsf{q}(T')| = (|X|-1)(|Y|-1)$. If half of the quartets of the form $\{a,b,x,y\}$ with $x \in X$ and $y \in Y$ have quartet topology $ax|by$ in Q and the other half have quartet topology $bx|ay$ in Q then no quartet cleaning algorithm only looking at quartet across e can guarantee that quartet errors across e can be corrected. Under the basic principle of trying to satisfy a maximum number of input quartets, $(|X|-1)(|Y|-1)/2$ is then the upper bound of quartet errors that an edge can have across it to have the guarantee of being recovered. This motivates the following formulations of quartet *edge* cleaning:

Local A quartet edge cleaning algorithm has **local cleaning bound** b if it corrects all quartet errors across any edge with fewer than b quartet errors across it.

Global A quartet edge cleaning algorithm has **global cleaning bound** b if it corrects all quartet errors in Q if each edge of T has fewer than b quartet errors across it.

Analogous definitions apply to local and global *vertex* cleaning algorithms by requiring the same constraints on the three edges connected to an internal vertex.

Local edge/vertex cleaning is more robust than global edge/vertex cleaning since it can be applied to an edge/vertex independently of the number of quartet errors across other edges/vertices. This is a significant feature especially when some edges/vertices have a high number of quartet errors across them. In contrast, global cleaning algorithms are applicable only if all edges/vertices satisfy the cleaning bound.

The example from Figure 4.1 illustrates that edge/vertex cleaning bounds should not be constant but depend on the bipartition sizes. Hence, an edge $e = (X,Y)$ would have an edge cleaning bound that depends upon $|X|$ and $|Y|$. In particular, the example demonstrates that the optimal edge cleaning bound is $b_{opt} = (|X|-1)(|Y|-1)/2$.

A quartet cleaning algorithm A simple global edge cleaning algorithm can be obtained by a bottom-up approach:

Algorithm Global-Clean(Q)
1. Let $R := S$.
2. For every pair of rooted subtrees T_1 and T_2 in R
3. Let A denote the leaf labels of T_1 and T_2, i.e. $L(T_1) \cup L(T_2)$.
4. If $|\mathsf{q}(A, S-A) - Q| < (|A|-1)(|S-A|-1)/2$ then
5. Create a new tree T' that contains T_1 and T_2 as its subtrees.
6. Let $R := R - \{T_1, T_2\} \cup \{T'\}$.
7. Repeat step 2 until $|R| = 3$.
8. Connect the three subtrees in R at a new vertex and output the resulting unrooted tree.

A straightforward implementation of Global–Clean results in a time complexity of $O(n^2 \cdot n^4) = O(n^6)$, since each quartet can be checked at most $O(n^2)$ times. However, a careful use of quartets and of adequate data structures enabled us to factorize tests performed between the different analyzed bipartitions and to reduce the complexity of the algorithm down to the optimal $O(n^4)$ running time.

Local cleaning is a more difficult algorithmic problem, as edges across which too much errors are present might hinder the recovery of the other ones. A first polynomial-time algorithm was proposed in [JKL98], but this algorithm had a suboptimal cleaning bound and the polynomial of its running time had very high degree, so that this algorithm was only of theoretical interest. In [BJK$^+$99] we where able to design an $O(n^7)$ local vertex cleaning algorithm with cleaning bound $(|X|-1)(|Y|-1)/4$. Later on, we proposed a more practical $O(n^5)$ edge cleaning algorithm with optimal bound [BBJ$^+$00], then [DVW02] obtained an $O(n^7)$ vertex cleaning algorithm with optimal cleaning bound.

Empirical evaluation of the cleaning algorithms. Experiments on randomly generated 10-taxa trees were performed to measure both the need for cleaning algorithms and their efficiency. Results showed that:

- quartet sets inferred by regular methods (NJ, ML, OQM) almost always contain incorrect quartets. The number of quartet errors decreases when sequence length increases, as expected, and when mean edge-length decreases. However quartet errors remain significant even for sequences of 2000bp and mean edge-length 0.0025: still more than 50% of the datasets contain quartet errors in this case. This clearly shows that quartet errors concern a non-marginal number of trees to be estimated.

- global edge and local vertex quartet cleaning algorithms can clean significant numbers of trees and vertices per tree, respectively, under moderate mean edge–lengths and large sequence lengths. Both algorithms decrease the number of quartet errors significantly under a wide variety of conditions. For instance, when resorting to OQM to infer quartets and mean edge–length is 0.1 while sequence length is 200, the increase in accuracy is approximately 25%. When the mean edge–length is large and/or the sequence length is small, the

increase in accuracy is dramatic. For instance, for 200bp sequences and mean edge-length 0.25, where only $\approx 2\%$ input quartet sets Q are correct, the global edge cleaning algorithm enables to correct *all* errors for $\approx 82\%$ input trees. In this case, $\approx 30\%$ vertices of the estimated tree have no quartet errors across them in Q, and $\approx 69\%$ of these trees are completely error-free after the local vertex cleaning algorithm is applied. Table A.1 in appendix details results for all simulated conditions.

For more experiments on quartet cleaning see also [SJWMV03] whose findings are reported at the end of this chapter.

Recovering all correct bipartitions The experiments related above showed that in some cases the correct bipartitions (X,Y) had more quartet errors across them with respect to Q than allowed by the $(|X|-1)(|Y|-1)/2$ bound detailed above. Thus, if we want to recover all true edges, we have to be less stringent on the number of quartet errors. Doing so, leads to infer more bipartitions, but at the risk of obtaining an incompatible set of bipartitions. For this reason, bipartitions need also to be ranked to select the most supported ones from an incompatible set. It is natural to rank bipartitions according to the number of quartet errors across them with respect to the input set Q. The **distance** from a set of quartets Q to a bipartition (X,Y) is defined to be

$$|\mathsf{q}(X,Y) - Q|.$$

Note that the number of quartet topologies in $\mathsf{q}(X,Y)$ is $|X|(|X|-1)|Y|(|Y|-1)/4$. In order to compare the support for two bipartitions, the distance function must be normalized. Let us define the **normalized distance** from Q to (X,Y) by

$$\delta(Q,(X,Y)) = \frac{4 \cdot |\mathsf{q}(X,Y) - Q|}{|X| \cdot (|X|-1) \cdot |Y| \cdot (|Y|-1)} \quad .$$

When (X,Y) is *trivial* ($|X| = 1$ or $|Y| = 1$), the normalized distance is defined to be 0. The δ measure can be used as the bipartition support function: the bipartitions with lower distances from Q are those better supported by Q.

Using the normalized distance δ defined above, we can implicitly define a bipartition list that orders all bipartitions of S by increasing distance from Q. Assuming that correct bipartitions are well–supported by Q (a claim supported by experimental results) and appear near the start of the list, the task is to generate a bipartition neighborhood of Q of the form

$$\{(X,Y) \mid \delta(Q,(X,Y)) \leq r\}$$

which is called the **closed** r**–neighborhood** of Q. When the inequality is strict it is called the **open** r**–neighborhood** of Q.

This framework nicely integrates results mentioned since the beginning of this section:

- when $r = 0$, the closed r–neighborhood of Q corresponds to those bipartitions that have 0 quartet topology differing from those of Q. This corresponds to the set of bipartitions B^* computed by the Q^* method.
- when $r = \frac{2}{|X||Y|}$ the *open* r–neighborhood of Q corresponds to cleaning algorithms with the "optimum" bound detailed previously. As a result, the *closed* $\frac{2}{|X||Y|}$–neighborhood of Q is not necessarily compatible. A good example is that given in Figure 4.1.

Widening the set of inferred bipartitions to include all correct bipartitions corresponds to computing the r-neighborhood of Q with higher values of r. For this purpose, define

$$Best(Q,m) \quad = \quad \left\{ (X,Y) \mid \delta(Q,(X,Y)) < \frac{2m}{|X||Y|} \right\}.$$

Thus the set $Best(Q,m')$ contains the set $Best(Q,m)$ for all $m \leq m'$. Note that $Best(Q,1)$ is the set obtained by local edge cleaning and the limit of $Best(Q,m)$ as m tends to zero is the set of bipartitions recovered by the Q^* method. An algorithm that constructs the set $Best(Q,m)$ is called a **hypercleaning** algorithm, indicating that m can take on values greater than 1 which corresponds to *cleaning* algorithms detailed previously in this section.

Though there is an exponential number of bipartitions on n objects, focusing on those that differ from Q by only a limited number of quartets helps a lot. As we will now see, we can obtain an fpt algorithm in m, the parameter indicating the "quartet lag" of the sought bipartitions to the set Q. This shows that this computational problem can be solved efficiently for small values of m. Experiments reported hereafter indicate that a moderate value of m is sufficient to recover all correct bipartitions.

A hypercleaning algorithm. Let $S = \{s_1, s_2, \ldots, s_n\}$, $S_k = \{s_1, s_2, \ldots, s_n\}$ and Q_k be the subset of Q induced by S_k. The hypercleaning algorithm proceeds by first computing sets of the form

$$Best_{xy}(Q_k,m) = \quad \{ \quad (X,Y) \text{ such that } x \in X, y \in Y \text{ and there are fewer than } m \text{ quartet errors across } (X,Y) \text{ involving } x \text{ and } y \quad \}$$

for all $x, y \in S$ and $1 \leq k \leq n$.These sets can be computed by a recurrence relation:

Theorem 4.3 ([BBJ$^+$00]) *If $k = 1$ then $Best_{xy}(Q_k,m) = \emptyset$. If $k \geq 2$ then*

$$Best_{xy}(Q_k,m) \subseteq L_{xy} \cup R_{xy} \cup M_{xy}$$

where

$$\begin{aligned} L_{xy} &= \{(X \cup \{s_k\}, Y) \mid (X,Y) \in Best_{xy}(Q_{k-1},m)\} \\ R_{xy} &= \{(X, Y \cup \{s_k\}) \mid (X,Y) \in Best_{xy}(Q_{k-1},m)\} \\ M_{xy} &= \{(\{s_k\}, S_{k-1})\} \end{aligned}$$

The algorithm for computing $Best_{xy}(Q_k,m)$ for all $x,y \in S$ and $1 \leq k \leq n$ follows from Theorem 4.3: For all $x,y \in S$, for k ranging from 1 to n, and for all $(X,Y) \in L_{xy} \cup R_{xy} \cup M_{xy}$, place (X,Y) in $Best_{xy}(Q_k,m)$ if there are fewer than m quartet errors across (X,Y) involving x and y.

The number of bipartitions in $Best_{xy}(Q_k,m)$, for any $1 \leq k \leq n$, is $O(n \cdot f(m))$, where $f(m) = 4m^2(1+2m)^{4m}$. There are $O(n^3)$ sets $Best_{xy}(Q_k,m)$ to construct. Constructing each set $Best_{xy}(Q_k,m)$ takes time proportional to the size of $Best_{xy}(Q_{k-1},m)$ times the complexity of testing if a bipartition has fewer than m quartet errors involving x and y. Factorizing the detection of quartet errors from Q_{k-1} to Q_k, at step k we only need to check for the $O(n)$ quartets involving x,y and s_k. It follows that the complexity of constructing all sets $Best_{xy}(Q_k,m)$ is $O(n^5 f(m))$.

Now, let

$$Best(Q_k,m) = \{(X,Y) | \delta(Q_k,(X,Y)) < \frac{2m}{(|X||Y|)}\}$$

for all $1 \leq k \leq n$. Observe that $Best(Q_n,m) = Best(Q,m)$. $Best(Q_k,m)$ can be computed thanks to the following recurrence relation:

Theorem 4.4 ([BBJ+00]) *If $k = 1$ then $Best(Q_k,m) = \emptyset$. If $k \geq 2$ then*

$$Best(Q_k,m) \subseteq L \cup R \cup M$$

where

$$\begin{aligned} L &= \{(X \cup \{s_k\}, Y) : (X,Y) \in Best(Q_{k-1},m)\} \\ R &= \{(X, Y \cup \{s_k\}) : (X,Y) \in Best(Q_{k-1},m)\} \\ M &= \cup_{x \in S_{k-1}} Best_{xs_k}(Q_k,m) \end{aligned}$$

The algorithm for computing $Best(Q_k,m)$, for $1 \leq k \leq n$ follows from Theorem 4.4: For k ranging from 1 to n, and for all $(X,Y) \in L \cup R \cup M$, place bipartition (X,Y) in $Best(Q_k,m)$ if there are fewer than $m(|X|-1)(|Y|-1)/2$ quartet errors across (X,Y). To obtain the complexity of this algorithm, we obtained the following bound on the size of each set $Best(Q_k,m)$:

Theorem 4.5 ([BBJ+00]) *The number of bipartitions in $Best(Q_k,m)$, for any $1 \leq k \leq n$, is $O(n^3 f(2m))$.*

By a similar reasoning as above, we showed that the overall complexity to compute $Best(Q,m)$ is $O(n^5 f(2m) + n^7 f(m))$. This establishes the problem of determining $Best(Q,m)$ as fixed parameter tractable. Like other fixed parameter tractable problems, it is likely that more efficient implementations can be achieved. Anyhow, from the above equations it is clear that the method can already be heavily parallelized.

As we could compute $Best(Q,m)$ and rank these bipartitions by the δ measure, we proposed a simple greedy "optimization" algorithm to produce a tree that is as resolved as possible and that

maximizes the agreement with the input set Q. More precisely, it tries to select a maximal set of compatible bipartitions (X,Y), with the smallest distance to Q, *i.e.*, minimizing $\sum_{(X,Y)} \delta(Q,(X,Y))$. For this purpose, the greedy algorithm orders bipartitions in $Best(Q,m)$ by increasing normalized distance to Q and selects a bipartition when it is compatible with all previously chosen bipartitions. The set of chosen bipartitions are then assembled into a tree output by the greedy algorithm.

A simulation study starting from a real 10-taxa tree showed that $m = 5$ was enough to recover all correct bipartitions, and that, even when $m = 2$, the hypercleaning greedy algorithm obtains systematically and significantly better results than the local cleaning algorithm. Note that the definition of $Best(Q,m)$ takes into account the size of the two parts of a bipartition. Hence, it is likely that the minimum value of m needed to find all correct bipartitions does not depend on the number of taxa in the estimated tree, but rather only on the difficulty of the reconstruction, e.g. depending on LBA phenomena.

Conclusion and future directions. The m parameter contributes to the appeal in the hypercleaning technique, in that it allows for a complete range of quartet methods, from conservative (with values of m close to 0) to optimization (large values of m). The heuristic detailed above choosing the output bipartitions among those of $Best(Q,m)$ is simplistic, and there is a real hope that a very accurate quartet method is obtained by replacing it with a more involved selection procedure. Indeed, the experiments showed that $m = 5$ allows to circumvent in 99% of the cases the set of all correct bipartitions in a set whose size is at most 7 times larger. This is already an impressive results: at the beginning we had the challenge of recovering the $O(n)$ correct bipartitions from an exponentially large set of possible bipartitions. Now we only have to decipher them from a set whose size is constant with respect to n. There are at least two ways in which the simplistic bipartition selection heuristic could be improved:

- First, the bipartitions could be ranked depending on another criteria. For instance, different quartets could contribute different weights to rank a bipartition. From the interesting results obtained by latest method relying on short quartets [SWR08], it might be interesting to give more weight to short quartets than to others to avoid LBA problems. The weighting scheme among quartets could be progressive. A weighting scheme giving better rank to the correct bipartitions would lead the selection procedure to choose more correct bipartitions at the start, hence increase the total number of them chosen overall. Indeed, every incorrect bipartition chosen impedes one or several correct bipartitions to be selected as they are mutually incompatible.

- A complementary possibility is to adopt a less naïve algorithmic procedure to select bipartitions from $Best(Q,m)$. A bipartition could be chosen not only depending on those already chosen but also on the other candidates ones. For instance, a bipartition that invalidates four fifths of the remaining ones could be given less chance than one invalidating just one fifth of them. A weighted incompatibility graph among candidate bipartitions could also be computed, in which a maximum weighted clique (or an approximation of it) would be sought. The connectivity of the vertices in this graph depends on the value of m, and it is worth ex-

ploring whether a fixed-parameter algorithm or an approximation algorithm can be designed depending on the actual value of m. Many ideas remain to be tested there.

- In 2001, S. Willson proposed an "error correcting map" (called EC) that aims at correcting quartets incorrectly inferred [Wil01]. This map has a purpose apparent to that of the quartet cleaning methods detailed here. However, EC works from a very different principle than cleaning methods and both methods could correct different quartet errors. The complementarity between EC and QC/HQC is indicated by the fact that more resolved trees are obtained when combining EC with QC/HQC. Given current results in [Wil01, pp347-348], a couple of theoretical results remain to be obtained such as showing that correcting quartets by EC before applying QC/HQC can never lead to loose some inferred splits. Moreover, a complete simulation study would be welcomed to examine the overall level of accuracy obtained by combining the two kinds of error correction methods.

4.3 Link between some quartet- and distance-based methods

Between 1998 and 2000, I worked with David Bryant on parallels between the above-mentioned quartet methods and several distance-based methods. The latter take as input a matrix of "distances" between pairs of objects, from which is built a tree whose leaves are labeled by the objects. Distance methods have been used in phylogenetics since the 80s, the age of *numerical taxonomy*, where such methods were called *phenetic* methods. Distance methods preexisted in other scientific fields such as classification, where they are often called *hierarchical clustering* methods. The aim of almost every clustering method is to cluster objects in groups of objects such that objects in a group (or *cluster*) are more similar to one another than they are to objects outside the group. Note that there is a slight but important distinction here with phylogenetic reconstruction which aims at clustering evolutionary related objects together.

Duality between clustering and splitting methods. Any "distance" method takes as input a squared or triangular matrix of values between studied objects. In fact, these values can either represent similarities or dissimilarities between objects. Methods working from similarities usually produce a collection of compatible clusters, equivalently a rooted tree, and are hence called clustering methods hereafter. Methods working from dissimilarities usually produce a collection of bipartitions or *splits*, equivalently an unrooted tree, and are called splitting methods. There is a duality between the two kinds of methods, as a method of one kind can usually be turned into a method of the other kind. Below is detailed an example we studied.

Let S be the set of studied objects. Define the **Apresjan clusters** [Apr66] of a similarity function s to be $\{A : A \subseteq S \text{ such that } \iota_s(A) > 0\}$, where $\iota_s(A)$ is the strong isolation index

$$\iota_s(A) := \min_{a,a',x} \{s(a,a') - s(a,x) :\ a,a' \in A, x \in S - A\}.$$

Define the **Buneman splits** [Bun71] of a dissimilarity function δ to be the set of splits $\{A|B : \mu_\delta(A|B) > 0\}$, where μ_δ is the strong separation index

$$\mu_\delta(A|B) = \tfrac{1}{2}\min\{(\delta(a,b)+\delta(a',b')) - (\delta(a,a')+\delta(b,b')) : a,a' \in A,\, b,b' \in B\}. \tag{4.1}$$

In [BB99] we formalized the link between the two methods. Let δ be a dissimilarity function on S. The *Farris Transform* of δ with respect to $x \in S$ is the similarity function s_x on $S - \{x\}$ with

$$s_x(a,b) = \tfrac{1}{2}(\delta(a,x) + \delta(b,x) - \delta(a,b))$$

for all $a,b \in S - \{x\}$ [FKE70]. The inverse of the Farris transform is given by

$$\delta(a,b) = s_x(a,a) + s_x(b,b) - 2s_x(a,b)$$

for all $a,b \in S - \{x\}$, and $\delta(a,x) = s_x(a,a)$ as well as $\delta(x,x) = 0$.

Theorem 4.6 ([BB99]) *Let δ be a dissimilarity function on S and let s_x denote the Farris transform of δ with respect to $x \in S$. For any split $A|B$ of S,*

$$\mu_\delta(A|B) = \min_b\{\iota_{s_b}(A) : b \in B\} = \min_a\{\iota_{s_a}(B) : a \in A\}.$$

Thus $A|B$ is a Buneman split for δ if and only if A is an Apresjan cluster of s_b for all $b \in B$, and this holds if and only if B is an Apresjan cluster of s_a for all $a \in A$. This connection leads to the suprising algorithmic result that Buneman splits of a dissimilarity function can be constructed in $O(n^3)$ time [BB99], even though the definition appears to imply that $O(n^4)$ quartets need to be considered (see equation (4.1)).

Connection betweeen distance and quartet methods. The fact that quartets appear in the definition of the Buneman method can be exploited to define a link between splitting methods and quartet methods. Indeed, the Buneman tree can be redefined in the following way: consider the *Buneman score* of a quartet $ab|cd$ with $a,b,c,d \in S$, to be defined as:

$$\beta_\delta(ab|cd) = \tfrac{1}{2}\left(\min\{\delta(a,c)+\delta(b,d), \delta(a,d)+\delta(b,c)\} - \delta(a,b) - \delta(c,d)\right) \tag{4.2}$$

The strong separation index of a split $A|B$ of S is then

$$\mu_\delta(A|B) = \min_{a,a' \in A, b,b' \in B} \beta_\delta(aa'|bb').$$

Formulated as above, the Buneman tree method can be easily generalized to the case when quartets are not necessarily weighted by their Buneman score. Given any set of quartets Q and a weighting function w for Q such that at most one quartet has positive weight for each subset of four species, the set of splits

$$\{U|V : \forall q \in q(U|V), w(q) > 0\}$$

is precisely the set of bipartitions B^* computed by the Q^* method. The gateway from dissimilarities to quartet weights provided by equation (4.2) allowed us to define a whole family of splitting methods by defining different separation indices and looking each time at the set of splits that have positive index. Depending on the strictness of the separation index, methods with a varying degree of conservatism are obtained. Each method in the family outputs a set of splits that includes those of the more conservative methods. Equivalently, the unrooted tree output by a method refines the trees output by the more conservative methods. The progression begins with the Buneman tree [Bun71], then contains the refined Buneman tree [MS99], then a generalization of quartet cleaning methods [JKL98, BJK$^+$99, BBJ$^+$00] and ends with a construction called *stable* splits, that uses the most lenient separation index.

Thanks to the duality between clustering and splitting methods, a whole family of clustering methods (with a varying degree of conservatism) was also defined starting from the Apresjan clusters [Apr66] to a construction called stable clusters [BB01]. Additionally, we were able to prove links between some of these methods and well-known methods in classification, such as the Single Linkage and Average Linkage methods.

Fast algorithms The link with these classical methods, as well as the gateways between clustering, splitting and quartet methods enabled us to obtain efficient polynomial time algorithms for most constructions, except for stable clusters and splits that we showed NP-hard to compute. These algorithms have been implemented in the well-known SplitsTree phylogenetic package [Hus98, HB06]. Table 4.1 lists the running times obtained for the various methods we considered, where splitting methods can accept either a dissimilarity function or a quartet weighting function, while clustering methods take as input either a similarity function or a triple weighting function[1].

Future direction. Sometimes, data simply does not support a tree but a more intricate structure, such as a network, e.g. due to recombination events. To build networks from distance data, Bandelt and Dress introduced the *split decomposition* method, which constructs a set of weakly compatible splits [BD92]. In fact, split decomposition is the exact network analogue of the Buneman tree construction. We already exploited this relationship to provide a faster algorithm for split decomposition than previously known [BB99]. However, the generalization of the split decomposition to a whole family of network methods refining it, i.e. outputting more splits, has not yet been considered. The only work I know of in this direction is the interesting but somewhat cryptic work of Andreas Dress on weak hierarchies [Dre97].

[1] i.e. a weighting function on rooted triples.

Construction	Input	Complexity
Strong clusters [Apr66]	similarity function	$O(n^2)$
	triple weighting function	$O(n^3)$
Buneman splits[Bun71]	dissimilarity function	$O(n^3)$
	quartet weighting function	$O(n^4)$
Clean clusters	similarity function	$O(n^3)$
	triple weighting function	$O(n^4)$
Clean splits	quartet weighting function	$O(n^5)$
Refined Buneman splits [MS99]	quartet weighting function	$O(n^5)$
Stable clusters		NP-hard.
Stable splits		NP-hard.

Table 4.1: A summary of the construction complexities for the clustering and splitting methods studied in [BB01]. Methods are ordered by decreasing degree of conservatism.

Publication(s) whose material is described in this chapter:

- Inferring Evolutionary Trees with Strong Combinatorial Evidence, **V. Berry** and O. Gascuel, *Theoretical Computer Science*, 240(2), 271-298, 2000. A less complete preliminary version of this paper appeared in Berry, V., Gascuel, O., Proceedings of COCOON'97.
- Quartet Cleaning: Improved Algorithms and Simulations, **V. Berry**, P. Kearney, M. Li, T. Jiang and T. Wareham, *European Symposium on Algorithms* (ESA'99), *LNCS*, num. 1643, 313–324, 1999.
- A practical algorithm for recovering the best supported edges of an evolutionary tree, **V. Berry**, D. Bryant, T. Jiang, P. Kearney, M. Li, T. Wareham et H. Zhang, *Symposium on Discrete Algorithms* (SODA'00), ACM press, 287–296, 2000.
- Faster reliable phylogenetic analysis, *International Conference on Computational Molecular Biology* (RECOMB'99) – best student paper, **V. Berry** and D. Bryant, ACM press, 59–68, 1999.
- A structured family of clustering methods, D. Bryant and **V. Berry**, *Advances in Applied Mathematics*, vol 27, pp 705-732, 2001.

Chapter 5

The fall of quartet methods in phylogenetics?

Quartet methods can be used in various fields as a paradigm to obtain a hierarchical classification of a set of objects under study. The application field of prime interest to me is phylogenetics and I will discuss here experimental results of quartet methods in this field.

In phylogenetics, the first methodological works on quartets mainly date from the 1990's, with many new methods being proposed around year 2000. The emergence of these new methods naturally raised the question of their accuracy to build phylogenies, relative to other methods in use.

Simulation studies. In biology, the correct, historical tree is almost never known, which makes assessing the accuracy of tree-building methods problematic. Thus, the habit in the field is to compare methods by using computationally intensive simulations where the "correct" tree is fixed. A large number of runs are repeated with varying parameters. For each run, a random model tree is chosen; sequences are evolved according to a simple evolution model from the root down to the leaves of the model tree; then sequences obtained at the leaves are collected to form a dataset given as input to the competing tree-building methods. Performances of the methods are compared on the basis of various measures. The accuracy of the methods is usually measured with the Robinson and Foulds (RF) distance [RF81]: its value is defined as the number of edges in the inferred tree that are not in the model tree (type I error) plus the number of edges in the model tree that are not in the inferred tree (type II error).

Accuracy of the inferred quartets. By studying the impact of taxonomic sampling among groups of a phylogeny to estimate, [LPVLLG93] pointed out that the accuracy of a proposed phylogeny is linked to the number of taxa it contains, with trees including more taxa being better estimated. This raised doubts on quartet methods simply because the quartet estimation process – building trees on four leaves only – is likely to produce an unlimited number of incorrect quartets.

Hence, any quartet method would be handicaped from the start. However, quartet methods are in a way quite resistant to errors in the input quartet set, as I showed experimentally [Ber97]. I studied the case of a 15-taxa tree, extracting its quartets from which I deleted p_e percent at random. The resulting incomplete quartet set was given as input to a simple quartet-based heuristic method. For $5\% \leq p_e \leq 80\%$, this method was able to find the starting tree in 100% of the times (on 200 runs). For $p_e = 95\%$, the correct tree was still found 84% of the times. Additionally, reversing (i.e. replacing by an incorrect quartet) the p_i percent of the remaining quartets at random did not make the reconstruction a lot harder. For instance, choosing $(p_e = 50\%, p_i = 33\%)$, the heuristic quartet method was still able to find the correct tree 99% of the times, and for $(p_e = 75\%, p_i = 33\%)$ its success rate was 98.3% (on 300 runs). So quartet methods can still perform well despite a lot of missing and incorrect information.

However, accuracy problems can arise when the errors are not homogeneously distributed among quartets. On real or simulated datasets, it can happen that all quartet errors point in the same direction, e.g. toward an alternative global topology than that of the phylogeny being estimated. This mostly happens when the estimated phylogeny is submitted to long-branch attraction (LBA), i.e. when there is a succession of long-short-long edges in the phylogeny. In such a case, quartets concerning a short internal edge can be systematically incorrect and consistently point toward another resolution of the edge. The whole quartet set thus leads any quartet method to infer a partially incorrect topology. This is the conclusion of [RG01] who obtained less accurate results for quartet methods in simulations where LBA arises. Their results could be explained by the fact that distance methods (such as NJ) and character methods (such as parsimony or ML), as opposed to quartet methods, could be less affected by LBA since they would first cluster taxa in the different subtrees surrounding the short problematic edge, and then choose a resolution of that edge on the basis of the information given by more than four taxa.

Accuracy of quartet methods to infer phylogenies. More generally, the accuracy of quartet methods has been compared to that of NJ a large number of times, usually on the basis of the RF distance, with mitigated results:

- [SvH96] obtained better accuracy results for QP than for NJ on an 8-taxa tree with sequences of 500 bp or 1000 bp. However, their results were criticized by [RG01] as they used a well-balanced model tree, which favors QP (as clearly demonstrated in [RG01, Table 5]).

- [Ber97] showed on 10-taxa trees that a simple quartet-based heuristic further resolving the tree output by Q^* is able to beat NJ on the RF measure when short sequences are available ($\leq$ 300 bp). But as I remember it, NJ had better RF results on longer sequences (unpublished results).

- [RG01] obtained better RF results for NJ than for QP and for the Weight Optimization (WO) quartet heuristics they presented. These results were obtained on 12-taxa trees for sequences of 300 and 600 bp and trees likely submitted to LBA.

- [SJWMV03] observed that NJ recovered on average a larger proportion of correct edges than conservative quartet methods (Q^*, QC, QP), on 10- to 40-taxa trees with sequences containing up to 2000 bp.

In defense of quartet methods, it can be noted that most of these studies did not use the most accurate quartet inference methods available at the time, e.g. the OQM [Kea98] and the Higher Order Parsimony method proposed by S. Willson to specifically reduce LBA problems [Wil99b]. It is however not clear that this factor alone would inverse the tendency of above related results. Personally, I doubt this would be the case, simply because of comparison measures chosen above would not allow it.

Inadequate comparative measures. Indeed, conclusions obtained on QP, Q^* and QC in these papers – and others – are misleading. These studies consider two categories of tree-building methods, independent of their reconstruction paradigm (quartets, distances or characters): "*optimization*" methods that infer fully resolved trees, i.e. trees with a structure similar to that of the estimated tree, as opposed to "*conservative*" methods, that usually propose partially resolved trees. It is important to make the distinction between these two kinds of methods to realize that some comparison measures are inadequate here. For instance, [SvH96, RG01] measure the number of model trees correctly recovered by the methods under study. But inherently, conservative methods will achieve lower success rates, with these rates being bounded above by the percent of times they propose a fully-resolved tree. Even randomly completing the partially resolved trees proposed by QP will on average increase its results for this measure. Similarly, [SJWMV03] compare methods on the basis of the number of correct edges they are able to find. Clearly Q^*, QC and QP are penalized according to this measure. Instead of estimating the accuracy, this measure mainly estimates the average level of resolution of the trees inferred by the different methods. This is particularly clear in [SJWMV03, Figs. 6 and 7], where methods have a success rate directly dependent on their degree of conservatism. The observation of [SJWMV03] that the observed performance of Q^* drops to zero as the number of taxa increases is also simply explained: for a given edge, the method does not allow for a single error on a quartet crossing that edge. Hence, the more taxa in the estimated trees, the more quartets estimated and then the more chances that an edge has an incorrectly inferred quartet. Moreover, [SJWMV03] observed that the gap in accuracy between QC and NJ increases proportionally to the number of taxa. Here again, the number of quartets is $O(n^4)$ while the number of accepted errors across an edge for QC to recover the edge is $O(n^2)$, indicating that the probability that an edge will be recovered by QC diminishes as the number of taxa increases.

Optimization and conservative methods have also been compared in terms of the MAST distance (e.g. [SWR08]). The MAST distance between two trees is proportional to the smallest number of leaves to remove from both trees so that they will become homeomorphic. However, this distance again favors the method proposing the most resolved trees (at equal accuracy): any multifurcation greatly penalizes conservative methods by excluding full subtrees and their leaves.

The RF distance is also inadequate for comparing conservative and optimization methods. In-

deed, the former methods start from the assumption that inferring incorrect edges (type I error, also called false positive) is more penalizing than not inferring a correct edge (type II error, also called false negative), whereas the RF distance gives – somewhat arbitrarily – equal weight to both error terms. The comparison between the two kinds of methods is then, here again, flawed from the start.

The point I want to make here is that conservative quartet methods and distance-based methods have mainly been compared on the basis of error measures previously in use in the field but inadequate for this job. False negative (FN) and false positive (FP) rates should be presented separately when comparing conservative and optimization methods, as done in [SWR08]: these two error terms cannot really be combined together as done in the RF distance. Besides, the average resolution degree of the inferred trees should be given to help distinguish between the quality of the estimation proposed and the structural gap separating the inferred trees from fully resolved trees. For the same reasons, a measure based on the compatibility criterion (e.g. MCT, see Chapter 6) would be more suited than one relying on subtree isomorphism, such as MAST.

Combining conservative and optimization methods. The two kinds of methods have different goals. Conservative quartet methods only seek to provide reliable edges. Thus, they can serve as a basis for a regular tree-building method, by considering the tree they provide as a backbone tree to be completed using a usual mathematical criterion, such as NJ, MP or ML. In this way, different methods could be appropriately combined so as to get better estimates than either of the method. For instance, I showed that a simple quartet heuristics called *AddQuad* always obtains more accurate results when starting from the Q^* tree than from no tree at all [Ber97]. T. Warnow and her team particularly followed this track proposing a series of hybrid methods. In accordance with the observations of [BG97], they observed that the Q^* method and the Buneman tree (its distance counterpart) provided trees that were contractions of the correct tree in 99% of the time. They first proposed a very simple method combining NJ and the Buneman tree: edges of the NJ tree incompatible with those proposed by the Buneman tree where simply collapsed. The resulting tree was already showing better accuracy than the native NJ method [RSWY97]. Then, they designed much more involved methods that are additionally fast converging [HNR$^+$98, HNW99, NRSJ$^+$01, RMWW04].

Quartet inference errors are the crux of the matter. Turning now to the comparison of quartet-based optimization methods with optimization methods of other paradigms, the results obtained for WO in [RG01] where not encouraging. The reason for these results can stem from the fact that WO tries to output the tree that satisfies a maximum number of quartets. However, this behaviour (also advocated in a number of other papers [SvH96, BdCG$^+$98, GN03]) is misleading simply because a non-negligible part of the input quartets are incorrect. An experimental demonstration of this point lies at the end of the paper of V. Ranwez and O. Gascuel who report that in their simulations the correct tree most often satisfied less input quartets than other trees found by heuristic methods [RG01].

The point is thus clearly to decipher between correct and incorrect quartets. This is were

progress needs to be done. This can be achieved at least in three directions: *(i)* providing better quartet inference rules, e.g. following efforts of [Wil99b], *(ii)* detecting incorrect quartets, e.g. by extending cleaning algorithms, or *(iii)* avoiding “dangerous" quartets, e.g. by focusing on those of small diameter as done by SQM, hence partly escaping LBA problems. Concerning *(i)*, a more involved quartet inference method has been recently proposed on the basis of phylogenetic invariants [CFS07]. Though the authors emphasize that the computation of the invariants of a given evolutionary model just needs to be done once, they do not provide running time of their method on a single quartet. Further investigation is then required to know whether practical times can be achieved given the large number of quartets to consider. Concerning *(ii)* and *(iii)*, more involved quartet optimization methods have been designed: the team of P. Kearney proposed the HyperCleaning* method, an optimization method inspired from conservative cleaning techniques, which showed better results than NJ on 35- and 40-taxa trees [Hu02]; then the team of T. Warnow proposed the Short Quartet Puzzling method, an optimization variant of the conservative SQM method, which consistently beat NJ on 30- to 80-taxa trees [SWR08].

Thus it seems that the history of quartet methods in phylogenetics has not yet ended. However, it must be noted that these methods are not universal, due to their inherent combinatorics, i.e. dealing with $O(n^4)$ sets of elements in a study considering n taxa. Thus no study on more than several hundreds taxa is likely to be amenable to reconstruction through quartet methods even when resorting to massively parallel computing resources. Though this still covers a majority of practical phylogenetic datasets, this limits the use of quartet methods in the first assembling steps of large supertrees.

Part III

Agreement of subtrees and trees

Chapter 6

Mast and Mct

6.1 The maximum agreement subtree problem

Definition. Labeled trees can be compared using various methods, but a well-known technique is the **Maximum Agreement SubTree** (MAST) method [Gor79]. Informally, given a set of trees whose nodes are labeled, it seeks a tree with the largest number of labels that is homeomorphically included in all input trees, i.e. that has exactly the same topology as any input tree restricted to its set of labels. In that respect, MAST is sometimes referred to as a pattern matching problem on trees. See Figure 6.1 for an example. MAST has been applied in various areas of computer science such as image processing, databases, data mining, knowledge representation, but also in other sciences such as linguistics, mathematical psychology, classification, chemical sciences and biology.

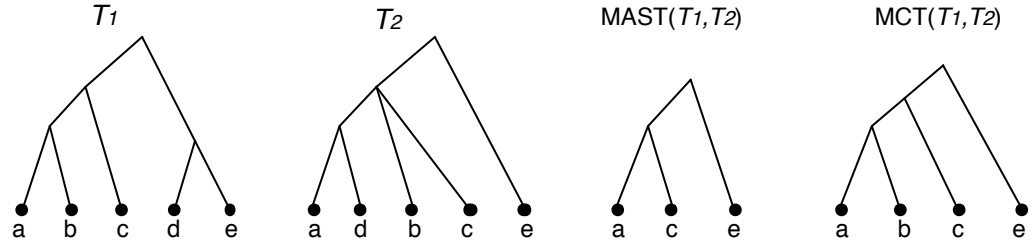

Figure 6.1: A collection $\mathcal{T} = \{T_1, T_2\}$ of two incompatible trees, a maximum agreement subtree for $\mathcal{T}$ and a maximum compatible tree for $\mathcal{T}$. Thanks to a less stringent definition, the MCT tree contains here more labels than the MAST tree.

Comparing trees through the MAST method can have several purposes:

- to measure the similarity of trees in a collection, e.g. trees inferred by different tree-building methods from the same dataset. In this case, the information that matters is the maximum number of leaves whose positions all input trees agree upon.

- to obtain a consensus of trees with different characteristics. In this case, the user is more interested in the structure of a largest subtree that is common to all input trees.

Previous results on MAST. The MAST method was introduced in 1980 by A. Gordon [Gor80] in a paper aimed at the classification field. It was used for the first time on biological data, though with no particular focus on phylogenetics, by [FG85]. The first published algorithms applied only to the case of *two* input trees. For this particular case, Finden and Gordon gave a heuristic algorithm, meaning that the output of their algorithm is not guaranteed to be optimal. Then, an exact algorithm of complexity $O(n^{1/2+\varepsilon \log n})$ was presented by Kubicka *et al.* [KKM92], where n denotes the number of leaves in an input tree. This result was improved by M. Steel and T. Warnow to finding MAST of two trees in time $O(n^2)$ for bounded degree trees and $O(n^{4.5}\log n)$ for unbounded degree trees [SW93]. A similar result was independently found by Goddard *et al.* who provided an $O(n^2)$ algorithm for the case of two binary trees [GKKM93]. A. Amir and D. Keselman were the first to provide an algorithm handling the case of more than two trees, achieving a complexity of $O(kn^{d+1} + n^{2d})$ for k trees including one whose degree is bounded by d [AK94]. Additionally, they proved that the MAST problem is NP-Hard for three trees of unbounded degree and gave an $O(kn^5)$ 3-approximation algorithm for the problem. This means that their heuristic algorithm always outputs a tree whose size is at most three times less than that of a maximum agreement subtree.

The following years, many papers improved the above listed complexities and efficient algorithms were obtained for particular cases of the general problem. For the case of two rooted trees, the current best results are an $O(n\log n)$ algorithm for binary trees [CFCH$^+$01] and an $O(\sqrt{D}n\log\frac{2n}{D})$ algorithm for trees of degree bounded by D [KLST01]. When the two input trees are unrooted and of unbounded degree, the $O(n^{1.5})$ algorithm of [KLST99] can be used. Suppose k rooted trees are given as input, if at least one input tree has maximum degree d, then MAST can be solved in $O(n^d + kn^3)$ time [FPT95, Bry97] or slightly faster when resorting to quite involved data structures [LHC$^+$05].

The MAST problem is also known to be fpt[1] in p, the smallest number of labels to remove from the input set of labels such that the input trees agree: [DFS99] describe an $O(3^p kn\log n)$ time algorithm and [AGN01] give an $O(2.27^p + kn^3)$ time algorithm. The fact that MAST is fpt means that some instances of the problem can be solved efficiently, i.e. here those where the input trees disagree on the position of few leaves. This parameterized version of the problem is of particular interest in phylogenetics where many instances of MAST consist of phylogenies inferred by different tree-building methods on the basis of molecular sequences of reasonable length. Hence, the trees given as input to MAST usually differ with respect to the location of a small number p of species. This means that the above algorithms are usually likely to achieve fast running times.

[1] fixed-parameter tractable, see Chapter 2.

6.2 The Maximum Compatible Tree problem

Definition. The Maximum Compatible Tree (MCT) problem is a variant of MAST that is particularly relevant in phylogenetics. It has been introduced independently in [HS96] and [HJWZ96, under the MRST acronym]. The MCT problem is solved by finding a largest subset of leaves on which the input trees are *compatible*. When the purpose is to obtain a consensus tree of the input trees, the problem is formulated as finding a tree on such a set of leaves. See Figure 6.2 for an example.

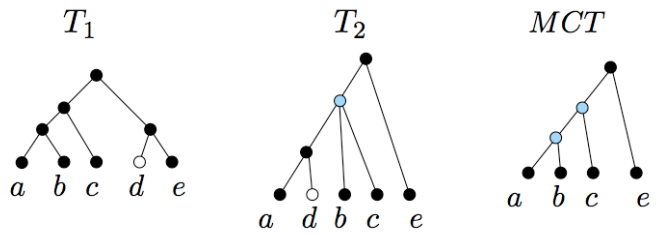

Figure 6.2: An incompatible collection of two input trees $\{T_1, T_2\}$ and their maximum compatible tree T. Here T_1 and T_2 only disagree on the position of leaf d. In this example, T strictly refines T_2 restricted to $L(T)$, which is expressed by the fact that the blue node in T_2 has its child subtrees distributed between several connected nodes of T (blue nodes).

The difference between MAST and MCT is that the former asks for isomorphism while the latter asks for compatibility. In other words, MAST requires a consensus tree whose branching information (i.e. groups of leaves or *clades*) are contained in *each* input tree, while MCT asks for a tree whose branching information is contained in *some* input tree(s), while being compatible with the topological information of the other trees. This difference allows the MCT tree to be more informative than the MAST tree, i.e. to generally contain more clades and leaves. For instance, on the collection $\{T_1, T_2\}$ of Figure 6.2, a MAST tree contains only three leaves and one non-trivial clade, while the MCT tree contains four leaves and two non-trivial clades. Note that there is no difference between MAST and MCT in the case where the input trees are binary.

In phylogenetics, quite often some edges of trees inferred from primary data can be collapsed due to insufficient statistical support. This results in some higher-degree nodes in the trees to be compared. Each such node does not indicate a multi-speciation event but rather the uncertainty with respect to the branching pattern to be chosen for its child subtrees. In such situations, the MCT problem is to be preferred to MAST, as it correctly handles high degree nodes, enabling them to be resolved according to branching information present in other input trees. In other words, MAST considers multifurcating nodes as hard polytomies, while MCT considers them as soft polytomies. MCT thus conserves more input leaves in the output tree, hence a larger degree of similarity is detected between the input trees.

Previous results on MCT. In 1996, the MCT problem was shown to be NP-hard on six trees [HS96] then on two trees as soon as one of them is of unbounded degree [HJWZ96]. A first algorithm for the problem was proposed by G. Ganapathy and T. Warnow that solves the problem for k rooted trees of n leaves and maximum degree D in $O(2^{2kD}n^k)$ time [GW01]. The same authors also provided a 3-approximation algorithm for CMCT running in $O(k^2n^2)$, where CMCT is the complement version of MCT aiming at selecting the smallest number of leaves to be removed from the input trees such that they become compatible [GW02]. More recently [GN05], S. Guillemot and F. Nicolas proposed an $O(n^{2^{D+2}} + kn^3)$ exact algorithm for MCT inspired by the $O(n^D + kn^3)$ algorithm of D. Bryant for MAST. This proves that MCT can be solved in polynomial time on trees of bounded degree. The complexity bound of this algorithm is tight as MCT cannot be solved in time $O(n^{o(2^{D/2})} + poly(k,n))$ unless $SNP \subseteq SE$ [2] [GN05]. Guillemot and Nicolas also showed that MCT is W[1]-hard for parameter D, i.e. that it is likely that no fpt algorithm on this parameter will be obtained [GN06].

6.3 Results obtained on MAST and MCT

I first worked on the MAST and MCT problem with F. Nicolas when he was in his last PhD year at LIRMM, then to a further extent with C. Paul and S. Guillemot when supervising the PhD of the latter. The results obtained during these collaborations are given below.

With F. Nicolas, we obtained two linear time *certifying* algorithms for deciding the isomorphism, resp. compatibility, of k rooted trees [BN06]. This means that the algorithms are able to produce a positive or negative certificate for the considered problem. For instance, for compatibility, we provided an algorithm, called FIND-REFINEMENT-OR-CONFLICT, that either outputs a tree refining all input trees, hence proving their compatibility, or alternatively a set of three leaves x, y, z whose resolution the input trees disagree on, i.e. such that the triple $xy|z$ is observed in one tree and $xz|y$ is observed in another input tree. As no single tree can include both triples at the same time, this shows the incompatibility of the input trees. To handle unrooted input trees, all of them can simply be rooted at the same arbitrary leaf and then the above-mentioned algorithms can be applied.

We then used these algorithms as subroutines in exact algorithms to solve the following parameterized versions of MAST and MCT:

Name: PARAMETERIZED ROOTED MAXIMUM AGREEMENT SUBTREE (p-rMAST)
Input: A collection $\mathcal{T}$ of k rooted trees with identical leaf set L of cardinality n.
Parameter: an integer $p \geq 0$.
Task: Find an agreement subtree T of $\mathcal{T}$ s.t. $|T| \geq n - p$, if such a tree exists.

Name: PARAMETERIZED ROOTED MAXIMUM COMPATIBLE TREE problem (p-rMCT)
Input: A collection $\mathcal{T}$ of k rooted trees with identical leaf set L of cardinality n.

[2] The inclusion $SNP \subseteq SE$ is widely believed to be unlikely in complexity theory [GN05].

Parameter: an integer $p \geq 0$.
Task: Find a tree T compatible with $\mathcal{T}|L(T)$ s.t. $|T| \geq n - p$, if such a tree exists.

As discussed in Section. 6.1, parameter p is particularly relevant to phylogenetic applications. MAST and MCT problems are linked with the well-known 3-HITTING-SET problem in that when the input trees are not isomorphic, resp. compatible, then there are topological conflicts on sets of three leaves.

Definition 6.1 *Given a collection $\mathcal{T}$ of rooted trees, a set of three leaves a,b,c is*

- *a **hard conflict** between trees of $\mathcal{T}$ if both $ab|c$ and $ac|b$ belong to $\mathsf{rt}(\mathcal{T})$.*
- *a **soft conflict** between trees of $\mathcal{T}$ if both $ab|c \in \mathsf{rt}(\mathcal{T})$ and $(a,b,c) \in \mathsf{f}(\mathcal{T})$.*

To obtain the compatibility, resp. isomorphism, of the input trees, the hard conflicts, resp. hard and soft conflicts between these trees must all be removed. This is done by excluding some leaves belonging to these conflicts. These 3-leaf obstructions prompted R. Downey and M. Fellows [DFS99] to formulate an fpt algorithm for solving p-rMAST by a bounded-search tree technique similar to that used for 3-HITTING-SET (see [AGN01] for another fpt algorithm based on the link between the two problems). In [BN06], we slightly improved the algorithm of [DFS99], obtaining a very simple algorithm to solve p-rMAST and p-rMCT. Below, we see that p-rMCT can be solved by using the FIND-REFINEMENT-OR-CONFLICT certifying subroutine for compatibility.

Algorithm 1: Recursive-MCT$(\mathcal{T}, p)$

Input: A collection $\mathcal{T}$ of k rooted trees with identical leaf set L and an integer $p \geq 0$.
Result: A tree T compatible with $\mathcal{T}$ s.t. $|T| \geq |L| - p$ if such a tree exists or, otherwise, the empty tree $\emptyset$.

```
res ← FIND-REFINEMENT-OR-CONFLICT(T)
if res is a tree T then return T /* this tree is compatible with T */
/* Otherwise res is a set of three leaves that is a hard conflict between trees in T */
if p > 0 then
    foreach leaf ℓ ∈ res do
        T ← Recursive-MCT(T|(L − {ℓ}), p − 1)
        if T ≠ ∅ then return T
return ∅
```

A very similar algorithm called Recursive-MAST was designed to solve p-rMAST.

Theorem 6.1 ([BN04]) *Given a collection $\mathcal{T}$ of k input trees on the same n leaves*

(i) algorithm Recursive-MAST *solves the p-rMAST problem in* $O(3^p \cdot kn)$ *time.*

(ii) algorithm Recursive-MCT *solves the p-rMCT problem in* $O(3^p \cdot kn)$ *time.*

An alternative way to solve p-rMAST and p-rMCT is to reduce them to the 3-HITTING-SET problem, as implicitly done in [AGN01, GW02]. This is achieved quite simply on the basis of the link existing between the problems. Thanks to the reduction and the algorithm of [NR03] for solving 3-HITTING-SET, we can obtain:

Theorem 6.2 ([BN04]) *The p-rMAST and p-rMCT problems can be solved in* $O(2.27^p + kn^3)$ *time.*

When unrooted trees are available, problems p-uMAST and p-uMCT can be defined similarly to p-rMAST and p-rMCT above. p-uMAST and p-uMCT can be solved by resorting to the above-described algorithms a given number of times, while each time rooting the trees at a different leaf. We managed to bound this number of times as a function of parameter p rather than n, with the interest being that p is smaller than n and usually $p << n$:

Theorem 6.3 ([BN07]) *Given a collection* $\mathcal{T}$ *of k unrooted trees on an identical set of n leaves, p-uMAST and p-uMCT can be solved in time* $O\big((p+1) \cdot \min\{3^p kn, 2.27^p + kn^3\}\big)$.

The obtained running times are exponential in p, while measuring the level of disagreement of the input trees. Thus, these algorithms are interesting alternatives to those of [GW01, GN06] whose complexities are exponential in the number of trees and/or maximum degree of the trees. I did a rough implementation of some of these algorithms that is available on my webpage. The practicality of these algorithms has been demonstrated since they have been used for a while in a statistical procedure to estimate the congruence level of a collection of trees [dVGM07] and showed reasonable running times for almost all tested collections (deVienne, personal communication).

The link between MAST, MCT and 3-HITTING SET has also been used to obtain 3-approximation algorithms for the complement versions of the two former problems [AK97, GW02]. The complement version of MAST and MCT, denoted CMAST and CMCT, aim at selecting the smallest number of leaves to be removed from the input trees in order to obtain their agreement (i.e. isomorphism, resp. compatibility). In practice, input trees usually agree on the position of most leaves, thus approximating CMAST and CMCT is more relevant than approximating MAST and MCT.

Fast approximation algorithms are the alternatives to fpt algorithms when faced with NP-hard problems such as MAST and MCT. With S. Guillemot, F. Nicolas and C. Paul, we improved the previously published approximation algorithms for the problems. Using dedicated but simple data structures, we obtained a linear-time, i.e. $O(kn)$, algorithm for CMAST and an $O(n^2 + kn)$ algorithm for CMCT. This represents a substantial improvement of the $O(kn^5)$ algorithm of [AK97] and $O(k^2n^2)$ algorithm of [GW02].

We also obtained several theoretical results on the approximability of the problems. We showed that:

- for all $\varepsilon > 0$, the general MCT problem is not approximable within $n^{1-\varepsilon}$ unless NP=ZPP as was already known for the MAST problem [BVM00].
- CMAST on 3 trees and CMCT on 2 trees are APX-hard, ruling out the possibility of a PTAS for these particular problems unless P=NP. A *PTAS* is a $(1+\varepsilon)$-approximation scheme that runs in polynomial time for fixed ε. This means that a tradeoff can be established between the precision and running time of the heuristics.
- MAST on 3 trees and MCT on 2 trees cannot be approximated within $2^{\log^{\delta}(n)}$, for all $\delta < 1$, unless NP $\subseteq DTIME(2^{\text{polylog}(n)})$.

Publication(s) whose material is described in this chapter:

- Improved parameterized complexity of Maximum Agreement Subtree and Maximum Compatible Tree problems, **V. Berry** and F. Nicolas, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3), 289–302, 2006. Part of this paper appears in the proceedings of the 15th *Ann. Combinatorial Pattern Matching Symposium* (CPM'04).
- On the Approximation of Computing Evolutionary Trees, **V. Berry**, F. Nicolas, S. Guillemot and C. Paul, Proc. of the *11th Annual International Conference on Computing and Combinatorics* (COCOON'05), *LNCS* 3595, pp. 115-125, Springer, 2005. The first results presented in this conference paper gave rise to two journal papers containing more complete results. One is to appear in *Discrete Applied Mathematics* and one in the *Transactions on Algorithms* journal.
- Maximum Compatible Tree, **V. Berry** in *Encyclopedia of Algorithms*, M.-Y. Kao Ed., Springer, 2008.

Other publication(s) relevant to the topic but not described in this chapter:

- From constrained to Unconstrained Maximum Agreement Subtree, **V. Berry**, D. Peng and H.-F. Ting, *Algorithmica*, 50(3), 369–385, 2008.
- One fig to bind them all: host conservatism in a fig wasp community unraveled by cospeciation analyses among pollinating and nonpollinating fig wasps, E. Jousselin, S. van Noort, **V. Berry**, J.-Y. Rasplus, N. Rønsted, J.C. Erasmus and J.M. Greef, *Evolution*, 62-7: 1777–1797, 2008.

Chapter 7

SMAST and SMCT

7.1 Extending MAST and MCT to the supertree context

An input for MAST and MCT is a set of trees having all the same set of taxa labeling their leaves. The input trees only differ with respect to the branching pattern or groups they display for the labels. From 2003 on until recently, I studied the extension of MAST and MCT to the supertree context, where input trees have different though intersecting labels sets. The obtained methods/computational problems where denoted SMAST and SMCT.

There are several interests in generalizing the MAST and MCT problems to the supertree context. First, they can be used to measure the congruence of a collection of source trees to be combined into a supertree. The relative congruence of several collections can hence be compared and the most congruent one can then be chosen to assemble a consensual supertree. The added value of any source tree can also be measured as the net effect it has on the congruence of the collection as measured by SMAST and SMCT.

Another interest of SMAST and SMCT is that they propose alternatives to usual supertree methods that incorporate all labels of the source trees in the supertree. This is a problem whenever the input trees contain some "rogue" taxa, i.e. taxa whose position greatly differs from one input tree to the other. Unfortunately, this phenomenon happens quite often in real instances. In such cases, veto supertree methods [Gor86, GP02] propose unresolved nodes (multifurcations) in the supertree. If each position is supported by the same number of source trees, even voting methods [Bau92, Rag92, SS00, CEFBS02] will not be able to choose among the different positions for the leaves and produce partially resolved supertrees or, worse, will choose arbitrarily between one of them depending on the number of neighboring leaves [BEB98, Gol05]. Thus, when the supertree inferred by a classical supertree method contains multifurcations, it is worth checking whether excluding a few leaves would result in a much more informative picture. Pushing it further, several authors [Gor86, WTLB00, BEGS02] have suggested that alternative supertree methods could be obtained by focusing on leaves individually – and removing uncertain ones – rather than on subsets of leaves (triples, quartets or clusters), as usually done. SMAST and SMCT are then well suited to

suggest a set of leaves upon whose positions the input trees disagree.

Replacing MAST by SMAST also provides some advantage in the detection of horizontal gene transfers (HGT). MAST has indeed been used in this context on the rationale that leaves whose position gene trees highly disagree upon are susceptible to being subject to HGT events [GWK05, NRW05]. The broad-scale study of [GWK05] shows that this technique can indeed be quite successful. In this study, for each pair of trees, the size of a MAST tree is computed for the two trees restricted to their common labels. A gene tree is detected to be affected by HGT events depending on the distribution of MAST scores obtained for the pairs to which it belongs. However, here a MAST tree cannot be computed for a vast majority of pairs simply because the considered genes do not have enough labels in common, which limits the confidence in the final conclusions [GWK05]. Replacing MAST computations on pairs of gene trees by SMAST computations on more than two trees would undoubtedly increase the proportion of cases where there is enough overlap to conduct the analysis.

Still another use of SMAST and SMCT is in improving the accuracy of traditional methods. For instance, the popular MRP method has relatively low accuracy when the input trees overlap moderately and [BES01] recommend adding to the set of input trees a tree with leaves spanning most input trees, that they call a *seed* tree. Any supertree provided by SMAST and SMCT most likely contains leaves from most, if not all, input trees (see next section) and, moreover, fully agrees with all of these trees by definition. It is thus a potential candidate for being a seed tree.

7.2 Results obtained on SMAST and SMCT

The results presented below where obtained during the PhD period of F. Nicolas and S. Guilemot. With F. Nicolas, we proposed the following definition to extend MAST and MCT to the supertree context [BN04]:

Definition 7.1 *Given a collection $\mathcal{T}$ of leaf-labeled trees, an **agreement supertree** of $\mathcal{T}$ is a tree T with $L(T) \subseteq L(\mathcal{T})$ such that $\forall T_i \in \mathcal{T}$, $T|L(T_i) = T_i|L(T)$. An agreement supertree of $\mathcal{T}$ that is of maximum size is called a **maximum agreement supertree** of $\mathcal{T}$ and is denoted $SMAST(\mathcal{T})$. A **supertree compatible with** $\mathcal{T}$ is a tree T with $L(T) \subseteq L(\mathcal{T})$ such that $\forall T_i \in \mathcal{T}$, $T|L(T_i) \trianglerighteq T_i|L(T)$. A supertree compatible with $\mathcal{T}$ that is of maximum size is called a **maximum compatible supertree** of $\mathcal{T}$ and is denoted $SMCT(\mathcal{T})$.*

The optimization problem corresponding to SMAST is stated as follows:

Name: MAXIMUM AGREEMENT SUPERTREE (SMAST)
Input: A finite collection $\mathcal{T}$ of trees (all rooted or all unrooted).
Task: Find a maximum agreement supertree of $\mathcal{T}$.

The optimization problem for SMCT is defined in the same way. Figure 7.1 shows examples of SMAST and SMCT trees for a collection of two trees.

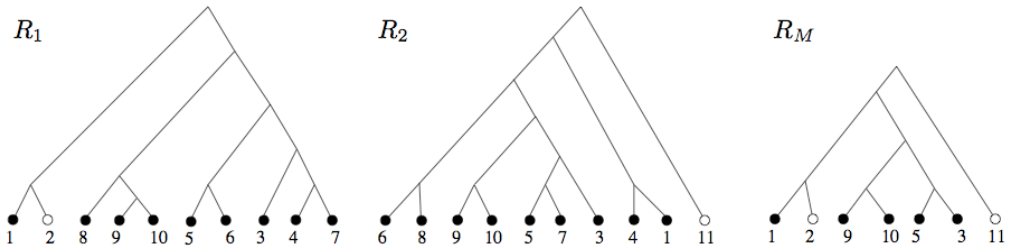

Figure 7.1: A collection $\mathcal{R} = \{R_1, R_2\}$ of two source trees on tomatoes taken from [BR04] and a supertree R_M. In this example, the supertree represents both a SMAST tree and an SMCT tree for $\mathcal{R}$. Leaves appearing in only one source tree are displayed in white. Correspondance between numbers and species: 1 – *L. lycopersicoides*, 2 – *L. juglandifolium*, 3 – *L. peruvianum*, 4 – *L. chilense*, 5 – *L. pennellii*, 6 – *L. hirsutum*, 7 – *L. chmielewskii*, 8 – *L. esculentum*, 9 – *L. pimpinellifolium*, 10 – *L. cheesmanii*, 11 – *L. rickii*.

The basic case of two compatible rooted trees. We first concentrated on the computation of SMAST and SMCT for a collection of two rooted trees. To achieve this goal, we first designed a fast algorithm for computing a supertree in the case of two compatible trees such that one refines the other when restricted to their common leaves. When faced with such input, a supertree on all leaves can be obtained by grafting in the refining tree the subtrees of the other tree that contain leaves specific to the latter. By defining links between nodes of the two trees on the basis of least common ancestor relationships and identifying the specific subtrees of the input trees, we achieved linear running time for this algorithm, that we called MERGETREES.

Theorem 7.1 *Given a collection $\mathcal{R} = \{R_I, R_A\}$ of two rooted trees such that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$, the algorithm* MERGETREES (R_I, R_A) *returns a tree R such that $L(R) = L(\mathcal{R})$ and such that R is a SMCT tree for $\mathcal{R}$. In the particular case where $R_A|L(R_I) = R_I|L(R_A)$, then R is a SMAST tree for $\mathcal{R}$. Moreover, a call to algorithm* MERGETREES (R_I, R_A) *costs $O(n)$ time where $n = |L(R_I) \cup L(R_A)|$.*

A corollary is that this grafting procedure can be used to compute in linear time the Strict Consensus Supertree defined by A. Gordon and for which he proposed an $O(n^3)$ algorithm [Gor86].

Cases in which the general problems are sovable in polynomial time. Considering a general collection $\mathcal{T}$ of input trees, we can split the leaves $L(\mathcal{T})$ into three caterogies: those belonging to just one tree, called **specific** leaves and denoted $L_S(\mathcal{T})$, those belonging to all trees and denoted $L_\cap(\mathcal{T})$, and those belonging to several but not all input trees, denoted $L_\delta(\mathcal{T})$. We showed that $L_S(\mathcal{T})$ is fully included in any SMAST and SMCT tree for $\mathcal{T}$. The intuitive reason for this is that the relative position of the specific leaves is indicated by just one source tree, hence no conflict can arise when combining all source trees[1]. As a direct corollary, $L_\delta(\mathcal{T}) = \emptyset$ is a sufficient condition

[1] note, though, that the formal proof that everything works fine requires half a page [BN07].

to solve SMAST and SMCT problems by resorting to MAST and MCT as subproblems. Note, however, that things can go wrong when $L_\delta(\mathcal{T}) \neq \emptyset$, as shown by the example of Figure 7.2. As $L_\delta(\mathcal{T}) = \emptyset$ is always met on two trees.

This shows a special case where the SMAST problem can be solved in linear time. Additionally SMAST, can be solved in polynomial time for collections of more than two trees when $L_\delta(\mathcal{T}) = \emptyset$ and the maximum degree of at least *one* input trees is bounded. In comparison, using the latest results on MCT [GN06], SMCT can be solved in polynomial time when $L_\delta(\mathcal{T}) = \emptyset$ and the degree of *all* input trees is bounded.

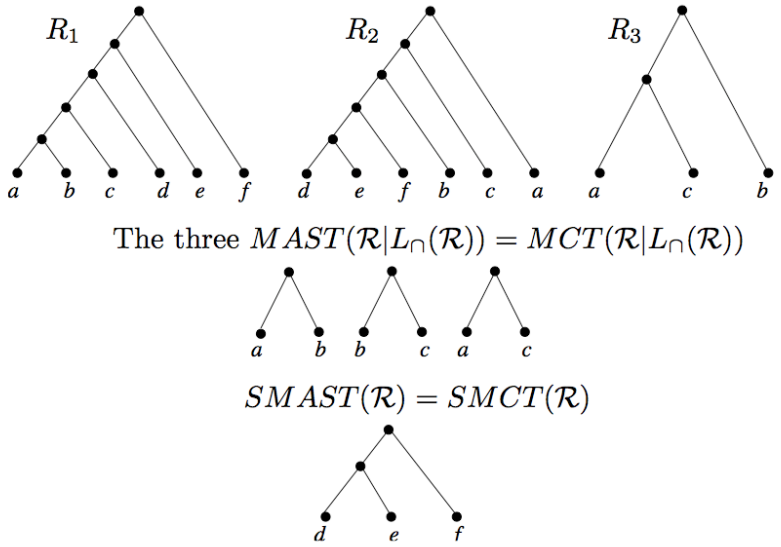

Figure 7.2: A collection $\mathcal{R} = \{R_1, R_2, R_3\}$ of rooted input trees for which the MAST and MCT trees cannot be used as backbones of SMAST and SMCT trees.

The above results also apply to the case of unrooted input trees. For instance, when faced with two unrooted trees, rooting them on the external edge leading to a common leaf allows us to solve SMAST and SMCT problems by reducing them to the rooted case.

Reducing to instances of bounded size. On the basis of several results known in the field for characterizing trees by triples, fan, quartets and stars [BS95, AK97, Bry97, BGNP05, BN06] we then showed that SMAST and SMCT for trees of arbitrary size can be reduced to computing the same problems on trees of bounded size:

Lemma 7.1 ([BN04]) *Let R, R' be two rooted trees and U, U' be two unrooted trees.*

(i) *R is isomorphic to R' if and only if $\mathsf{rt}(R) = \mathsf{rt}(R')$ and $\mathsf{f}(R) = \mathsf{f}(R')$.*

(ii) *R refines R' if and only if $\mathsf{rt}(R') \subseteq \mathsf{rt}(R)$ and $L(R) = L(R')$.*

(iii) *U is isomorphic to U' if and only if* $\mathsf{q}(U) = \mathsf{q}(U')$ *and* $\mathsf{s}(U) = \mathsf{s}(U')$.

(iv) *U refines U' if and only if* $\mathsf{q}(U') \subseteq \mathsf{q}(U)$ *and* $L(U) = L(U')$.

Then accumulating small trees for collections of trees, we deduce:

Corollary 7.1 ([BN04]) *Let $\mathcal{R}$ be a collection of rooted trees and let R be a rooted tree with $L(R) \subseteq L(\mathcal{R})$.*

(i) *R is an agreement supertree of $\mathcal{R}$ if and only if R is an agreement supertree of* $\mathsf{rt}(\mathcal{R}) \cup \mathsf{f}(\mathcal{R})$,

(ii) *R is a supertree compatible with $\mathcal{R}$ if and only if R is a supertree compatible with* $\mathsf{rt}(\mathcal{R})$.

Let $\mathcal{U}$ be a collection of unrooted trees and let U be an unrooted tree with $L(U) \subseteq L(\mathcal{U})$.

(iii) *U is an agreement supertree of $\mathcal{U}$ if and only if U is an agreement supertree of* $\mathsf{q}(\mathcal{U}) \cup \mathsf{s}(\mathcal{U})$,

(iv) *U is a supertree compatible with $\mathcal{U}$ if and only if U is a supertree compatible with* $\mathsf{q}(\mathcal{U})$.

A corollary of the possible reduction of SMAST and SMCT to collections of bounded size in an exact algorithm solving the problems in $O(n^{\frac{p(p+1)}{2}} \cdot poly(n,k) + kn)$, where p is the number of leaves to remove to obtain the agreement of the input trees [BN03]. This algorithm is obtained by exhaustive examination of all subsets of size less than p, each time resorting to the ONETREE algorithm of [NW96]. The time complexity of this algorithm indicates that SMAST and SMCT are solvable in polynomial time when the input trees disagree on a constant number of leaves. However, this algorithm is mainly a theoretical freak and can only be used for practical collections of trees disagreeing on the position of a very small number of leaves.

The case of binary input trees. The three algorithms reported below consider the particular case where the input trees are binary. For this particular case of the SMAST problem, more efficient algorithms can be obtained, as shown by the PhD work of S. Guillemot. Note also that since the input trees are binary, SMAST and SMCT are equivalent here.

Given a collection $\mathcal{T}$ of k input binary trees, the new algorithms rely on the consideration of **positions** in $\mathcal{T}$, where a position is a tuple of nodes, with one in each tree of $\mathcal{T}$. This combinatorial object is similar to *lca-tuples* considered in early works on the MAST problem [AK94, FPT95]. A position can be seen to define a restriction of the input collection $\mathcal{T}$ by considering subtrees of the input trees rooted at nodes composing the position. By building an intersection graph of the largest clades of these subtrees it is possible to determine whether this restriction of $\mathcal{T}$ is compatible. When this graph is connected, the restricted collection is incompatible, similar to what happens with the well-known Aho graph [ASSU81, SS03]. However, the interest in the intersection graph is that, when incompatibility arises, a set of at most k labels can be easily found that is responsible for the incompatibility. We do not know of such a result for the Aho graph. When the intersection

graph is disconnected, then it is recursively decomposed until incompatibility is detected or graphs of trivial sizes are obtained. The intersection graph is the basis of a certifying algorithm to decide the compatibility of a collection of rooted trees:

Theorem 7.2 ([GB07]) *Given a collection $\mathcal{T}$ of k rooted trees on n leaves, there is an algorithm which, in $O(kn^2)$ time, decides if $\mathcal{T}$ is compatible, and returns a conflict of size $\leq 2k$ in case of incompatibility.*

This algorihtm is in turn the basis of a simple algorithm based on the bounded-search tree technique:

Theorem 7.3 ([GB07]) *The SMAST problem on a collection of k rooted binary trees can be solved in $O((2k)^p \times kn^2)$ time.*

This algorithm is really simple to implement and should be efficient on collections of gene trees conflicting on the position of few leaves. Its time complexity shows that the SMAST problem *on binary trees* is fpt for the combination of parameters (k, p). By considering an even more particular case of SMAST, an fpt algorithm on parameter p alone can be obtained. Consider the case of a collection $\mathcal{T}$ of binary rooted trees such that each 3-label subset of $L(\mathcal{T})$ appears in at least one input tree. Recall that SMAST on trees of arbitrary size can be reduced to SMAST on rooted triples and fans. The particular case considered here then corresponds to solving the SMAST problem on a collection of triples that is at least complete (see definition in chapter 2), but potentially contains several different resolutions for the same set of three labels. What makes things easier in the case of a complete collection is that compatibility amounts to tree-likeness. Consequently, we can rely on the fact that conflicts among input triples can be circumvented to a small number of leaves. Define here a **conflict** to be a small subset $C \subseteq L(\mathcal{T})$ such that $\mathcal{T}|C$ is incompatible. The following result on rooted triples is an analog of a known result on quartets [BD86]:

Theorem 7.4 ([GB07]) *Given a complete collection of triples $\mathcal{T}$, it is possible to decide in $O(n^3)$ time if $\mathcal{T}$ is tree-like, and in this case to return the tree displayed by $\mathcal{T}$, or in the case where $\mathcal{T}$ is incompatible to return a conflict C with $|C| \leq 4$.*

This certifying algorithm can then be used in an fpt algorithm for solving the SMAST problem by detecting a conflict, removing alternatively each leaf in the conflict (and the corresponding triples from $\mathcal{T}$) and then recursing until no conflict remains or more than p leaves have been removed:

Theorem 7.5 *The SMAST problem parameterized in p can be solved in $O(4^p n^3)$ time.*

Lastly, S. Guillemot also proposed an $O((8n)^k)$ time algorithm resorting to a dynamic programming approach based on positions. This significantly improves on the $O(k(2n)^{3k^2})$ algorithm proposed in [JNSS04].

Intractability results. In [BN03], we have shown that the SMAST problem is NP-hard by reducing from the 3-HITTING-SET problem[2]. The latter is fixed-parameter tractable (fpt) for parameter p. This first gave us some hope on the tractability of SMAST, but later on we found a reduction from the general HITTING-SET problem, showing that SMAST is in fact W[2]-hard for p [BN04, BN07]. This means that it is very unlikely that an fpt algorithm exists when considering this parameter. This shows that there is a significant gap between the difficulties of SMAST and MAST, as the latter is fpt in p [DFS99, BN06].

Note that the above-mentioned reduction from HITTING-SET is approximation preserving. This rules out the possibility of any polynomial time algorithm approximating, within a constant factor, CSMAST and CSMCT, i.e. the complement optimization problems of SMAST and SMCT. This contrasts with CMAST and CMCT that can be 3-approximated by efficient algorithms as detailed in the previous chapter.

Restricting SMAST to the case of binary input trees is not sufficient to make it a lot more tractable in terms of parameterized complexity. Let q be the complement parameter of p, i.e. let q be a lower bound on the number of labels that are missing in a SMAST solution. SMAST is W[1]-complete for parameter q [JNSS05], but also for parameter k and for the parameter pair (q,k) [GB07].

7.3 Are SMAST and SMCT hot topics?

As a final remark, I have noted that the extension of MAST and MCT to the supertree context seems to be a hot topic as on two occasions our results were published at the same time as similar – and complementary – results independently proposed by other teams!

After having published our first results on SMAST and SMCT in 2004, we learnt that another team had independently studied the SMAST problem (that they called MASP). Their paper – appearing 3 months before ours – also proposed a polynomial time algorithm for the case of two rooted trees [JNSS04, JNSS05]. For the particular case of binary trees, they also proposed an exponential algorithm running in $O(k(2n)^{3k^2})$, showing that SMAST can be solved in polynomial time when the input collection contains a constant number of binary trees. We later subsumed this result by the $O((8n)^k)$ algorithm mentioned in the previous section.

By reduction from INDEPENDENT SET and VERTEX COVER, Jansson *et al.* also showed that SMAST is NP-hard for any fixed $k \geq 3$ when the trees are of unbounded degree, and also when the input trees are of fixed maximum degree $D \geq 2$ when k is unrestricted. Additionally, they proposed an $(n/\log n)$-approximation algorithm running in polynomial time.

More recently, when we proposed the results for SMAST (and thus SMCT) on binary trees, we were in competition with V. Hoang and W. Sung. Several months after us, they independently proposed the same algorithm as that mentioned above for binary rooted trees running in

[2]In comparison, the MAST problem was shown to be NP-hard by reduction from 3-DIMENTIONAL-MATCHING [AK94].

$O((8n)^k)$ time [GB07]. Their finer analysis shows that the complexity bound is in fact $O((6n)^k)$ [HS08]. Additionally, they generalized this algorithm to input trees of degree larger then 2: given k rooted trees of maximum degree D over n taxa, they showed that SMAST can be solved in $O((kD)^{kD+2}(2n)^k + (2n)^k)$ time and that *SMCT* can be solved in $O(4^{kD}n^k)$ time. The latter complexity matches that of the algorithm for MCT proposed in[GW01]. This is a good surprise, as the results reported above (at least in terms of parameterized complexity) have shown that SMCT is generally less tractable than MCT. Though it is hard to make a comparison with the $O(n^{2^{D+2}} + kn^3)$ algorithm of [GN05], the latter seems preferable in the case of a collection containing a non-trivial number of binary trees, as usually happens in practice. V. Hoang and W. Sung also considered the case of unrooted trees, showing that SMAST can be solved in $O((kD)^{kD+2}(2Dn)^k + (2n)^{2k}D^k)$ time and SMCT in $O(2^{2kD}n^k)$ time [HS08].

7.4 Future work

On the theoretical side, the algorithm of [HS08] for solving the SMCT problem with the same running time as the algorithm of [GW01] for MCT calls for more work on the latter problem to obtain a faster algorithm. A first approach is to study whether the former algorithm simplifies in the case of input trees with identical label sets.

On the practical side, the running times of the algorithms reported above are close to the limit where programs can be written and used for data of reasonable size. Hence, much experimental work remains to be done to test the efficiency of SMAST and SMCT for the various applications mentioned in the introduction of this chapter. Among others, the detection of HGT will be of prime importance for me as this comes within the scope of the recent ANR funding we obtained on bacteria, where this kind of event is common. Testing whether an SMCT tree would be a good seed tree for a collection of trees whose labels only slightly overlap is also of interest since this can be done easily by reusing the simulations we did recently (see condition $d75$ of the simulation study mentioned in part IV). The performance of the usual MRP method would be compared to that of MRP when the collection is completed with the SMCT tree. Measures such as accuracy and amount of information in the supertree would indicate whether the SMCT improves the situation.

Publication(s) whose material is described in this chapter:

- Maximum Agreement and Compatible Supertrees, V. **Berry** and F. Nicolas, *Journal of Discrete Algorithms*, 5(3), 564–591, 2007.
  An extended abstract of this work appears in the proceedings of the 15^{th} *Ann. Combinatorial Pattern Matching Symposium* (CPM'04), LNCS 3109, pp. 205-219, Springer, 2004,

- Fixed-Parameter Tractability of the Maximum Agreement Supertree Problem, S. Guillemot and V. **Berry**, *IEEE/ACM Transactions in Computational Biolology and Bioinformatics*, (in press), 2008.
  An extended abstract of this work appears in the proceedings of the 18th Ann. Combinatorial Pattern Matching Symposium (CPM'07), LNCS 4580, pp. 274-285, Springer, 2007.

Part IV

Supertree methods from new principles

In the preceding chapters, we saw several supertree methods, some affiliated with quartet methods and others to the well-known MAST method. In the following part, I relate results concerning supertree methods with an original design.

The first method tackles a problem mentioned by R. Page, namely allowing internal labels in source trees and supertrees. This is the result of a collaboration with C. Semple and O. Bininda-Emonds.

Then I describe methods relying on desirable combinatorial properties that supertree methods should verify. This work results from a collaboration with several researchers in Montpellier, with the main ones being E. Douzery, V. Ranwez, and C. Scornavacca. The latter is a PhD student I co-supervise with V. Ranwez.

Chapter 8

Ancestral compatibility of trees

8.1 motivation

A high number of phylogenetic trees published in biology journals over the last 15 years are stored in TreeBASE [SDPE94, PSD03]. As of summer 2008, this database contains 6,237 trees belonging to 3,696 different studies, covering a total of 93,013 taxa. The database also stores references to papers in which the trees appear and primary data from which they were obtained. TreeBASE can be used for many purposes such as obtaining information on the phylogeny of particular groups of interest, obtaining datasets for studies of character evolution, linking trees of particular groups into more inclusive phylogenies or discovering understudied groups. This database has been used in a number of studies addressing phylogenetic, biogeographic and coevolution questions. However, this important resource has not yet been used for building large supertrees. These are still inferred from patiently hand-collected source trees [BECJ$^+$07] or from phylogenies automatically inferred from data obtained by mining sequence databases [San08]. This is really a shame as TreeBASE contains several thousand phylogenies, each validated by experts, and hence represents a valuable resource for building both large and accurate supertrees. However, supertree people are not to blame here as the content of TreeBASE is quite confusing. It was, and still is, populated by many different researchers, having varying interests and varying views on naming conventions for taxa. As a result, a number of problems impede proper use of TreeBASE without adequate preprocess.

In 2001, the well-known biologist M. Sanderson challenged the phylogenetic community to propose the “largest" and “best" possible supertree from TreeBASE. His challenge still remains unanswered. The just as famous R. Page listed several problems to solve before such a goal could be achieved [Pag04, Pag07]. A major problem he highlighted is that trees in TreeBASE consider taxa at different levels in the Tree of Life. Since the work of C. Linnaeus (1707-1778), species are grouped according to shared characteristics; the groups form a hierarchical structure called taxonomy or biological classification. The content of this hierarchy evolves regularly but its ranks, or *taxonomic levels*, are well-established; they are e.g. domain, kingdom, order, family, genus,

species[1].

The fact that TreeBASE contains taxa at different taxonomic levels, such as “mammal" and “elepehant" is a problem for assembling supertrees: historically, supertree methods combine source trees with labels only at the leaves and, similarly, produce leaf-labeled supertrees. Thus, if source trees have labels on internal nodes, these are not taken into account. More importantly, it can happen that two taxa belonging to different taxonomic levels and nested into one another, like elephant and mammal, appear as leaves of different source trees. In such cases, the output supertree will erroneously contain both taxa as distinct leaves. From this remark, R. Page challenged researchers in the field to design supertree methods to correctly handle the case of nested taxa. Such methods must allow both the internal nodes and leaves of trees to represent taxa. In these nested-taxa trees, an interior label corresponds to a taxon at a higher taxonomic level than any of its descendants.

8.2 Previous work

There has been very little work related to this problem. Let us first define some notions in this context before reporting related work.

Definition 8.1 *A **nested taxa tree**, or **semi-labeled tree**, is a tree in which all leaves and some internal nodes are associated with labels (i.e. taxa), such that each node has at most one label and each label appears at only one node. This definition applies both to rooted and unrooted trees, though we will here only consider rooted trees.*

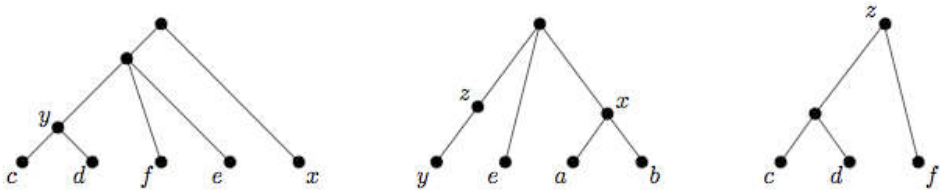

Figure 8.1: A compatible collection of three nested taxa source trees.

Definition 8.2 *Let T be a semi-labeled tree with label set X and let T' be a semi-labeled tree with label set X' with $X \subseteq X'$. Then T'* ancestrally displays *T if, up to suppressing non-root nodes of degree-two, the minimal rooted subtree of T' that connects the elements in X is a refinement of T and, for all $a, b \in X$, whenever a is a descendant label of b in T, then a is a descendant label of b in this rooted subtree.*

A collection $\mathcal{T}$ of semi-labeled trees is ancestrally compatible *if there is a rooted semi-labeled tree T that ancestrally displays each of the trees in $\mathcal{T}$, then we say that T* ancestrally displays *$\mathcal{T}$.*

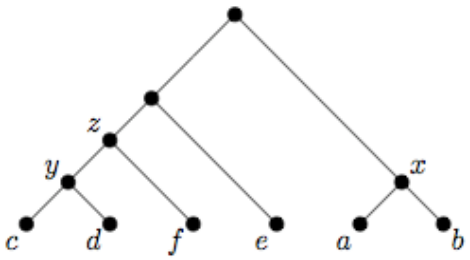

Figure 8.2: A tree ancestrally displaying the collection of Figure 8.1.

These definitions are illustrated on Figures 8.1 and 8.2.

The first computational problem that arises is to find a polynomial-time algorithm for deciding whether or not a collection of nested-taxa source trees is ancestrally compatible and, if so, constructing an appropriate supertree. In response to this problem, [DS04] provided an algorithm called AncestralBuild, generalizing the Build algorithm [ASSU81]. Recall that the latter decides in polynomial time if a collection of rooted leaf-labeled trees is compatible, in which case it returns a leaf-labeled supertree that **displays** the collection[2]. The algorithm AncestralBuild takes a collection $\mathcal{T}$ of nested-taxa trees as input and then outputs a supertree that ancestrally displays $\mathcal{T}$ if such a supertree exists, otherwise it states that the collection is not ancestrally compatible.

Though this is an all-or-nothing algorithm, it has nonetheless important potential use as Llabrés *et al.* have shown that trees in TreeBASE are mostly compatible [LRRV06]. However, the time complexity of AncestralBuild on a collection of k trees spanning n taxa is $O(k^2n^3)$. Such large complexity impedes the use of the algorithm to more than a few dozen trees. This algorithm is then an important milestone, showing that the problem can be addressed in polynomial time, but additional progress is required to handle practical collections.

The Build algorihm uses a top-down process, where each step considers a graph whose vertices bijctively correspond to studied taxa[ASSU81]. The algorithm requires computation of connected components (CC) in all these graphs. However, each graph considered at some step in the process is a subgraph of the graph considered in the previous step, in which some edges are deleted. Thus, the CC computations of the whole process can be avoided if one maintains connected components in a dynamic graph under edge deletion. Henzinger *et al.* relied on sophisticated techniques dedicated to dynamic graphs to propose a fast implementation of Build in the case of fully resolved (i.e. binary) leaf-labeled trees [HKW99]. More precisely, their implementation runs in $O(m \cdot n^{\frac{1}{2}})$ time, where m is the total sum of the number of nodes in each of the source trees and n is the total number of taxa[3].

[1] from the most to the least inclusive one.

[2] see Chapter 2 for the definition.

[3] They also proposed a variant of BUILD running in $O(m+n^2 \log n)$ time but, in the case of compatibility, it typically resulted in a supertree with edges not supported by any part of the data, an undesirable feature for a supertree algorithm.

In 2005, I started a collaboration with C. Semple to solve the problems revealed by M. Sanderson and R. Page. We started with the idea of adapting the techniques used by Henzinger *et al.* to obtain a fast algorithm for deciding ancestral compatibility. However, the algorithm had to accept as input semi-labeled trees (i.e. not only leaf-labeled trees), and trees that can be partially resolved (i.e. not only binary trees). Our findings are detailed below.

8.3 Descendancy graphs

To step from leaf-labeled trees to semi-labeled trees necessitates the use of a different graph than that of Aho [ASSU81]. Indeed, in the latter case, two kinds of relationships between taxa need to be encoded: ancestral relationships between nodes on top of one another in a source tree and incomparability relationships between nodes that are not ancestors of one another in a source tree. To achieve this goal, P. Daniel and C. Semple relied on a mixed graph, i.e. a graph containing both directed edges, called **arcs**, from one vertex to another, and undirected edges between two vertices [DS04]. The **descendancy graph** they defined contains m nodes but a large number of edges and arcs, namely $O(kn^2)$ of both sorts, where k is the number of source trees in the input collection $\mathcal{T}$.

We first proposed a modification of the descendancy graph, called the **restricted descendancy graph**. The latter is of smaller size than the former: it still contains m nodes but now contains only $O(m)$ arcs and

$$O\Big(\sum_{T_i \in \mathcal{T}} \sum_{u \in I(T_i)} d(u)^2\Big)$$

edges, where $I(T_i)$ denotes the set of interior nodes of tree T_i for all i and $d()$ denotes the degree of a node. Note that, depending on the degree of overlap of the source trees and the degree of each of their nodes, the number of edges can range from $O(m)$ to $O(kn^2)$. In particular, if the source trees are all fully-resolved, then the restricted graph contains $O(m)$ nodes, arcs, and edges.

We then proved that running the AncestralBuild algorithm [DS04] on this restricted graph still correctly answers the ancestral compatibility question [BN06, Prop. 4.1].

8.4 A faster algorithm to decide ancestral compatibility

Though we generally reduce the size of the graph that lies at the core of AncestralBuild, the algorithm still runs in the same time bound as before in the worst case, i.e. for collections or highly multifurcating trees.

We thus proposed modifications of the algorithm to reduce its running time even in the worst case. The computing steps of AncestralBuild that have the most influence on the running time are the computation of arc-components (AC) in the restricted graph and the identification of edges between them. We showed that the AC computations can be transferred into CC computations in a smaller ad-hoc graph called the **component graph**. This graph contains $O(n)$ nodes instead

of the $O(m)$ nodes of the restricted graph, i.e. we gain here a k factor since $m = O(kn)$. This graph also facilitates the identification of edges linking different AC of the restricted graph. The modification of AncestralBuild based on this graph was called AncestralBuild*. This graph also has the interesting property that recursive calls issued during an ongoing step of the algorithm consider node disjoint restrictions of this graph. Hence, we can apply similar techniques as those used in [HKW99] to maintain CC thanks to dedicated data structures. As a result, we obtain the following result:

Theorem 8.1 ([BS06]) *Let $\mathcal{T}$ be a collection of rooted semi-labeled trees with $|L(\mathcal{T})| = n$. Then, it is possible to decide if $\mathcal{T}$ is ancestrally compatible, and in this case propose supertree that display $\mathcal{T}$ in*

$$O\big(\log^2 n \cdot \big(\sum_{T_i \in \mathcal{T}} \sum_{u \in I(T_i)} d(u)^2\big)\big),$$

total time.

In general, AncestralBuild* allows for the source trees to be rooted semi-labeled trees of unbounded degree. However, in the special case where the source trees are all rooted binary semi-labeled trees, the running time of AncestralBuild* is $O(m \log^2 n)$. This running time is the same as the running time of the algorithm in [HKW99] whose source trees are all rooted binary *leaf-labeled* trees[4]. Moreover, it is almost linear in the size of the input, which guarantees short execution times even on large datasets.

8.5 Beyond the decision problem

When a considered collection $\mathcal{T}$ of semi-labeled trees is not compatible, we can adopt several strategies to still obtain a supertree thanks to the AncestralBuild* algorithm, e.g. by considering a subset of the input trees or compatible parts of them. However, the most natural solution consists of conceiving an optimization problem to solve when incompatibility is met, i.e. when the restricted descendancy graph is connected. In such cases, to progress in the building of the supertree, we need to disconnect the graph, i.e. to remove some arcs and edges. A first idea is to identify and remove the smallest number of arcs and edges to achieve this goal, with similar ideas as in several other supertree methods [SS00, SWR08]. This procedure would be biased toward large trees, contributing more edges and arcs. As some supertree methods have been criticized for the same behavior [BEB98], it is preferable to first investigate criteria that would not favor trees depending on their size. A natural idea is then to find the smallest number of input trees such that ignoring the arcs and edges they contribute in the graph disconnects it. Unfortunately, this problem is NP-hard [BBES08]. We have then chose another optimization criterion to evaluate the cost of

[4]The running time of their algorithm can be improved from $O(mn^{\frac{1}{2}})$ (as stated in [HKW99]) to $O(m \log^2 n)$ by changing the dynamic connectivity algorithm it uses.

disconnecting the graph and designed a polynomial time algorithm that provides an optimal solution to the problem[5]. Starting from the Splitstree package [HB06], I implemented the algorithm in Java. The implementation performs fine on a *Phocidea* dataset and its results are currently under investigation for several other datasets.

Publication(s) whose material is described in this chapter:

- Fast Computation of Supertrees for Compatible Phylogenies with Nested Taxa, **V. Berry** and C. Semple, *Systematic Biology*, 55(2), U108-U126, 2006.
- Amalgamating source trees with different taxonomic levels, **V. Berry**, O. Bininda-Emonds and C. Semple, in preparation.

[5] as we are in competition with several teams on this topic, I cannot reveal here the details of the algorithm, but they will be available as soon as we submit our paper [BBES08].

Chapter 9

Supertree construction from desirable properties

9.1 Introduction

To assess the relevance of supertree methods, it is most useful to have properties that characterize the extent to which the supertrees they infer are reliable syntheses of source trees [SDB00, WTP+04, Gol05]. For instance, Steel *et al.* suggest that the output supertree should (i) encompass every source tree when possible, (ii) always contain every label (taxon) that occurs in at least one source tree, and (iii) be computable by a polynomial-time algorithm [SDB00]. These authors also showed that rooted input trees are more appealing than unrooted ones for supertree methods that aim to satisfy several desirable properties simultaneously.

In Chap. 2, we have seen that operational supertree methods can mainly be divided into *liberal* methods and *veto* methods, with the latter aiming at proposing supertrees containing only reliable clades at the price of obtaining less resolved supertrees. As of 2004, several supertree methods akin to the veto philosophy had been proposed, all of which were inspired by *consensus* methods, i.e. methods that operate on trees with identical leaf sets. For instance, extensions of the strict consensus [Gor86, HNW99, Bry02], semi-strict consensus [GP02] and maximum agreement subtree consensus [BN04] had been proposed to infer veto supertrees. Technically speaking, veto methods adopt an axiomatic approach, i.e. specify an interesting mathematical property and only propose supertrees verifying this property. However, the above-mentioned veto methods rely on properties that either impose very strong constraints on the supertree, like the Strict Consensus Supertree [Gor86] and SMAST [BN04], or are computationally hard to ensure, like the Semi-Strict Supertree [GP02] and SMCT [BN04]. At that time, I decided to hunt for what seemed to be the holy grail of veto supertree methods, i.e. interesting and computationally tractable properties that supertrees should verify. This study began with E. Douzery, then V. Ranwez later joined us, as well as C. Scornavacca, whose PhD is co-supervised by V. Ranwez and me.

Our starting point was papers criticizing methods that could infer supertrees containing ar-

bitrary clades (e.g. see [BEB98, Gol05]). We then investigated several formalisms according to which the topology of the supertree could be compared to that of source trees and that would allow us to check whether parts of a proposed supertree are really induced by the data. This led us to investigate the *identification* property [SS03, Dan04, GSS07]. In the same time, we wanted a formalism that could enable us to check that a candidate supertree would not contradict the source trees (recall that we are looking for a veto method). Rooted triples then seemed to be the ideal formalism, for which tractable results could be obtained. Eventually, we proposed two desirable properties that veto supertree methods should display and that could be checked in polynomial time for a supertree in regards to a given collection of source trees. We were also able to design supertree methods that would only produce supertrees satisfying these properties.

9.2 Non-contradiction and induction properties

This section gives details on the two important points mentioned above for supertree construction. On the one hand, supertree methods should avoid arbitrary resolutions, i.e. resolutions that are not entailed by the source topologies. Indeed, novel relationships displayed by a supertree "*are worrying if they are not implied by combinations of the input trees*" [WPCC05], and "*should be identified as such, to highlight their lack of any known justification*" [PW02]. Thus, we first wanted a way to verify that every piece of phylogenetic information displayed in a supertree was present in one or several source topologies, or induced by them collectively; we called this the **induction property** (PI).

On the other hand, as we adopted a veto view point, we focused on ways to distinguish between unanimous and contradicted clades of a supertree with respect to a collection of source trees. In the veto approach, a supertree is not allowed to contain a clade that conflicts either directly with a source tree or indirectly with a combination of them. We call this the **non-contradiction property** (PC). Supertree clades that are detected to not verify this property could then be removed from the supertree, hence leading to a supertree that can be considered as a reliable baseline for subsequent analyses, as e.g. suggested by Goloboff [GP02, Gol05].

Last, inferred supertrees often contain multifurcating nodes, reflecting the impossibility for the inference method to choose among the possible evolutive scenarios for the subtrees hanging from the node. Actually, this intermixes two distinct phenomena: either a lack of overlap in the topological information of the source trees (hence, the impossibility to resolve the clade) or the occurrence of topological conflicts among them. PI and PC are key properties to distinguish between these two phenomena. With this in mind, we later included specific information in the supertrees inferred by methods we designed.

But let us now concentrate on the property definitions.

Definition 9.1 *Given a rooted triple t, let $\bar{t}$ denote any of the two other rooted triples on the same set of labels. A tree T is said to **display** a set $\mathcal{R}$ of triples when $\mathcal{R} \subseteq \mathsf{rt}(T)$; moreover, T **strictly displays** $\mathcal{R}$ if additionally $L(T) = L(\mathcal{R})$.*

To find a tree displaying a triple set $\mathcal{R}$, it is useful to take into account that some triples of the tree are *induced* by $\mathcal{R}$:

Definition 9.2 *A compatible set $\mathcal{R}$ of triples **induces** a triple t, denoted by $\mathcal{R} \vdash t$, if and only if $\mathcal{R} \cup \{\bar{t}\}$ is not compatible, or equivalently if any tree T that displays $\mathcal{R}$ contains t. We can extend this definition to the case of incompatible collections in the following way: say that a set $\mathcal{R}$ of triples **induces** a triple t when there is a compatible subset $\mathcal{R}'$ of $\mathcal{R}$ that induces t.*

The induction process mentioned in the above definition has been known for quartets and triples for roughly 20 years [BD86, Dek86]. The possible inductions have mainly been grouped in *induction rules*. For instance, the following rule is well-known:

$$\{AB|C, BC|D\} \vdash AC|D,$$

i.e. any tree displaying $AB|C$ and $BC|D$ also has to display the triple $AC|D$. Since they were introduced [BD86, Dek86], induction rules have been used in a number of papers (see [GSS07] for a recent development).

Definition 9.3 *Given a collection $\mathcal{T}$ of input trees and a candidate supertree T, let $\mathcal{R}(T,\mathcal{T})$ denote the set of triples of $\mathcal{T}$ for which T proposes a resolution. More formally, $\mathcal{R}(T,\mathcal{T}) = \big\{AB|C \in \mathsf{rt}(\mathcal{T})$ such that $\{AB|C, AC|B, BC|A\} \cap \mathsf{rt}(T) \neq \emptyset\big\}$.*

Set $\mathcal{R}(T,\mathcal{T})$ corresponds to the topological information present in collection $\mathcal{T}$ relative to that present in supertree T. Using these notations, we can express the induction property (PI) and the non-contradiction property (PC) as follows:

Definition 9.4

- *T **satisfies PI for** $\mathcal{T}$ if and only if for all $t \in \mathsf{rt}(T)$, it holds that $\mathcal{R}(T,\mathcal{T}) \vdash t$. In other words, PI requires that each and every triple of T is induced by $\mathcal{R}(T,\mathcal{T})$.*
- *T **satisfies PC for** $\mathcal{T}$ if and only if for all $t \in \mathsf{rt}(T)$ and all $\bar{t}$, it holds that $\mathcal{R}(T,\mathcal{T}) \nvdash \bar{t}$. This means that, for each and every triple of T, $\mathcal{R}(T,\mathcal{T})$ induces no alternative resolution.*

For instance, considering collection $\mathcal{T} = \{T_1, T_2\}$ and supertree T' in Fig. 9.1, the set $\mathcal{R}(T',\mathcal{T})$ is $\{AC|E, AC|F, AB|E, AB|F, BC|E, BC|F, EF|A, EF|B, EF|C\}$. Note that the triple $AD|C$ present in $\mathsf{rt}(\mathcal{T})$ due to T_2 is not in this set because A, D, C are multifurcating in T'. Note also that when considering supertree T of the same figure, $\mathcal{R}(T,\mathcal{T})$ contains two different triples for taxa A, B, C. For this reason, T does not satisfy PC for $\mathcal{T}$. Trees T and T" satisfy both PI and PC. However, tree T" is preferable, as it is more informative.

When the source trees are compatible, any reasonable method is expected to produce a supertree satisfying PC. However, some methods can propose a supertree that does not satisfy PI, by

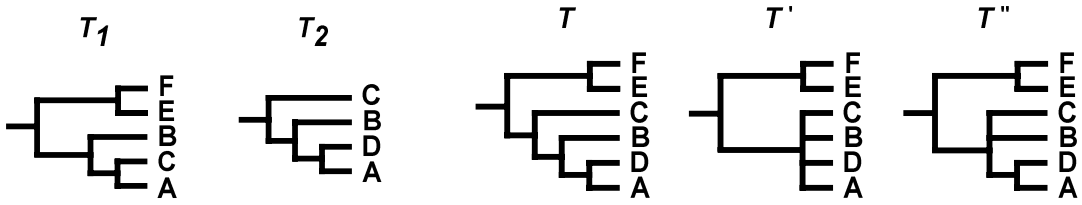

Figure 9.1: Two source trees T_1 and T_2 and three possible supertrees T, T', T".

proposing one of the numerous trees that display the source trees (there can be an exponential number of such trees). However, when $\mathcal{T}$ is compatible, it is possible to find a supertree that displays all triplets of the collection and is also refined by all other possible supertrees. This corresponds to the *identification* property between trees:

Definition 9.5 *A set $\mathcal{R}$ of triples is said to **identify** a tree T if and only if T strictly displays $\mathcal{R}$ and T is refined by every tree T' that strictly displays $\mathcal{R}$.*

A set $\mathcal{R}$ can identify at most one tree thus, when this happens, this tree is a *canonical* representation of all possible supertrees. It is unlikely that for practical collections $\mathcal{T}$, the set $\mathsf{rt}(\mathcal{T})$ identifies a tree. Nevertheless, it is possible that a subset of $\mathsf{rt}(\mathcal{T})$ identifies a tree T, and then the topological information contained in T exactly corresponds to a subset of the topological information contained in $\mathcal{T}$. Such a subset is most interesting when the triples t it contains do not have an alternative resolution $\bar{t}$ in $\mathsf{rt}(\mathcal{T})$. We showed that this situation actually occurs for the subset $\mathcal{R}(T,\mathcal{T})$ of $\mathsf{rt}(\mathcal{T})$ precisely when the supertree T satisfies both PI and PC.

Property 9.1 ([RBG$^+$07]) *Given a collection of rooted trees $\mathcal{T}$, a supertree T satisfies PI and PC for $\mathcal{T}$ if and only if $\mathcal{R}(T,\mathcal{T})$ identifies T.*

We later found out that P. Goloboff and D. Pol had evoked similar properties [GP02]. They mention as interesting: "*the property of [the supertree] displaying AB|C if it is found in some input tree or implied by some combination of input trees and no input tree or combination of input trees displays or implies AC |B or BC |A*". From this citation, we can obtain properties PI' and PC', whose definition is close to that of PI and PC, but still different. However, we showed several cases where PI and PC are preferable to PI' and PC' , and additionally the latter does not seem easy to verify. In contrast, PI and PC can be checked in polynomial time. To show this, we proposed the following equivalent properties, whose formulation is less intuitive but whose checking is easier.

Definition 9.6 *Let $\mathcal{T}$ be a collection of rooted source trees and T be a proposed supertree for $\mathcal{T}$. Define PC_{eq} and PI_{eq} to be the following properties:*

- *PC_{eq}: $\mathsf{rt}(T) \cup \mathcal{R}(T,\mathcal{T})$ is compatible.*
- *PI_{eq}: for any $t \in \mathsf{rt}(T)$ and for all $\bar{t}$, the set $\{\bar{t}\} \cup \mathcal{R}(T,\mathcal{T})$ is incompatible.*

Property 9.2 ([RBG$^+$07])

(PI_{eq} and PC_{eq}) $\Leftrightarrow$ (PI and PC).

More precisely, PC and PC_{eq} are equivalent but PI_{eq} is equivalent to PI only when PC_{eq} holds. PI_{eq} and PC_{eq} can easily be checked by using the BUILD algorithm [ASSU81], which indicates in polynomial time whether a set of rooted trees is compatible or not. A similar procedure was proposed by [Ste92], and refined by [Dan04]. However, for the case of PC, we proposed a faster linear time algorithm [RBG$^+$07].

9.3 An algorithm computing supertrees that verify PI and PC

I briefly discuss now a method that from the start produces supertrees verifying the PI and PC properties. I should point out that we want more than that, since the uninformative star tree verifies PI and PC simply because it does not resolve any triplet. Our precise goal was to produce a method that infers supertrees that verify PI and PC *and* that contain as much resolution as possible, e.g. resolve as many triples as possible. More precisely, we require a method that, given any collection $\mathcal{T}$, proposes a supertree T such that $\mathcal{R}(T,\mathcal{T})$ identifies T and $\mathcal{R}(T,\mathcal{T})$ has maximum size over all such subsets of $\mathsf{rt}(\mathcal{T})$. We proved that computing such a subset of $\mathsf{rt}(\mathcal{T})$ is an NP-hard task. We thus proposed a heuristic algorithm, but only approximate in the size of the subset, i.e. we imposed that inferred supertrees always verify PI and PC. This algorithm is a variant of the BUILD algorithm and is called PhySIC – *Phylogenetic Signal with Induction and non-Contradiction*[1]. We showed that it can be implemented to run in $O(kn^3+n^4)$ time on a collection of k rooted trees spanning a set of n labels. We proposed an implementation of this algorithm and a webpage from which it can be run online (`http://www.atgc-montpellier.fr/physic`). The inferred supertrees contain internal labels at multifurcating nodes to indicate whether they originate from contradiction issues or from a lack of overlap (or both) between source trees. Hopefully, this will help biologists in deciding on further analyses to resolve these nodes.

We showed the relevance of the produced supertree with respect to other supertree methods on two biological datasets on the historical group of primates [Pur95]. We studied this group at the genus level by combining 24 source trees covering 95% of its genera. I encourage the reader to have a look at the resulting supertree, displayed at the end of the corresponding paper, included the last pages of the final part of this manuscript.

In simulations, PhySIC inferred on average less than 1% incorrect edges. Such a very low type I error rate is an expected characteristics of veto methods.

[1]Note that, unlike BUILD, PhySIC always returns a supertree.

9.4 A more involved algorithm

The PhySIC algorithm was a first step in that its purpose was mainly to show that polynomial-time supertree algorithms with interesting properties can be designed. In the simulation we did hereafter, we noted that this algorithm is very sensitive to the difficulty of the considered instances, proposing highly unresolved trees for the most difficult instances. This shows that in some cases it is not good heuristics to obtain the most resolved supertree verifying PI and PC. To remedy this situation, we designed a new algorithm inferring supertrees that verify PI and PC by either incorporating multifurcating nodes in the supertree – as PhySIC – or by not including some taxa in the supertree – similar to the SMAST and SMCT methods (see Chap. 7). The rationale is that excluding taxa whose position greatly differs among source trees from the analysis allows us to obtain much more resolved, hence more informative, supertrees.

In the PhySIC study, we expressed the amount of information of a supertree as the number of triples it contains. This shortcoming is not well suited when some input taxa are no longer in the supertree. As a supertree verifying PI and PC is in essence a common denominator of all possible fully-resolved supertrees for the data, we then chose to measure the information of a supertree as a function of the number of binary plenary supertrees by which it is compatible. For this purpose, we proposed a variant of the Cladistic Information Content (CIC) measure [TWC98], that is denoted CIC*. We also changed the approach with which the supertree was built. Instead of using a variant of the top-down BUILD algorithm that could be blocked at some step due to a difficult situation, we resorted to an iterative insertion procedure [SvH96, RG01]. The resulting algorithm was called PhySIC_IST (*PHYlogenetic Signal with Induction and non-Contradiction Inserting a Subset of Taxa*). Its results on simulated data are far better than those of PhySIC in terms of resolution: the improvement over PhySIC according to the CIC* measure is 1.5-fold. This is a considerable improvement, as the measure is valued on a logarithmic scale. In return, PhySIC_IST shows higher type I error than PhySIC due to the fact that it proposes more clades in a supertree. However, its type I errors remains under 1% and decreases even more as the number of source trees grows. PhySIC_IST clearly offers a better tradeoff between resolution and accuracy for dealing with practical datasets.

9.5 Navigating between veto and liberal methods

Consider the case of a biologist inferring a supertree for a given collection of source trees, according to any chosen method. After this first computation, as he accumulates more source trees, e.g. appearing in new individual studies published in the literature or obtained by scanning databases for newly discovered genes, he will complete his original set of source trees and infer a new supertree. In doing so, he hopes to obtain a more complete and resolved picture of the taxonomic group he is fond of. However, there can be disappointing cases in which he will actually obtain a less resolved supertree. The first reason is that the new taxa hosted by the added source trees by far outnumbers added trees, in which case it has on average lowered the overlap between

the source trees. This inevitably leads to less average resolution in the supertree, whatever the method. Alternatively, he could obtain less resolution due to additional conflicts on the position of taxa already included in the initial collection. In fact, by adding new trees, one can only increase the number of conflicts on these taxa. If this number of conflicts remains small, liberal supertree methods will infer more resolved trees, but veto methods will inherently tend to infer less resolved supertrees as they are not allowed to propose clades contradicting any source tree. Indeed, adding more trees provides more information on the relative position of some taxa, but also increases the number of local conflicts. This means that pure veto methods cannot satisfactorily handle large collections of trees. For such instances, one has to use the liberal approach that allows us to arbitrate between conflicts arising among source trees. These methods will take ad-hoc decisions, according to their objective criterion, when dealing with individual conflicts that occur during the tree-building process.

Lately, we have explored a new way to arbitrate between conflicts in large collections of source trees. We proposed a statistical preprocess to detect and correct anomalies in source trees. A veto method can then be applied to obtain a supertree not contradicting the retained information. This approach has the advantage over liberal methods of making the removal of conflicts between source trees explicit and parametrizable.

The preprocess we proposed, called STC (Source Trees Correction), analyzes contradictions among the source trees on the basis of the rooted triple formalism; for all direct contradictions[2], it considers the possible topological alternatives and drops the alternative(s) that is (are) statistically less supported. The level of significance required to drop an alternative is a value related to a confidence threshold in a statistical test and can be chosen by the user. Then STC modifies each source tree (using an algorithmic schema similar to that of PhySIC_IST) so that it does not contain any dropped alternative and yet remains as informative as possible. In other words, STC aims at correcting source trees that propose anomalous phylogenetic positions for some taxa (due to lateral gene transfers, long branch attractions, paralogy, etc.).

If the user approves the proposed modifications, the PhySIC_IST veto method is then applied to the modified source trees. The resulting supertree satisfies both PI and PC properties for the collection of modified source trees. If the user is not satisfied with the modified source trees, he can change the threshold and restart the procedure, or choose to skip it. In this way, the liberal component of the supertree inference is not only made to be explicit but also interactive and parameterizable.

We proposed an implementation of the STC preprocess and of the PhySIC_IST method that is available for download or online execution (`http://www.atgc-montpellier.fr/physic_ist/`).

We applied the STC preprocess to the PhySIC and PhySIC_IST veto methods in a simulation study. Figures B.1, B.2 and B.3 in Appendix B detail the results obtained for the supertree methods under study in the reconstruction conditions we simulated. When setting the STC threshold at 95% (i.e. a 5% probability that a detected anomaly is not actually an anomaly), the resolution of

[2] i.e. 3-leaf set $\{A,B,C\}$ such that both $AB|C$ and $AC|B$ are in the source trees

supertrees inferred by both methods greatly increases. Moreover, the type I error rate of the two methods does not change significantly when applied after source trees have been modified by STC. This shows that the preprocess corrects the source trees in an appropriate way. Lastly, when more source trees are added, the number of input taxa not appearing in the supertree decreases and more informative trees are obtained. Thus, as more information is provided, the STC fully plays its role of liberal preprocess in allowing the user to obtain more informative supertrees.

The case where the overlap between source trees is very weak is an exception to the results reported above. In such a situation (for small collections of small trees in comparison to other studied conditions), the STC preprocess is not able to detect anomalies to a significant extent since it has too few rooted triples at hand. Hence, supertree methods output by the veto methods retain the same characteristics as when not using the STC preprocess.

To conclude, it seems that the above STC preprocess allows to combine the advantages of both veto and liberal methods: informative supertrees are obtained that also verify interesting properties with respect to the (modified) source trees.

Publication(s) whose material is described in this chapter:

- Votez veto pour l'Arbre de la Vie, **V. Berry**, V. Ranwez, P.-H. Fabre, E.J.P. Douzery, *Journées Ouvertes Biologie, Informatique et Mathématiques* (JOBIM), p. 251-263, 2006.
- PhySIC: a Veto Supertree Method with Desirable Properties, V. Ranwez, **V. Berry**, A. Criscuolo, P.-H. Fabre, S. Guillemot, C. Scornavacca and E.J.P. Douzery, *Systematic Biology*, 56(5), 293-304, 2007.
- PhySIC_IST: cleaning source trees to infer more informative supertrees, C. Scornavacca, **V. Berry**, E.J.P. Douzery and V. Ranwez, *BMC Bioinformatics* (to appear).

Part V

Ongoing and future research

Chapter 10

Ongoing research

I am currently involved in several collaborative studies, one of which is the ongoing work with C. Semple and O. Bininda-Emonds described in the previous part of this manuscript.

Co-supervision of Celine Scornavacca's PhD work with V. Ranwez, led to a collaboration with V. Daubin on multi-gene families that started last year. More precisely, handling such data necessitates tree-comparison algorithms that are able to deal with trees containing several copies of the same labels. As a first simple solution, we are currently investigating preprocessing algorithms to decompose such trees with the idea of feeding them as input into the PhySIC_IST algorithm detailed in Chap. 9. This collaboration has greatly motivated us and we decided to write an ANR project proposal. Since our proposal was accepted, this collaboration will last for several years (see next chapter for more details). A paper on multigene trees should be submitted to the *RECOMB'09* conference.

For a year, I have been co-supervising the PhD work of Philippe Gambette, in collaboration with C. Paul. During his first year, Philippe worked on networks as a tool to represent phylogenetic relationships and especially on a combinatorial formalism to decompose increasingly complex networks. In this context, we also started a collaboration with D. Huson and R. Rupp on algorithms to infer rooted networks of minimum complexity from collections of rooted trees. A paper relating these results should be submitted to the *RECOMB'09* conference.

A collaboration with T. Gernhard, V. Ranwez and A. Jean-Marie was started in June on consensus methods explicitly taking tree evolution models into account. We obtained several algorithms that should lead to consensus trees whose clade content should be less biased by down-weighting clades that are most likely to appear frequently via the models underlying the input trees. A paper should be submitted to the *Molecular Biology and Evolution* journal in the next six months.

Another collaboration with V. Ranwez concerns the TreeBASE phylogenetic database. Together with students S. Pourali and N. Clairon that we supervised, we designed a website to explore and correct a collection of source trees. By resorting to several taxonomic resources, this website provides statistics on the characteristics and overlap of source trees, helps in locating unknown or misspelled taxon names and indicates TreeBASE trees that were published with a similar

taxonomic focus. A short paper has been submitted to the *BMC Bioinformatics* journal.

I am also involved in some collaborations with local members of LIRMM. A collaboration with C. Caraux started last year on resampling methods to obtain accurate congruence measures of collections of source trees. Our work is mainly based on a critique of the method proposed by Lapointe and Rissler involving the MAST consensus method [LR05]. We propose a method that better accounts for the influence of source tree topological traits on MAST scores computed at intermediary steps of the method. A first variant of the method was successfully experimented in a co-evolution study on fig wasps and fig trees [JvNB$^+$08]. We also extend the method to the case of trees having different though overlapping taxon sets.

Following supervision of Sushant Gupta, an Indian student, I also started a collaboration with L. Brehelin on a variant of the MRP supertree method. This variant relies on statistical models to extract topological information from a set of source trees. The collaboration has just begun but looks promising as MRP is the most known supertree method and since our approach is likely to obtain more information from the source trees than can be obtained with the current signal extraction protocol. This would benefit the MRP method when faced with source trees having little overlap, i.e. in such cases the method has been shown to perform poorly.

More recently, I had an animated discussion with F. Chevenet and R. Christen concerning a scripting language for graphical display of phylogenetic trees that the former is currently developing. We used this language, called ScripTree, as an intermediate step to set up webpages enabling users to browse modifications proposed by the PhySIC_IST program on collections of source trees considered for building supertrees. An application note on ScripTree should be submitted in the following month to the *Bioinformatics* journal.

Chapter 11

Plans for the future

11.1 Introduction

In the next few years I will conduct research within the framework of the long-term ANR project that we just got funds for and that I will coordinate. This project is a collaboration with F. Chevenet, V. Daubin, E. Douzery, N. Galtier, C. Paul, V. Ranwez and E. Tannier. Our project as several goals, including the building of large phylogenies by integrating, on a large scale, both micro-evolutionary events, such as sequence substitutions, and macro-evolutionary events, such as duplications/losses (D/L) and horizontal transfers (HGT) that may occur in genes. I believe that this is a necessary step in assembling the tree of life, and gene sequences produced by the numerous ongoing genome sequencing projects is an important source of data for such ends. The data we will focus on are multi-gene families, i.e. genes existing in several copies in the genome of studies species. Indeed, very few phylogenetic methods are able to deal with such genes, though they represent the vast majority of genes. Our application mainly concerns bacteria where HGT are relatively frequent and interleaved with D/L events. Bacteria are an economically important target due to their involvement in many aspects of human life (symbiotic relationship for digestion, responsible for lethal infections, tool in food production, etc). In the process, we hope to obtain useful information on D/L and HGT events in bacteria, whose mechanics remain highly unknown. A long-term application is in understanding the spread of antibiotic-resistance among pathogenic bacteria. Our first taxonomic group of interest will be a group of proteobacteria for which members of the project have expertise. Ultimately, we hope to be able to consider the history of the major groups of bacteria, i.e. to determine the early relationships of these groups, annotated with flows of HGT events between them.

Below I give a brief overview of why we think there is important research to be conducted in this direction, what kind of data we can use and what strategies we plan to employ.

11.2 Setting

The tree of life. As mentioned previously in this manuscript, it is generally thought that all existing life on Earth has evolved from a single common, very distant ancestor from which new species originated and evolved following a tree-like pattern, up to the current contemporary species represented as leaves in this tree of life. Currently, 1.8 million species have been inventoried and 0.01 million are discovered every year [Bou07]. Yet, the relative position of the major groups of species on the tree of life is still highly debated, e.g. 64% of the nodes of the well-known NCBI estimation of the tree of life are currently unresolved. In particular, no significant progress has been achieved since the 1970s on the relationship of the earliest groups of bacteria [WRG$^+$01, BDI$^+$01, DGP02] and recent developments have radically revised the relative branching and roots of the eight major eukaryote groups, suggesting "*major gaps in our understanding simply of what eukaryotes are*" [Bal03, p.1703]. This shows that much work remains to be done before we obtain an acurrate picture of the evolutionary relationships linking early living organisms.

Data to infer the tree of life. A considerable amount of data is routinely produced by sequencing projects and readily made available on the Internet. The current trend is to obtain complete genome sequences, as this plays a key role in understanding functions of genomic elements and in predicting precise gene locations, which has important medical applications. The availability of complete gene repertoires also greatly benefits the task of recovering (ancient) evolutionary relationships between living organisms [LDOM05]. The first complete genome was obtained only 10 years ago, but renewed progress in sequencing technologies in the last few years has led to the sequencing of 730 complete genomes, spanning the major life groups. These were subsequently scanned for the presence of genes, annotated with additional information, and then released in the public domain. Moreover, more than 3600 new genomes are being sequenced and will be released in the following years. Thus, plenty of data is available to infer the tree of life or parts of it, and the task is now to infer something meaningful from this flood.

Horizontal gene transfers. When gene sequences are used to estimate the tree of life, the main difficulty lies in separating the signal due to speciation events that accounts for the structure of the species tree from the signal due to gene-specific evolutionary events such as HGT and D/L events. HGT appears to have played a crucial role in the history of life [GDL02]. Not only have they been of great importance in many evolutionary transitions, such as colonization of new environments, but they also are still a preferred means of adaptation in many organisms. In pathogenic bacteria, for instance, HGT is a preponderant cause of antibiotic resistance, and allows some bacteria to be resistant to most, if not all known antibiotics. Therefore, recognizing the evolutionary origins of genes in genome, and the length of time they have been part of the genome, i.e. knowing when and where HGT occur, cannot only benefit assembly of the tree of life but is also important in understanding genomes and their evolutionary mechanisms.

Possible approaches. Several approaches have been proposed to identify horizontally transferred genes in a genome. Some are based on comparing the features of genes in a genome, and distinguishing those that contrast with the genome-wide pattern. Others compare the gene repertoires of closely related genomes to identify newly integrated genes. However, both of these approaches are limited in terms of the information they provide. First, they only point to genes that have been very recently acquired. Secondly, they can only be applied in favorable conditions, i.e. when the considered genomes display sufficiently homogeneous features so that alien genes can be recognized. And finally, they provide no information on the origin of these alien genes. There is now a general consensus among evolutionary biologists that phylogenetic analysis is the best way to reliably and fully document HGT events. Reconstructing the history of a gene theoretically reveals even ancient HGT in any sequenced genome and enables identification of putative donor organisms. However, several difficulties are encountered when identifying HGT based on a phylogenetic approach. First, HGT have to be distinguished from D/L events, as both contribute to driving gene and species histories apart. Secondly, in order to identify atypical gene histories, one needs to know the actual history of the species. To summarize, when inferring a species tree, it is essential to know what HGT events are at stake in gene histories, which in turn can be determined in comparison to the species tree. These interdependent relationships show that both horizontal (HGT) and vertical (speciations, D/L) events have to be estimated in a well-coordinated effort.

The data complexity. Each gene history is the result of a multi-scale stochastic evolutionary process: at the lowest level, individual nucleotide sites are submitted to micro-events, i.e. substitutions, insertions and deletions, and at the highest level, HGT and D/L macro-events take place in genes. Thus, biological data resulting from this process is inherently complex. An extremum of complexity is encountered when dealing with multicopy genes, i.e. genes found in several copies in one or more genomes. However, previous attempts at reconstructing the history of life have tried to avoid considering this complexity, e.g. by selecting only the few genes that were ubiquitous (i.e. present in all considered organisms) and that seemed to have simple histories, in particular without duplications [CDvM$^+$06]. This view gives a very incomplete if not anecdotal vision of evolution, as only at most 30 genes can be considered in such analyses. Therefore, these approaches have been criticized as representing a "*tree of 1%*" [BSE$^+$07]. In addition, even these genes cannot be considered *a priori* as free from HGT.

To reconstruct the tree of life and explain its incongruences with gene trees and those between gene trees, the vertical signals, i.e. speciation and duplication events, and horizontal signal, i.e. genetic exchanges between distinct species must be disentangled.

Previous work. Literature written on the subject mostly involves, (i) standard phylogenetic analyses, in contexts within which HGTs are expectedly low [LDOM05], (ii) reports on specific HGT events for specific genes [NLSD01, CGP05, For02, DBAG07], (iii) approaches that voluntarily do not rely on gene phylogenies (e.g. gene content analyses [DM07]), and (iv) quasi-philosophical statements about the relevance of tree-like representations in the HGT context [DB07]. Methods investigating the network model to represent evolution – as an alternative to the tree model

– have studied the degree of “reticulate evolution" in a gene, or a set of genes [KH08], but these approaches do not handle multicopy genes. Moreover, this work mainly identifies conflicting phylogenetic signals, and hence do not propose specific gene evolution scenarios (including HGT and D/L events). Concerning the tree model, some algorithmic studies have begun to consider the reconciliation of the phylogenetic history of a gene family with a given species tree. This field is still at the budding age: algorithms often handle a single copy of each gene in one species, and those considering multicopy genes generally avoid the real complexity of the problem by focusing either on duplications [Ste99, LV03, ABLS04, DHV05, VSGD07] or HGT [HL01, ABHL03, LV03], while overlooking more elaborate scenarios with mixtures of these events. However, some authors have started to study the feasibility of incorporating all of these events in a single model to reconciliate a single gene tree with a given species tree [Gor03, HL04], while allowing multicopy gene trees. This is progress over former reconciliation studies [Cha98, PC98]. However, the additional complexity of considering several gene trees simultaneously has not yet been examined, nor has the case where the reference species tree is not given but instead has to be estimated. These are the most complex but by far most relevant variants we will have to study in order to reach our goals.

11.3 Important steps

Here is a brief account of the important steps we plan to follow.

The first step involves elaborating a protocol to collect data from genome databases. Several options exist for splitting gene sequences into gene families, but also for building gene trees and removing uncertain parts of them. In this task, we will be inspired by the ENSEMBL, HOGENOM, PhyloFact, GeneTree and COG databases but will likely need to set up our own protocol according to the specificity of the problems we want to solve.

Then we will need to design criteria and algorithms to assess the accuracy and inherent complexity of the gene families and trees, but also to classify the families in different categories. These criteria will be useful for testing reconstruction algorithms on data of increasing difficulty. Moreover, when this will be needed, we will be able to separate gene families into subsets having different estimated ratios of vertical vs horizontal levels of sequence inheritance.

The main part of our research activity will be devoted to algorithms with thousands of gene families as input, including multi-copy genes, from which evolutionary scenarios will have to be built, compared and combined. Scenarios proposed by different gene trees may be correlated or, in contrast, incompatible with each other and the complexity of simultaneously considering thousands of gene trees in this context has not yet been studied. Two strategies are planned that respectively rely on supertree and supermatrix methodologies.

In the supertree strategy, micro- and macro-events that drive genetic evolution are considered in two separate steps: first, probabilistic models of evolution are used to build gene trees from sequence similarities, i.e. guided by the signal due to the inheritance of genes from ancestors to offspring; then, gene tree topologies are compared in a combinatorial analysis to detect gene HGT and D/L events. The difficulty in this strategy is clearly the second step, as several algorithms

have been proposed to build a species tree from gene trees but only reconciliating the gene trees by D/L events [Ste99, HL01, LV03] or alternatively HGT [GWK05]. Algorithms that take both into account [HL04] are limited to considering single gene family trees. We will have to go beyond this point as our project requires us to consider several gene trees and compare them jointly to a tentative species tree in order to simultaneously infer the species tree and macro-events to reconcile the gene trees. For this purpose, we will extend reconciliation algorithms [HL04] to arbitrate between different scenarios integrating all kinds of macro-events. Consideration of several gene trees at once may paradoxically make some algorithms easier, since the reconciliation of every gene tree will surely benefit from the information deduced from the reconciliation of other gene families. Though underlying problems are likely NP-hard, fixed-parameter tractability is a workaround technique that has been applied several times to reconcile gene and species trees [Ste99, HL04, VSGD07] and in which we already have some experience (e.g. see Part. III of this manuscript).

The supermatrix strategy simultaneously considers micro- and macro-events responsible for gene evolution. To implement this strategy, evolutionary models will have to be designed that integrate standard micro-event models with D/L and HGT events, so that the presence and location of such events will be inferable in a maximum likelihood framework – the approach that provides the most accurate phylogenetic reconstruction results. Then the species tree will be estimated on the basis of these annotated gene histories by specific algorithms to be designed. This is an innovative approach as only [ABLS04] proposed a comprehensive model integrating micro- and macro-events, but their model does not take HGT events into account. The lock-in of this approach is in estimating with sufficient accuracy and speed, the rate of HGT and D/L events, which are at a meta-level compared to substitutions affecting individual sequence sites. As regular parameters of the model, these rates will be estimated from the gene sequences, but the difficulty lies in the fact that transfer and duplication rates seem to be specific to lineages, so a non-trivial number of parameters will have to be estimated. The difficulty will then be in choosing an appropriate complexity level in the model design to allow for both accurate and efficient computations. The supermatrix approach might require high computing times due to the optimization starting from the mere sequences and to the estimation of a non-trivial number of parameters. These times will be reduced by grouping data into “modes" of congruent signals and by sharing parameters among genes with a similar profile.

Once the developed algorithms have provided a putative species tree together with HGT and D/L events affecting individual genes, we will set up a database to register all inferences. We will also provide specific software to allow efficient visualization of the inferences and facilitate their mining.

Exciting research in computational biology and bioinformatics thus awaits us in this large-scale collaborative project.

Part VI

Appendix

Appendix A

Efficiency of quartet cleaning as measured by experiments

These results are taken from [BJK+99].

Quartet Inference Method	Mean Edge Length	Global Edge (% of Trees Cleanable)			Local Vertex (Average % of Vertices per Tree that are Cleanable)		
		Sequence Length			Sequence Length		
		50	200	2000	50	200	2000
Maximum Parsimony	0.025	10.48% (0.45%)	62.85% (10.85%)	95.02% (56.67%)	38.98% (29.20%)	76.15% (59.82%)	94.80% (87.27%)
	0.1	38.62% (0.22%)	79.60% (6.17%)	92.34% (27.06%)	55.91% (28.82%)	80.76% (55.57%)	89.90% (74.01%)
	0.25	26.88% (0.01%)	64.44% (0.71%)	78.48% (6.83%)	45.18% (20.09%)	67.96% (37.73%)	77.03% (53.93%)
	0.5	3.25% (0.00%)	23.46% (0.03%)	56.33% (0.87%)	23.50% (10.59%)	43.47% (22.38%)	61.71% (38.42%)
	0.75	0.30% (0.00%)	3.81% (0.00%)	27.46% (0.08%)	13.08% (6.03%)	25.46% (14.11%)	46.57% (28.48%)
Neighbor Joining	0.025	16.36% (0.73%)	70.47% (13.49%)	95.67% (57.18%)	43.79% (32.53%)	79.85% (62.96%)	95.06% (87.46%)
	0.1	47.10% (0.34%)	81.36% (6.92%)	91.72% (24.57%)	60.86% (32.04%)	81.57% (56.65%)	88.97% (72.31%)
	0.25	30.85% (0.02%)	64.18% (0.84%)	76.48% (6.16%)	47.69% (23.19%)	67.50% (38.44%)	75.48% (52.55%)
	0.5	3.91% (0.00%)	22.27% (0.07%)	51.80% (0.79%)	26.91% (16.37%)	43.25% (26.27%)	58.88% (38.08%)
	0.75	0.41% (0.00%)	3.63% (0.00%)	21.47% (0.06%)	16.78% (11.33%)	27.56% (19.41%)	43.48% (30.49%)
Ordinal Quartet Method	0.025	20.49% (1.21%)	74.33% (17.11%)	96.12% (59.25%)	47.79% (37.36%)	82.19% (66.38%)	95.45% (88.19%)
	0.1	64.15% (1.21%)	86.91% (12.38%)	93.84% (34.53%)	71.92% (40.32%)	86.32% (63.45%)	91.50% (78.00%)
	0.25	63.64% (0.16%)	81.72% (2.42%)	87.31% (13.82%)	68.85% (30.29%)	81.13% (47.25%)	85.44% (65.10%)
	0.5	39.03% (0.04%)	67.26% (0.19%)	81.75% (1.82%)	52.41% (23.67%)	69.18% (32.06%)	79.47% (45.22%)
	0.75	12.32% (0.00%)	35.18% (0.01%)	63.60% (0.16%)	34.28% (18.56%)	49.52% (24.86%)	66.47% (33.28%)

Table A.1: Performance of the Quartet Cleaning Algorithms Under Simulation. The unparanthesized number is the average percent of all evolutionary trees (vertices per evolutionary tree) that have no quartet errors (no quartet errors across them) after global edge cleaning (local vertex cleaning) has been applied. The parenthesized number is the average percent of all evolutionary trees (vertices per evolutionary tree) that have no quartet errors (no quartet errors across them) before quartet cleaning is applied.

Appendix B

Efficiency of supertree methods in various reconstruction conditions

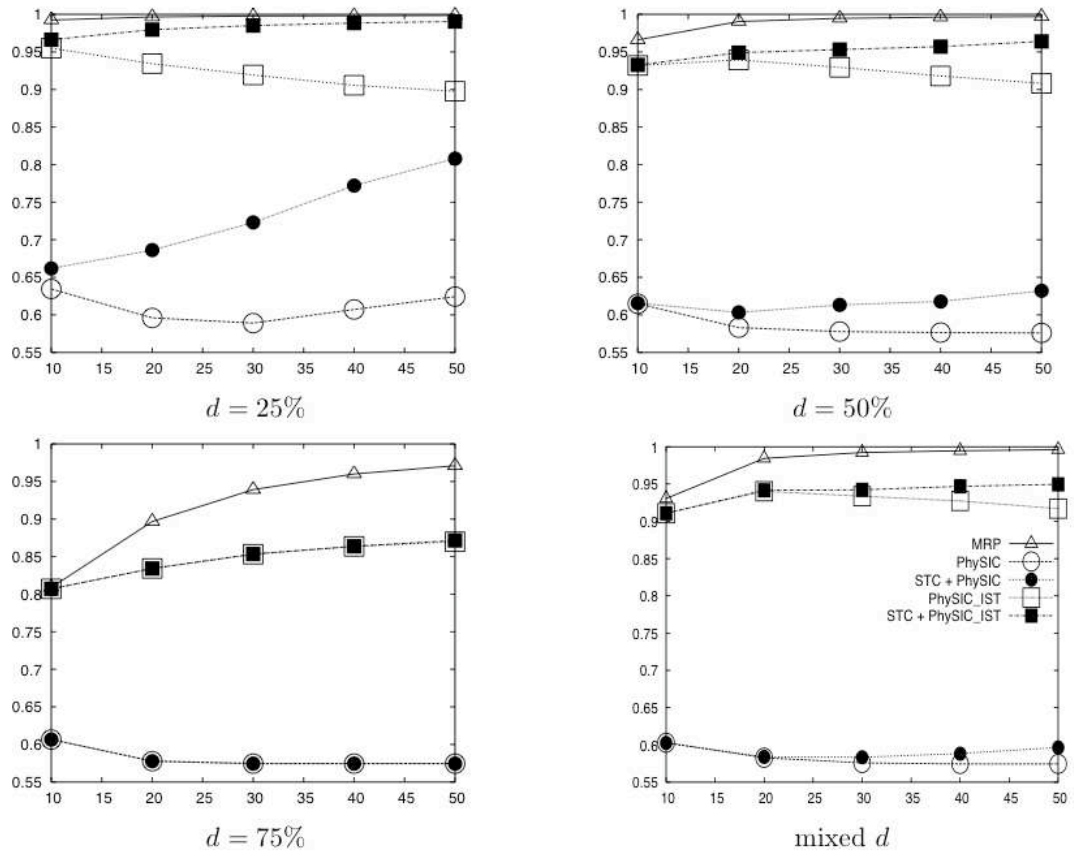

Figure B.1: Average level of supertree resolution for various supertree methods as measured by the CIC criterion, on input collections of 10 to 50 source trees depending on their overlap: each source tree lacks on average $d = 75\%$, $d = 50\%$ or $d = 25\%$ of the taxa in the correct tree to reconstruct; alternatively, condition *mixed d* represents collections containing trees taken at random from the three previous kinds of collections. A CIC value of 1 indicates a fully resolved tree, while a 0 value indicates a star tree.

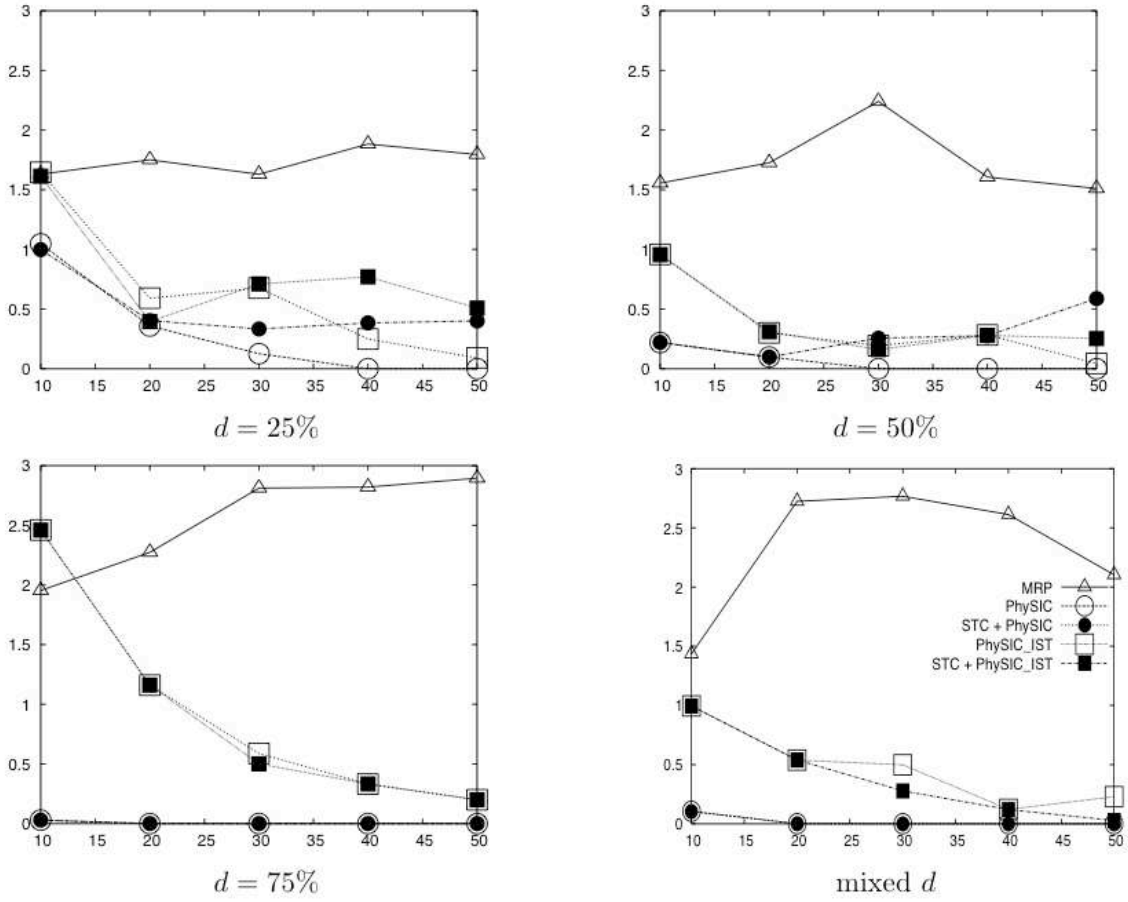

Figure B.2: Average percentage of type I error for various supertree methods, on input collections of 10 to 50 source trees depending on their overlap. Type I error is the percentage of incorrect edges contained in the inferred supertree.

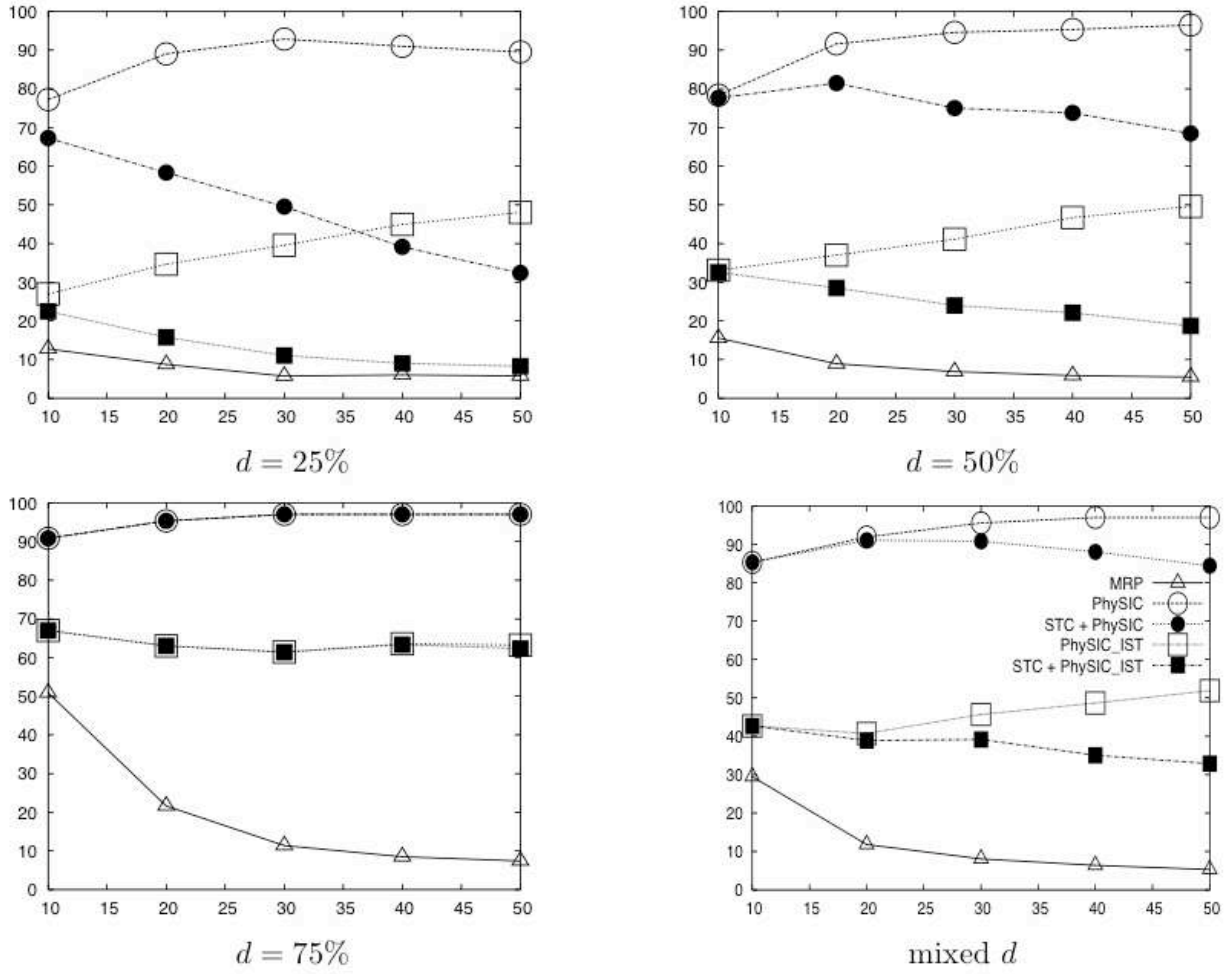

Figure B.3: Average percentage of type II error for various supertree methods, on input collections of 10 to 50 source trees depending on their overlap. Type II error is the percentage of edges in the correct tree (i.e. model tree) that are not contained in the inferred supertree.

Bibliography

[ABHL03] L. Addario-Berry, M. Hallett, and J. Lagergren. Towards identifying lateral gene transfer events. *Pac Symp Biocomput*, pages 279–290, 2003.

[ABLS04] L. Arvestad, A.C. Berglund, J. Lagergren, and B. Sennblad. Gene tree reconstruction and orthology analysis based on an integrated model for duplications and sequence evolution. In P.E. Bourne and D. Gusfield, editors, *Proc. of the 8th Ann. Int. Conf. on Comput. Molecular Biol. (RECOMB'04)*, pages 326–335. ACM, 2004.

[Ada72] E. Adams. Consensus techniques and the comparison of taxonomic trees. *Syst. Zool.*, 21:390–397, 1972.

[AGN01] J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics*, 229(1–3):3–27, 2001.

[AK94] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees. In *Proc. of the 35th Ann. Symp. on Found. of Comp. Sci. (FOCS'94)*, pages 758–769, 1994.

[AK97] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithm. *SIAM J. Comp.*, 26(6):1656–1669, 1997.

[Apr66] J.D. Apresjan. An algorithm for constructing clusters from a distance matrix. *Mashinnyi perevod: prikladnaja lingvistika*, 9:3–18, 1966.

[ASSU81] A.V. Aho, Y. Sagiv, T.G Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comp.*, 10(3):405–421, 1981.

[Bal03] S.L. Baldauf. The deep roots of eukaryotes. *Science*, 300:1703–1705, 2003.

[Bau92] B.R. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41:3–10, 1992.

[BB99] V. Berry and D. Bryant. Faster reliable phylogenetic analysis. In *3rd Ann. Int. Conf. on Research in Computational Molecular Biology (RECOMB)*, pages 59–68, 1999.

[BB01] D. Bryant and V. Berry. A structured family of clustering and tree construction methods. *Advances in Applied Mathematics*, 27(4):705–732, 2001.

[BBES08] V. Berry, O.R.P. Bininda-Emonds, and C. Semple. Multi-level supertrees. to be submitted, 2008.

[BBJ+00] V. Berry, D. Bryant, T. Jiang, P.E. Kearney, M. Li, T. Wareham, and H. Zhang. A practical algorithm for recovering the best supported edges of an evolutionary tree (extended abstract). In *Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 287–296, New York, 2000. Society for Industrial and Applied Mathematics.

[BD86] H-J Bandelt and A. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Adv. in appl. math.*, 7:309–343, 1986.

[BD92] H.-J. Bandelt and A. Dress. A canonical decomposition theory for metrics on a finite set. *Advances Math*, 92:47–105, 1992.

[BdCG+98] A. Ben-dor, B. Chor, D. Graur, Ophir R., and D. Pelleg. Constructing phylogenies from quartets: Elucidation of eutherian superordinal relationships. *Journal of Computational Biology*, 5:377–390, 1998.

[BDI+01] J.R. Brown, C.J. Douady, M.J. Italia, W.E. Marshall, and M.J. Stanhope. Universal trees based on large combined protein sequence data sets. *Nat. Genet.*, 28(3):281–285, 2001.

[BE03] O.R.P. Bininda-Emonds. Novel versus unsupported clades: assessing the qualitative support for clades in MRP supertrees. *Syst. Biol.*, 52(6):839–848, 2003.

[BE04] O. R. P. Bininda-Edmonds, editor. *Phylogenetic supertrees: Combining information to reveal the tree of life*, volume 4 of *Computational Biology series*. Kluwer Academic Publishers, 2004.

[BEB98] O.R.P. Bininda-Emonds and H.N. Bryant. Properties of matrix representation with parsimony analyses. *Syst. Biol.*, 47(3):497–508, 1998.

[BECJ+07] O.R.P. Bininda-Emonds, M. Cardillo, K.E. Jones, R.D.E. MacPhee, R.M.D. Beck, R. Grenyer, S.A. Price, R.A. Vos, J.L. Gittleman, and A. Purvis. The delayed rise of present-day mammals. *Nature*, 446(7135):507–512, 2007.

[BEGS02] O.R.P. Bininda-Emonds, J.L. Gittleman, and M.A. Steel. The (super)tree of life: procedures, problems, and prospects. *Ann. Rev. Ecol. Syst.*, 33:265–289, 2002.

[Ber97] V. Berry. *Méthodes et Algorithmes pour reconstruire les arbres de l'Évolution.* PhD thesis, Université Montpellier II, 1997.

[BES01] O.R.P. Bininda-Emonds and M.J. Sanderson. Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Syst. Biol.*, 50(4):565–579, 2001.

[BG91] J.P. Barthélemy and A. Guénoche. *Trees and proximities representations*. Wiley, 1991.

[BG97] V. Berry and O. Gascuel. Inferring evolutionary trees with strong combinatorial evidence. In T. Jiang and D.T. Lee, editors, *Computing and Combinatorics, 3rd Annual International Conference*, volume 1276 of *LNCS*, pages 111–123. Springer-Verlag, 1997.

[BGNP05] V. Berry, S. Guillemot, F. Nicolas, and C. Paul. On the approximation of computing evolutionary trees. In L. Wang, editor, *Proc. of the 11th Ann. Int. Conf. on Comp. and Combin. (COCOON'05)*, volume 3595 of *LNCS*, pages 115–125. Springer, 2005.

[BJK+99] V. Berry, T. Jiang, P.E. Kearney, M. Li, and T. Wareham. Quartet cleaning: Improved algorithms and simulations. In Springer, editor, *Proc. of the 7th Annual European Symposium (ESA)*, volume 1643, pages 313–324, 1999.

[BN03] V. Berry and F. Nicolas. Super-arbre d'accord maximum. Technical Report RR-LIRMM 03040, LIRMM, Université Montpellier 2, 2003.

[BN04] V. Berry and F. Nicolas. Maximum agreement and compatible supertrees. In S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, editors, *Proc. of the 15th Ann. Symp. on Combin. Pattern Matching (CPM'04)*, volume 3109 of *LNCS*, pages 205–219, 2004.

[BN06] V. Berry and F. Nicolas. Improved parametrized complexity of the maximum agreement subtree and maximum compatible tree problems. *IEEE/ACM Trans. on Comput. Biol. and Bioinf.*, 3(3):289–302, 2006.

[BN07] V. Berry and F. Nicolas. Maximum agreement and compatible supertrees. *Journal of Discrete Algorithms*, 5(3):564–59, 2007.

[Bou07] P. Bouchet. L'insaisissable inventaire des espèces. *La Recherche*, 28, 2007.

[BR04] B.R. Baum and M.A. Ragan. The MRP method. In O.R.P. Bininda-Emonds, editor, *Phylogenetic supertrees: combining information to reveal the Tree of Life*, pages 17–34. Kluwer, 2004.

[Bry97] D. Bryant. *Building trees, hunting for trees and comparing trees: theory and method in phylogenetic analysis*. PhD thesis, University of Canterbury, Department of Mathemathics, 1997.

[Bry02] D. Bryant. A classification of consensus methods for phylogenies. In M. Janowitz, F.-J. Lapointe, F.R. McMorris, B. Mirkin, and F.S. Roberts, editors, *Bioconsensus*, DIMACS, pages 163–184. Am. Math. Soc., 2002.

[BS95] D. Bryant and M.A. Steel. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16(4):425–453, 1995.

[BS06] V. Berry and C. Semple. Fast computation of supertrees for compatible phylogenies with nested taxa. *Syst. Biol.*, 55:270–288, 2006.

[BSE+07] E. Bapteste, Susko, Leigh E., J., I. Ruiz-Trillo, J. Bucknam, and W.F. Doolittle. Alternative methods for concatenation of core genes indicate a lack of resolution in deep nodes of the prokaryotic phylogeny. *Mol. Biol. Evol*, 25:83–91, 2007.

[Bun71] P. Buneman. *Mathematics in Archeological and Historical Sciences*, chapter The recovery of trees from measures of dissimilarity, pages 387–395. Edhinburgh University Press, 1971.

[BVM00] P. Bonizzoni, G. Della Vedova, and G. Mauri. Approximating the maximum isomorphic agreement subtree is hard. *Int. Journal of Foundations of Comp. Sci.*, 11(4):579–590, 2000.

[CBDG06] A. Criscuolo, V. Berry, E.J.P. Douzery, and O. Gascuel. SDM: a fast distance-based approach for (super)tree building in phylogenomics. *Syst. Biol.*, 55(5):740–755, 2006.

[CDE+03] D. Chen, L. Diao, O. Eulenstein, D. Fernández-Baca, and M.J. Sanderson. Flipping: A supertree construction method. In *Bioconsensus*, volume 61 of *Series in Discrete Math. and Theoretic Computer Science*, pages 135–160, Providence, 2003. DIMACS, Am. Math. Soc.

[CDvM+06] F.D. Ciccarelli, T. Doerks, C. von Mering, C.J. Creevey, B. Snel, and P. Bork. Toward automatic reconstruction of a highly resolved tree of life. *Science*, 311:1283–1287, 2006.

[CEFBS02] D. Chen, O. Eulenstein, D. Fernández-Baca, and M. Sanderson. Supertrees by flipping. In *Proc. of the 8th Ann. Int. Conf. on Comput. and Combin. (COCOON'02)*, pages 391–400, London, UK, 2002. Springer-Verlag.

[CFCH+01] R. Cole, M. Farach-Colton, R. Hariharan, T. M. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the Maximum Agreement SubTree problem for binary trees. *SIAM J. Comp.*, 30(5):1385–1404, 2001.

[CFS07] M. Casanellas and J. Fernández-Sánchez. Performance of a new invariants method on homogeneous and nonhomogeneous quartet trees. *Mol. Biol. Evol.*, 24:288–293, 2007.

[CGP05] A. Calteau, M. Gouy, and G. Perrière. Horizontal transfer of two operons coding for hydrogenases between bacteria and archaea. *J. Mol. Evol.*, 60:557–565, 2005.

[Cha98] M.A. Charleston. Jungles: a new solution to the host/parasite phylogeny reconciliation problem. *Math. Biosci.*, 149:191–223, 1998.

[CSW06] J.A. Cotton, C.S.C. Slater, and M. Wilkinson. Discriminating supported and unsupported relationships in supertrees using triplets. *Syst. Biol.*, 55(2):345–350, 2006.

[CV06] R. Cilibrasi and P.M.B. Vitányi. A new quartet tree heuristic for hierarchical clustering. In *Statistics and Optimization of Clustering Worksh. (EUPASCAL)*, pages 5–6, 2006.

[Dan04] P. Daniel. Supertree methods: some new approaches. Master's thesis, University of Canterbury, 2004.

[DB07] W.F. Doolittle and E. Bapteste. Pattern pluralism and the Tree of Life hypothesis. *Proc. Natl. Acad. Sci. U.S.A.*, 104:2043–2049, Feb 2007.

[DBAG07] E. Desmond, C. Brochier-Armanet, and S. Gribaldo. Phylogenomics of the archaeal flagellum: rare horizontal gene transfer in a unique motility structure. *BMC Evol. Biol.*, 7:106, 2007.

[Dek86] M.C. Dekker. Reconstruction methods for derivation trees. Master's thesis, University of Amsterdam, 1986.

[DF99] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[DFS99] R.G. Downey, M.R. Fellows, and U. Stege. Computational tractability: The view from mars. *Bulletin of the European Association for Theoretical Computer Science*, 69:73–97, 1999.

[DGP02] V. Daubin, M. Gouy, and G. Perrière. A phylogenomic approach to bacterial phylogeny: evidence of a core of genes sharing a common history. *Genome Res.*, 12(7):1080—1090, 2002.

[DHV05] D. Durand, B.V. Halldórsson, and B. Vernot. A hybrid micro-macroevolutionary approach to gene tree reconstruction. In S. Miyano, J.P. Mesirov, S. Kasif, S. Istrail, P.A. Pevzner, and M.S. Waterman, editors, *Proc. of the 9th Ann. Int. Conf. on Res. in Comput. Mol. Biol. (RECOMB'05)*, volume 3500 of *LNBI*, pages 250–264. Springer-Verlag, 2005.

[DM07] T. Dagan and W. Martin. Ancestral genome sizes specify the minimum rate of lateral gene transfer during prokaryote evolution. *Proc. Natl. Acad. Sci. U.S.A.*, 104:870–875, 2007.

[Dre97] A. Dress. Towards a theory of holistic clustering. In B. Mirkin, F.R. McMorris, F. Roberts, and A. Rzhetsky, editors, *Mathematical Hierarchies and Biology*, DIMACS series in Discete Math. and Theoretical Comp. Science, pages 271–290. AMS, 1997.

[DS04] P. Daniel and C. Semple. Supertree algorithms for nested taxa. In O.R.P. Bininda-Emonds, editor, *Phylogenetic supertrees: combining information to reveal the Tree of Life*, pages 151–171. Kluwer, Dordrecht, 2004.

[dVGM07] D. M. de Vienne, T. Giraud, and O. C. Martin. A congruency index for testing topological similarity between trees. *Bioinformatics*, 23(23):3119–3124, 2007.

[DVW02] G. Della Vedova and T. Wareham. Optimal algorithms for local vertex quartet cleaning. *Bioinformatics*, 18(10):1297–1304, 2002.

[ESSW97] P.L. Erdös, M. Steel, L.A. Székely, and T.J. Warnow. Constructing big trees from short sequences. In *Automata, Languages and Programming, 24th Int. Colloquium (ICALP)*, volume 1256 of *LNCS*, pages 827–837. Springer, 1997.

[FG85] C.R. Finden and A.D. Gordon. Obtaining common pruned trees. *J. of Classification*, 2:255–276, 1985.

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[FKE70] J.S. Farris, A.G. Kluge, and M.J. Eckart. A numerical approach to phylogenetic systematics. *Systematic Zool.*, 19:172–189, 1970.

[For02] P. Forterre. A hot story from comparative genomics: reverse gyrase is the only hyperthermophile-specific protein. *Trends Genet.*, 18:236–237, 2002.

[FPT95] M. Farach, T.M. Przytycka, and M. Thorup. On the agreement of many trees. *Information Processing Letters*, 55(6):297–301, 1995.

[Gam] P. Gambette. Who is who in phylogenetic networks - http://www.lirmm.fr/~gambette/PhylogeneticNetworks.

[GB07] S. Guillemot and V. Berry. Fixed-parameter tractability of the maximum agreement supertree problem. In B. Ma and K. Zhang, editors, *Proc. of the 18th Ann. Symp. on Combin. Pattern Matching (CPM'07)*, volume 4580 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 2007.

[GDL02] J.P. Gogarten, W.F. Doolittle, and J.G. Lawrence. Prokaryotic evolution in light of gene transfer. *Mol. Biol. Evol.*, 19:2226–2238, 2002.

[GG03] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, 52:696–704, 2003.

[GKKM93] W. Goddard, E. Kubicka, G. Kubicki, and F.R. McMorris. Agreement subtrees, metric and consensus for labeled binary trees. In *DIMACS Workshop on Partitioning Data Sets*. AMS, 1993.

[GN03] J. Gramm and R. Niedermeier. A fixed-parameter algorithm for minimum quartet inconsistency. *Journal of Computer and System Sciences*, 67:2003, 2003.

[GN05] S. Guillemot and F. Nicolas. Parameterized complexity of the MAST and MCT problems. Technical report, LIRMM, Univ. Montpellier 2, (extended version of the CPM'06 paper), 2005.

[GN06] S. Guillemot and F. Nicolas. Solving the maximum agreement subtree and the maximum compatible tree problems on many bounded degree trees. In M Lewenshtein and G. Valiente, editors, *Proc. of the 17th Ann. Symp. on Combin. Pattern Matching (CPM'06)*, volume 4009 of *LNCS*, pages 165–176. Springer-Verlag, 2006.

[GNT08] J. Gramm, A. Nickelsen, and T. Tantau. Fixed-parameter algorithms in phylogenetics. *Comput. J.*, 51(1):79–101, 2008.

[Gol05] P.A. Goloboff. Minority-rule supertrees? MRP, compatibility, and MinFlip may display the least frequent groups. *Cladistics*, 21:282–294, 2005.

[Gor79] A.D. Gordon. A measure of the agreement between rankings. *Biometrika*, 66:7–15, 1979.

[Gor80] A.D. Gordon. On the assessment and comparison of classifications. In R. Tomassone, editor, *Analyse de Données et Informatique*. Le Chesnay, INRIA, 1980.

[Gor86] A.D. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labelled leaves. *J. Classif.*, 3:335–348, 1986.

[Gor03] P. Gorecki. Reconciliation and horizontal gene transfer. poster, 2003.

[GP02] P.A. Goloboff and D. Pol. Semi-strict supertrees. *Cladistics*, 18(5):514–525, 2002.

[GSS07] S. Grunewald, M.A. Steel, and M.S. Swenson. Closure operations in phylogenetics. *Math. Biosci.*, 208:521–537, 2007.

[GW01] G. Ganapathy and T.J. Warnow. Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In O. Gascuel and B. M. E. Moret, editors, *Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI'01)*, pages 156–163, 2001.

[GW02] G. Ganapathy and T.J. Warnow. Approximating the complement of the maximum compatible subset of leaves of k trees. In *Proc. of the 5th Int. Workshop on Approx. Alg. for Combin. Optim. (APPROX'02)*, pages 122–134, 2002.

[GWK05] F. Ge, L.S. Wang, and J. Kim. The cobweb of life revealed by genome-scale estimates of horizontal gene transfer. *PLoS Biol.*, 3:e316, 2005.

[HB06] D.H. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Mol. Biol. Evol.*, 23:254–267, 2006.

[HJWZ96] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Disc. Appl. Math.*, 71(1–3):153–169, 1996.

[HKW99] M.R Henzinger, V. King, and T.J. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24:1–13, 1999.

[HL01] M.T. Hallett and J. Lagergren. Efficient algorithms for lateral gene transfer problems. In *Procs of the 5th Ann. int. conf. on Comput. Molec. Biol. (RECOMB'01)*, pages 149–156, New York, NY, USA, 2001. ACM.

[HL04] M. Hallett and A. Lagergren, J. anTofigh. Simultaneous identification of duplications and lateral transfers. In *Procs. of the 8th ann. int. conf. on Res. in Comp. molec. Biol. (RECOMB'04)*, pages 347–356, New York, 2004. ACM.

[HNR$^+$98] D. Huson, S. Nettles, K. Rice, T.J. Warnow, and S. Yooseph. Hybrid tree reconstruction methods. In *2nd Workshop on Algorithm Engineering (WAE'98)*, Saarbrücken, 1998.

[HNW99] D. Huson, S. Nettles, and T.J. Warnow. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Journal of Computational Biology*, 6(369-386), 1999.

[HS96] A.M. Hamel and M.A. Steel. Finding a maximum compatible tree is NP-hard for sequences and trees. *Applied Mathematics Letters*, 9(2):55–59, 1996.

[HS08] V.T. Hoang and W.-K. Sung. Fixed parameter polynomial time algorithms for maximum agreement and compatible supertrees. In S. Albers and P. Weil, editors, *25th Int. Symp. on Theoretical Aspects of Computer Science (STACS 2008)*, pages 361–372, Dagstuhl, Germany, 2008. Internationales Begegnungs- und Forschungszentrum f"ur Informatik (IBFI), Schloss Dagstuhl, Germany.

[Hu02] M. Hu. A collapsing method for the efficient recovery of optimal edges in phylogenetic trees. Master's thesis, University of Waterloo, Canada, 2002.

[Hus98] D.H. Huson. SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14:68–73, 1998.

[JKL98] T. Jiang, P. E. Kearney, and M. Li. Orchestrating quartets: Approximation and data correction. *Proc. of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 416–425, 1998.

[JNSS04] J. Jansson, J.H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted maximum agreement supertrees. In M. Farach, editor, *Proc. of the 6th Amer. Theor. Inform. Symp. (LATIN'04)*, pages 499–508, 2004.

[JNSS05] J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted maximum agreement supertrees. *Algorithmica*, 43(4):293–307, 2005.

[JvNB+08] E. Jousselin, S. van Noort, V. Berry, J.Y. Rasplus, N. Rønsted, J.C. Erasmus, and J.M. Greeff. One fig to bind them all: host conservatism in a fig wasp community unraveled by cospeciation analyses among pollinating and nonpollinating fig wasps. *Evolution*, 62(7):1777–1797, 2008.

[Kea98] P.E. Kearney. The ordinal quartet method. In *Proc. of the 2nd ann. int. conf. on Comput. molecular biol. (RECOMB'98)*, pages 125–134, New York, NY, USA, 1998. ACM.

[KH08] T.H. Kloepper and D.H. Huson. Drawing explicit phylogenetic networks and their integration into SplitsTree. *BMC Evol. Biol.*, 8:22, 2008.

[KKM92] E. Kubicka, G. Kubicki, and F.R. McMorris. An algorithm to find agreement subtrees. *Journal of Classification*, 1992.

[KLST99] M.-Y. Kao, T.W. Lam, W.-K. Sung, and H.F. Ting. A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees. In *Proc. of the 7th Annual European Symposium on Algorithms (ESA'99)*, pages 438–449, 1999.

[KLST01] M.-Y. Kao, T.W. Lam, W.-K. Sung, and H.F. Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212–233, 2001.

[LC97] F.-J. Lapointe and G. Cucumel. The average consensus procedure: Combination of weighted trees containing identical or overlapping sets of taxa. *Syst. Biol.*, 46:306–312, 1997.

[LDOM05] E. Lerat, V. Daubin, H. Ochman, and N.A. Moran. Evolutionary origins of genomic repertoires in bacteria. *PLoS. Biol.*, 3(5):e130, 2005.

[LHC+05] C.-M. Lee, L.-J. Hung, M.-S. Chang, C.-B. Shen, and C.-Y. Tang. An improved algorithm for the maximum agreement subtree problem. *Information Processing Letters*, 94:211–216, 2005.

[LPVLLG93] G. Lecointre, H. Philippe, H.L. Vân Lê, and H. Le Guyader. Species sampling has a major impact on phylogenetic inference. *Mol. Phylogenet. Evol.*, 2(3):205–224, 1993.

[LR05] F.J. Lapointe and L.J. Rissler. Congruence, consensus, and the comparative phylogeography of codistributed species in California. *Am. Nat.*, 166:290–299, 2005.

[LRRV06] M. Llabrés, J. Rocha, F. Rosselló, and G. Valiente. On the ancestral compatibility of two phylogenetic trees with nested taxa. *J. Math. Biol.*, 53(3):340–364, 2006.

[LV03] V.A. Lyubetsky and V.V. V'yugin. Methods of horizontal gene transfer determination using phylogenetic data. *In Silico Biol.*, 3:17–31, 2003.

[LWT99] J. Lyons-Weiler and K. Takahashi. Branch length heterogeneity leads to nonindependent branch length estimates and can decrease the efficiency of methods of phylogenetic inference. *Journ. Mol. Evol.*, 49:392–405, 1999.

[MS99] V. Moulton and M. Steel. Retractions of finite distance functions onto tree metrics. *Disc. Appl. Math.*, 91(1-3):215–233, 1999.

[NLSD01] C.L. Nesbo, S. L'Haridon, K.O. Stetter, and W.F. Doolittle. Phylogenetic analyses of two "archaeal" genes in thermotoga maritima reveal multiple transfers between archaea and bacteria. *Mol. Biol. Evol.*, 18:362–375, 2001.

[NR03] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003.

[NRSJ$^+$01] L. Nakhleh, U. Roshan, K. St. John, J. Sun, and T.J. Warnow. Designing fast converging phylogenetic methods. In *ISMB (Supplement of Bioinformatics)*, pages 190–198, 2001.

[NRW05] L. Nakhleh, D. Ruths, and L.-S. Wang. RIATA-HGT: a fast and accurate heuristic for reconstructing horizontal-gene transfer. In L. Wang, editor, *Computating and Combinatorics, 11th Ann. Int. Conf. (COCOON'05)*, volume 3595 of *LNCS*, pages 84–93. Springer, 2005.

[NW96] M.P. Ng and N.C. Wormald. Reconstruction of rooted trees from subtrees. *Disc. Appl. Math.*, 69(1–2):19–31, 1996.

[Pag02] R.D. Page. Modified mincut supertrees. In R. Guigó and D. Gusfield, editors, *Proc. of the 2nd Int. Worksh. on Algor. in Bioinf. (WABI'02)*, pages 537–552, 2002.

[Pag04] R.D. Page. Taxonomy, supertrees and the tree of life. In O.R.P. Bininda-Emonds, editor, *Phylogenetic Supertrees: Combining Information To Reveal The Tree Of Life*, pages 247–266. Kluwer Academic, 2004.

[Pag07] R.D. Page. TBMap: a taxonomic perspective on the phylogenetic database TreeBASE. *BMC Bioinformatics*, 8:158, 2007.

[PC98] R.D. Page and M.A. Charleston. Trees within trees: Phylogeny and historical associations. trends ecol. *Trends Ecol. Evol.*, 13:35–9, 1998.

[PSD03] W.H. Piel, M.J. Sanderson, and M.J. Donoghue. The small-world dynamics of tree networks and data mining in phyloinformatics. *Bioinformatics*, 19:1162–1168, 2003.

[PT04] Z.S. Peng and H.F. Ting. An $O(n \log n)$-time algorithm for the Maximum Constrained Agreement Subtree problem for binary trees. In *Proc. of the 15th International Symposium on Algorithms and Computations (ISAAC'04)*, pages 754–765, 2004.

[PTBE04] R. Piaggio-Talice, J.G. Burleigh, and O. Eulenstein. Quartet supertrees. In O.R.P. Bininda-Emonds, editor, *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, pages 173–191. Kluwer Academic, Dordrecht, the Netherlands., 2004.

[Pur95] A. Purvis. A composite estimate of primate phylogeny. *Philos. Trans. R. Soc. Lond. B Biol. Sci.*, 348(1326):405–421, 1995.

[PW02] D. Pisani and M. Wilkinson. Matrix representation with parsimony, taxonomic congruence, and total evidence. *Syst. Biol.*, 51(1):151–155, 2002.

[Rag92] M.A. Ragan. Matrix representation in reconstructing phylogenetic relationships among the eukaryots. *Biosystems*, 28(1–3):47–55, 1992.

[RBG+07] V. Ranwez, V. Berry, S. Guillemot, A. Criscuolo, and E.J.P. Douzery. PhySIC: a veto method with desirable properties. *Syst. Biol.*, 56(5):798–817, 2007.

[RF81] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Math. Biosci.*, 53:131–147, 1981.

[RG01] V. Ranwez and O. Gascuel. Quartet-based phylogenetic inference: Improvements and limits. *Mol. Biol. Evol.*, 18:1103–1116, 2001.

[RMWW04] U. Roshan, B.M.E. Moret, T.J. Warnow, and T.L. Williams. Rec-i-dcm3: A fast algorithmic technique for reconstructing large phylogenetic trees. In *3rd Int. IEE Computational Systems Bioinformatics Conference (CSB)*, pages 98–109, 2004.

[RR04] H.A. Ross and A.G. Rodrigo. An assessment of matrix representation with compatibility in supertree construction. In O.R.P. Bininda-Emonds, editor, *Phylogenetic supertrees (combining information to reveal the tree of life)*, volume 4. Kluwer academic publishers, 2004.

[RSWY97] K. Rice, M. Steel, T.J. Warnow, and S. Yooseph. Getting better topology estimates of difficult evolutionary trees. Technical report, U. Penn, 1997.

[San08] Michael J. Sanderson. Phylogenetic signal in the eukaryotic tree of life. *Science*, 321(5885):121–123, 2008.

[SBB+93] M.J. Sanderson, B.G. Baldwin, G. Bharathan, C.S. Campbell, D. Ferguson, J.M. Porter, C. Von Dohlen, M.F. Wojciechowski, and M.J. Donoghue. The growth of phylogenetic information and the need for a phylogenetic database. *Syst. Biol.*, 42:562–568, 1993.

[SDB00] M.A. Steel, A. Dress, and S. Böcker. Simple but fundamental limitations on supertree and consensus tree methods. *Syst. Biol.*, 49(2):363–368, 2000.

[SDPE94] M.J. Sanderson, M.J. Donoghue, W.H. Piel, and T. Eriksson. TreeBASE: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *Ann. J. Bot.*, 81:183, 1994.

[SJWMV03] K. St. John, T.J. Warnow, B.M.E. Moret, and L. Vawter. Performance study of phylogenetic methods: (unweighted) quartet methods and Neighbor-Joining. *Journal of Algorithms*, 48(1):173 – 193, 2003.

[SLM05] A. Stamatakis, T. Ludwig, and H. Meier. RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21:456–463, 2005.

[SPH98] M. J. Sanderson, A. Purvis, and C. Henze. Phylogenetic supertrees: assembling the trees of life. *Trends in Ecol. and Evol.*, 13(3):105–109, 1998.

[SS00] C. Semple and M.A. Steel. A supertree method for rooted trees. *Disc. Appl. Math.*, 105(1–3):147–158, 2000.

[SS03] C. Semple and M. A. Steel. *Phylogenetics*, volume 24 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2003.

[ST77] S. Sattah and A. Tversky. Additive similarity trees. *Psychometrika*, 42:319–345, 1977.

[Ste92] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *J. of Classification*, 9:91–116, 1992.

[Ste99] U. Stege. Gene trees and species trees: The gene-duplication problem in fixed-parameter tractable. In K. H Frank, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *6th Int. Worksh. on Algorithms and Data Structures (WADS'99)*, volume 1663 of *LNCS*, pages 288–293. Springer, 1999.

[SvH96] K. Strimmer and A. von Haeseler. Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13(7):964–969, 1996.

[SW93] M.A. Steel and T.J. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.

[SWR08] S. Snir, T.J. Warnow, and S. Rao. Short quartet puzzling: A new quartet-based phylogeny reconstruction algorithm. *Journal of Computational Biology*, 15(1):91–103, 2008.

[TN84] Tajima and M. Nei. Estimation of evolution distance between nucleotide sequences. *Mol. Biol. Evol.*, 1(3):269–285, 1984.

[TW03] J.L. Thorley and M. Wilkinson. A view of supertrees methods. In M.F. Janowitz, F.-J. Lapointe, F.R. McMorris, and F.S. Roberts, editors, *Bioconsensus*, volume 61 of *Discrete Mathematics and Theoretical Computer Science*, pages 185–194. DIMACS, 2003.

[TWC98] J. Thorley, M. Wilkinson, and M. Charleston. The information content of consensus trees. In M. Rizzi, A. andVichi and H.H. Bock, editors, *Advances in Data Science and Classification*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 91–98. Springer, Berlin, 1998.

[VSGD07] B. Vernot, M. Stolzer, A. Goldman, and D. Durand. Reconciliation with non-binary species trees. *Comput Syst Bioinformatics Conf*, 6:441–452, 2007.

[Wil99a] S.J. Willson. Building phylogenetic trees from quartets by using local inconsistency measures. *Mol. Biol. Evol.*, 16(5):685–693, 1999.

[Wil99b] S.J. Willson. A higher order parsimony method to reduce long-branch attraction. *Mol. Biol. Evol.*, 16(5):694–705, MAY 1999.

[Wil01] S.J. Willson. An error-correcting map for quartets can improve the signals for phylogenetic trees. *Mol. Biol. Evol.*, 18(3):344–351, 2001.

[WPCC05] M. Wilkinson, D. Pisani, J.A. Cotton, and I. Corfe. Measuring support and finding unsupported relationships in supertrees. *Syst. Biol.*, 54(5):823–831, 2005.

[WRG$^+$01] Y.I. Wolf, I.B. Rogozin, N.V. Grishin, R.L. Tatusov, and E.V. Koonin. Genome trees constructed using five different approaches suggest new major bacterial clades. *BMC Evol. Biol.*, 1:8, 2001.

[WTLB00] M. Wilkinson, J.L. Thorley, D.T.J. Littlewood, and R.A. Bray. Towards a phylogenetic supertree of platyhelminthes? In D. T. J. Littlewood and R. A. Bray, editors, *Interrelationships of the Platyhelminthes*, number 60 in Systematics Association Special Volumes, chapter 27. CRC Press, London, 2000.

[WTP$^+$04] M. Wilkinson, J. Thorley, D. Pisani, F.-J. Lapointe, and O. McInerney. Some desiderata for liberal supertree. In O.R.P. Bininda-Emonds, editor, *Phylogenetic supertrees (combining information to reveal the tree of life)*, volume 4, pages 227–246. Kluwer academic publishers, 2004.

Part VII

A selection of papers to which I contributed

The following pages are inserts of a small selection of papers to which I contributed, one for each part of the manuscript.

1. Bryant and Berry, Advances in Applied Mathematics, 2001.
2. Berry and Nicolas, Journal of Discrete Algorithms, 2007.
3. Ranwez, Berry, Criscuolo, Guillemot, Scornavacca and Douzery, Systematic Biology, 2007.

A Structured Family of Clustering and Tree Construction Methods

David Bryant

LIRMM, Montpellier, France and *Departments of Mathematics and Computer Science, McGill University, Canada*
E-mail: bryant@math.mcgill.ca

and

Vincent Berry

LIRMM, Montpellier, France
E-mail: vberry@lirmm.fr

A cluster A is an Apresjan cluster if every pair of objects within A is more similar than either is to any object outside A. The criterion is intuitive, compelling, but often too restrictive for applications in classification. We therefore explore extensions of Apresjan clustering to a family of related hierarchical clustering methods. The extensions are shown to be closely connected with the well-known single and average linkage tree constructions. A dual family of methods for classification by splits is also presented. Splits are partitions of the set of objects into two disjoint blocks and are widely used in domains such as phylogenetics. Both the cluster and split methods give rise to progressively refined tree representations. We exploit dualities and connections between the various methods, giving polynomial time construction algorithms for most of the constructions and NP-hardness results for the rest.

Key Words: clustering, splits, tree construction, algorithms, classification, compatibility, quartets, hierarchies, single linkage tree, average linkage tree.

1. INTRODUCTION

A cluster is a subset of a collection of objects. A split is a separation of the collection into two disjoint parts. We will be investigating methods for classification using clusters and methods usings splits, repeatedly exploiting connections between the two approaches. We start with two classification methods that are now almost classical: Apresjan clusters[1] and Buneman splits[10]. We will show how these closely related constructions can be

extended and refined to give a highly structured family of clustering and "splitting" classification methods.

The aim of almost every clustering methods is to cluster objects together that are more similar to one another than they are to objects outside the cluster. We represent the level of similarity between elements of a set X using a **similarity function** $s : X \times X \to \Re$ which we define to be any symmetric function from pairs of objects to $\Re$. The intuitive notion of clustering is captured in the definition of Apresjan clusters[1], which is the first clustering method we investigate. Following Bandelt and Dress [2], we define Apresjan clusters in terms of similarity functions, rather than dissimilarities, to emphasize the mathematical duality between the clustering and the splitting constructions. All of the clustering methods we describe here can be modified to handle dissimilarity data.

Define the **Apresjan clusters** of a similarity function s to be $\{A : A \subseteq X,\ \iota_s(A) > 0\}$, where $\iota_s(A)$ is the strong isolation index

$$\iota_s(A) := \min_{a,a',x} \{s(a,a') - s(a,x) :\ a, a' \in A,\ x \in X - A\}. \tag{1}$$

The construction was first studied by Parker-Rhodes and Needham [22] who called the clusters K-clumps. Apresjan clusters are also known as L-groups, K-groups, and strong clusters [2, 14, 18].

The Apresjan clusters of a similarity function form a **(strong) hierarchy**, in the sense that if two Apresjan clusters intersect then one contains the other. If we have that $s(a,a) > s(a,b)$ for all $a \neq b$, as is usually the case, then all singleton clusters are Apresjan and we obtain a proper hierarchy. The clusters can then be represented using a rooted tree T with leaves labelled by members of X. Every node v in T is associated with the cluster $C(v)$, the set of all leaves that are descendents of v. For each node v we assign the length $\iota_s(C(v))$ to the edge connecting v and its parent node.

If $s(a,a)$ is constant for all $a \in X$ then the rooted tree will be a dendrogram, with all leaves the same distance from the root. Furthermore, if s satisfies the (sign reversed) ultrametic inequality

$$s(a,b) \geq \min\{s(a,c), s(b,c)\}$$

for all a, b, c then the tree T will be the dendrogram corresponding to s [2].

Buneman introduced a related classifcation method in a paper on the filation history of ancient manuscripts [10]. The input for Buneman's method is a **dissimilarity function** $\delta : X \times X \to \Re^+$, and the basic grouping unit is a **split** (bipartition $A|B$ of X) rather than a cluster. The **Buneman**

splits of δ are $\{A|B : \mu_\delta(A|B) > 0\}$, where μ_δ is the **strong separation index**

$$\mu_\delta(A|B) = \tfrac{1}{2}\min\{(\delta(a,b)+\delta(a',b'))-(\delta(a,a')+\delta(b,b')) : a,a' \in A,\ b,b' \in B\}. \tag{2}$$

An X**-tree** is an *unrooted* tree $T = (V, E)$ with labelling $L : X \to V$ such that every node of degree less than three is labelled. The Buneman splits can be represented using a valued X-tree. If we remove any edge e in an X-tree we divide the tree into two components and induce a split $S(e)$ of X given by the label sets of the two components. Buneman showed that there is an X-tree T such that the splits $\{S(e) : e \in E\}$ are exactly the Buneman splits for δ. We say that T is the **Buneman tree** for δ. We can construct a valued X-tree by assigning the length $\mu_s(A|B)$ to the edge corresponding to $A|B$. In this way we obtain a mapping from dissimilarity functions to valued X-trees, which can be computed in polynomial time with respect to $|X|$. The separation indices $\mu_\delta(A|B)$ are continuous, so a small change in δ will induce only a small change in the tree. One can show that if δ is additive, that is

$$\delta(a,b) + \delta(c,d) \leq \max\{\delta(a,c) + \delta(b,d), \delta(a,d) + \delta(b,c)\},$$

for all a, b, c, d, then the tree T will be the valued X-tree corresponding to δ.

The methods of Apresjan and Buneman are quite conservative, only returning groupings with a lot of support in the data. With Apresjan clusters, the condition that $\iota_s(A) > 0$ is very strong: for *every* choice of $a, a' \in A$ and $x \in X - A$ we must have $s(a,a') > s(a,x)$. A single "errant" similarity would exclude the cluster from the collection. Similarly, the condition for Buneman splits that $\mu_\delta(A|B) > 0$ is also very strong, with a small pertubation in the input dissimilarity potentially excluding all splits from the Buneman tree. Consequently, when the similarity or dissimilarity data is not sufficiently "tree-like," the methods produce poorly resolved trees with high degree internal nodes and only a few small clusters or splits.

Both methods are continuous, in the sense that a small change in the input will induce only a small change in the final tree. This, as well as the ability to construct the corresponding trees in polynomial time and invariance on tree-like dissimilarities (i.e., respectively ultrametric and additive), make the two methods attractive for numerous applications such as phylogenetics. The uninformative output tree is still a shortcoming. The lack of resolution in the Buneman tree motivated Moulton and Steel [20] to investigate modifications of the Buneman tree construction. They defined a new separation index μ^r_δ such that $\mu^r_\delta(A|B) \geq \mu_\delta(A|B)$ for all splits $A|B$ and yet the collection of splits $\{A|B : \mu^r_\delta(A|B) > 0\}$ could still be represented by

an X-tree. The new separation index leads to a tree construction method that is continuous, polynomial time, invariant on additive dissimilarities, and gives trees that contain all of the Buneman splits and often additional splits. The tree produced is called the **refined Buneman tree**. A tree T' **refines** a tree T if every split induced by an edge of T is also induced by some edge of T'.

In this paper we extend the work of Moulton and Steel [20] in several directions. Our goal is to derive clustering and tree construction methods that are continuous, polynomial time, and informative. To this end we present a family of methods containing and extending the clustering method of Apresjan and the tree construction method of Buneman. We explore and exploit connections between the methods, formalise the connection between Buneman splits and Apresjan clusters, and extend this duality to the new constructions. We prove that Apresjan clusters are contained in the single linkage tree (Theorem 2.3), and that our extensions to Apresjan clusters are contained in the average linkage tree (Theorem 2.4). We use NP-hardness proofs to indicate the limit of this series of refinements and extensions.

We also extend the Apresjan and Buneman constructions so that they can accept different kinds of input. One can rewrite the definition of the strong isolation index (eq. (1)) as

$$\iota_s(A) = \min\{w(aa'|x) : a, a' \in A, x \in X - A\}. \tag{3}$$

where $w(aa'|x) = s(a, a') - \max\{s(a, x), s(a', x)\}$. We can now replace w with another function. Different weighting functions w give different clustering methods. We explore the conditions that the weighting function w has to satisfy for the resulting clusters to form a hierarchy, and introduce equivalent extensions for splits and unrooted tree constructions. The ability to use more general weighting functions greatly increases the flexibility and utility of these methods.

The algorithms have been implemented in C++. Source code, and pseudo code, will be made available by the authors.

2. A FAMILY OF CLUSTERING METHODS

2.1. Definitions

A **rooted X-tree** is a rooted tree $T = (V, E)$, with nodes V and edges E, together with a labelling function $L : X \to V$ such that every node with less than two children is labelled. Put $n = |X|$. Given two nodes u and v we write $u \prec_T v$ if u is a strict descendent of v in T. The **cluster associated**

to v is the set of labels on nodes that are descendents of v (including v) and is denoted $C(v)$. The **clusters of** T are $clus(T) = \{C(v) : v \in V\}$. A cluster A is **trivial** if $|A| = 1$ or $A = X$. We use $\text{lca}(x, y)$ to denote the node that is the least common ancestor of the nodes labelled by x and y.

A set of clusters $\mathcal{C}$ is **compatible** if $\mathcal{C} = clus(T)$ for some rooted X-tree T, which holds if and only if $\mathcal{C}$ forms a hierarchy. It is a classical result that $\mathcal{C}$ is compatible if and only if for each pair $A, B \in \mathcal{C}$ one of $A - B$, $B - A$, $A \cap B$ is empty (e.g. [10]).

A **rooted triple** $ab|c$ denotes a grouping of a and b relative to c. The set of all rooted triples of X is denoted $\mathcal{R}(X)$. The set of rooted triples of a cluster $A \subseteq X$ is defined as

$$r(A) = \{uv|z : u, v \in A,\ z \in X - A\}.$$

We say that $ab|c$ is a rooted triple of T if there is a cluster $A \in clus(T)$ such that $ab|c \in r(A)$. The set of rooted triples of T is denoted $r(T)$.

A **similarity function** is a symmetric function from $X \times X$ to $\Re$. An **isolation weighting** is a function $w : \mathcal{R}(X) \longrightarrow \Re$ such that $w(ab|c) + w(ac|b) \leq 0$ for all $a, b, c \in X$. The weight $w(ab|c)$ indicates the degree to which c is excluded the group $\{a, b\}$. Given a similarity function s define

$$\rho_s(ab|c) = s(a, b) - \max\{s(a, c), s(b, c)\}. \tag{4}$$

Then ρ_s is an isolation weighting.

Given a non-empty set R of rooted triples, and an isolation weighting w let $\text{av}(R)$ denote the average weight of the triples in R and let $\text{avmin}(R, k)$ denote the average weight of the $k \leq |R|$ triples with minimum weight in R. Formally,

$$\text{av}(R) = \frac{1}{|R|} \sum_{xy|z \in R} w(xy|z) \tag{5}$$

$$\text{avmin}(R, k) = \min_{R'}\{\text{av}(R') : R' \subseteq R,\ |R'| = k\}. \tag{6}$$

Given disjoint non-empty U, V, Z define

$$\overline{w}(UV|Z) = \text{av}\{uv|z : u \in U, v \in V, z \in Z\}, \tag{7}$$

the average weight over all triples $uv|z$ with $u \in U$, $v \in V$ and $z \in Z$.

2.2. A series of clustering methods

We investigate a family of hierarchical clustering methods. In each case, the clusters are selected according to an explicit criterion. The criteria used by the three methods are closely related, and are all based on different definitions of an *isolation index* for selecting clusters of X. Here, and

throughout the paper, we use $U \uplus V$ to denote the union of disjoint sets U and V.

Definition of Clustering Indices *Let w be an isolation weighting and A a cluster of X.*

(i) The **strong isolation index** *of A is defined*

$$\iota_w(A) = \min_{uv|z}\{w(uv|z) : uv|z \in r(A)\} = avmin(r(A), 1)$$

and the **strong clusters** *of w are $\{A : \iota_w(A) > 0\}$.*

(ii) The **clean cluster index** *of A is*

$$\iota^c_w(A) = avmin(r(A), |A| - 1)$$

and the **clean clusters** *of w are $\{A : \iota^c_w(A) > 0\}$.*

(iii) The **stability index** *of A is*

$$\iota^s_w(A) = \min_{U,V,Z \neq \emptyset}\{\overline{w}(UV|Z) : A = U \uplus V, Z \subseteq X - A\}$$

and the **stable clusters** *of w are $\{A : \iota^s_w(A) > 0\}$.*

Define the strong clusters (Apresjan clusters) of a similarity function s to be the strong clusters of $w = \rho_s$. Similarly for clean clusters and stable clusters of a similarity function.

The name "clean clusters" stems from a connection with the quartet cleaning construction of [6] (see section 3.5). The indices, and the construction complexities, for all of the clustering methods are summarised in table 1. The inclusion relations between the various clustering indices and methods are presented in Theorem 2.1. The following lemma is used in both compatibility and inclusion proofs.

LEMMA 2.1. *If U, V, Z are nonempty subsets of X such that $A = U \uplus V$ and $Z \subseteq X - A$ then $|U||V||Z| \geq |A| - 1$.*

Proof. Put $p = |V| = |A - U|$. Then

$$\begin{aligned} |U||V||Z| &= (|A| - p)p|Z| && (8)\\ &\geq (|A| - p)p, && (9)\end{aligned}$$

since $|Z| \geq 1$. By straightforward calculus we have $(|A| - p)p \geq (|A| - 1)$, giving the result. ∎

We now show that the clustering methods form a series in the sense of inclusion relations for the resulting sets of clusters.

THEOREM 2.1. *For any cluster A we have $\iota_w(A) \leq \iota^c_w(A) \leq \iota^s_w(A)$. Hence every strong cluster of w is a clean cluster of w and every clean cluster is a stable cluster.*

Proof. First note that if $k \leq k'$ then $\text{avmin}(r(A), k) \leq \text{avmin}(r(A), k')$. We therefore have

$$\iota_w(A) = \text{avmin}(r(A), 1) \leq \text{avmin}(r(A), |A| - 1) = \iota^c_w(A).$$

Choose nonempty subsets U, V, and Z such that $A = U \uplus V$, $Z \subseteq X - A$ and $\overline{w}(UV|Z) = \iota^s_w(A)$. By Lemma 2.1, $|U||V||Z| \geq |A| - 1$ and so

$$\begin{aligned} \iota^c_w(A) &= \text{avmin}(r(A), |A| - 1) \\ &\leq \text{avmin}(r(A), |U||V||Z|) \\ &\leq \overline{w}(UV|Z) \\ &= \iota^s_w(A). \end{aligned}$$

∎

Having demonstrated the inclusion relationships between the strong clusters, clean clusters, and stable clusters, we now establish compatibility.

THEOREM 2.2. *Let w be an isolation weighting. The set of strong clusters of w, the set of clean clusters of w and the set of stable clusters of w are all compatible collections of clusters.*

Proof. We first prove that the set of stable clusters is compatible. Let A be a stable cluster of w and let B be a cluster incompatible with A. Put $U = A - B$, $V = A \cap B$ and $Z = B - A$. Since A and B are incompatible, U, V, and Z are all non-empty. Then $\iota^s_w(A) \leq \overline{w}(UV|Z)$, $\iota^s_w(B) \leq \overline{w}(VZ|U)$, and

$$\begin{aligned} \iota^s_w(A) + \iota^s_w(B) &\leq \overline{w}(UV|Z) + \overline{w}(VZ|U) \\ &= \tfrac{1}{|U||V||Z|} \sum_{u \in U} \sum_{v \in V} \sum_{z \in Z} w(uv|z) + w(vz|u) \\ &\leq 0 \end{aligned}$$

since w is an isolation weighting. The cluster A is stable, so $\iota_w(A) > 0$ and $\iota_w(B) < 0$. Hence B is not a stable cluster and the collection of stable clusters is compatible.

The inclusion of the set of strong clusters and the set of clean clusters in the set of stable clusters (Theorem 2.1) proves compatibility for the other constructions. ∎

The relationship between the three clustering methods can be viewed as a progressive relaxation of the selection criterion. The criteria differ over the conditions on the subsets $R \subseteq r(A)$ that can have zero or negative average weight and the cluster A still be selected: the condition of the strong cluster method is the most strict; the condition of the stable cluster method the most lenient.

The strong cluster construction is the most conservative. The condition that $\iota_w(A) > 0$ implies that no subset $R \subset r(A)$ can have negative average weight.

At the other extreme, a cluster A can be stable and still have a subset $R \subseteq r(A)$ of size $O(n^3)$ with zero or negative average weight. Such subsets R must be of a particular composition related to the conflicting triples between two incompatible clusters. If A and B are incompatible clusters then $r(A)$ and $r(B)$ conflict on the resolution of several triples, called **conflicting triples**. These are exactly the rooted triples with one element in $A - B$, one element in $A \cap B$, and one element in $B - A$. The stable cluster selection criterion prohibits exactly those negative weight subsets $R \subseteq r(A)$ that correspond to the set of conflicting triples between A and some incompatible cluster B. This guarantees that no two stable clusters will be incompatible.

The clean cluster method falls in between the two other methods: every strong cluster is a clean cluster and every clean cluster is a stable cluster. The clean clustering method bounds the size of subsets $R \subset r(A)$ with negative average weight. The bound $(|A| - 1)$ is equal to the smallest possible set of conflicting triples between A and any other cluster B. We can not increase this bound without losing the connection with stable clusters and, additionally, the guarantee of compatibility. As we shall see, the bound enables the efficient construction of clean clusters (section 2.4) whereas the construction of stable clusters is an NP-hard problem (section 2.5).

2.3. Properties and construction of strong and Apresjan clusters

The single linkage tree is one of the oldest hierarchical clustering methods in classification. The method starts with a collection of singleton clusters $\mathcal{A}_1 = \{\{x\} : x \in X\}$. At each iteration $k > 1$ the algorithm chooses a pair of maximal (by inclusion) clusters in $\mathcal{A}_{k-1}$ for which $\max_{a,b}\{s(a,b) : a \in A, b \in B\}$ is largest, and puts $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{A \cup B\}$. Ties are broken randomly. The procedure finishes when $\mathcal{A}_k$ contains X.

Here we show that Apresjan clusters are always contained in any single linkage tree for s.

THEOREM 2.3. *Let s be a similarity function on X. Then every Apresjan cluster of s is a cluster in every single linkage tree for s.*

Proof. We use a characterization of single linkage trees presented in [4] and used in [12] to solve a related problem. For each $k \in \Re$ we define the threshold graph $G[k]$ with vertex set X and edge set $E[k] = \{\{a, b\} : s(a, b) \geq k\}$.

Let A be an Apresjan cluster of s and suppose that k is the maximum value such that the subgraph induced by A is connected in $G[k]$. This is well defined since $G[-\infty]$ is complete. We will show that no other element of X is in the same component as A. Given any a in A and $x \in X - A$ if $\{a, x\} \in E[k]$ then $s(a, x) \geq k$. Since A is a strong cluster, there is $\epsilon > 0$ such that $s(a, a') \geq \epsilon + s(a, x)$ for all $a' \in A$. Then A is connected in $G[k + \epsilon]$, contradicting the maximality assumption for k. We conclude then that A is a component of $G[k]$. From [4], every component of $G[k]$ is a cluster in every single linkage tree for s. ∎

The link leads to an efficient way to compute the Apresjan clusters:

COROLLARY 2.1. *We can construct the Apresjan clusters of a similarity function s in $O(n^2)$ time, where $n = |X|$.*

Proof. One can construct a single linkage tree T for s in $O(n^2)$ time [15, 23]. By Theorem 2.3, $clus(T)$ contains all the Apresjan clusters, and possibly additional clusters. For each node v and element $x \in C(v)$ we compute

$$\begin{aligned} m[v, x] &= \min_{x'}\{s(x, x') : x' \in C(v)\} \\ M[v, x] &= \max_{x''}\{s(x, x'') : x'' \notin C(v)\}. \end{aligned}$$

Let u be the (unique) child of v such that $x \in C(u)$. If $x' \in C(v)$ then either $x' \in C(u)$ or $\text{lca}(x, x') = v$. Hence

$$\begin{aligned} m[v, x] &= \min\left\{\min_{x'}\{s(x, x') : x' \in C(u)\}\ ,\ \min_{x'}\{s(x, x') : \text{lca}(x, x') = v\}\right\} \\ &= \min\left\{m[u, x],\ \min_{x'}\{s(x, x') : \text{lca}(x, x') = v\}\right\}. \end{aligned} \tag{10}$$

The lca relationships can be precomputed in $O(n^2)$ time. A post-order traversal of T using the above conversion then gives the values $m[v, x]$

in $O(n^2)$ total time. Note that a post-order traversal of T processes the children of a node *before* processing the node.

The computation of the values $M[v,x]$ proceeds from the root to the leaves. Let v be a node, x an element of $C(v)$, and u the unique child of v for which $x \in C(u)$. If $x'' \notin C(u)$ then either $x'' \in C(u)$ or $\mathrm{lca}(x,x'') = v$. Hence

$$\begin{aligned} M[u,x] &= \max\Big\{\max_{x''}\{s(x,x'') : x'' \notin C(v)\}\ ,\ \max_{x''}\{s(x,x'') : \mathrm{lca}(x,x'') = v\}\Big\} \\ &= \max\Big\{M[v,x],\ \max_{x''}\{s(x,x'') : \mathrm{lca}(x,x'') = v\}\Big\}. \end{aligned} \tag{11}$$

We can therefore use a pre-order traversal of T to compute all of the $M[v,x]$ values in $O(n^2)$ total time. Note that a pre-order traversal processes the children of a node *after* processing the node.

The values $M[u,x]$ and $m[u,x]$ enable the computation of the isolation index of the clusters of T. Let u be a node of T and let $U = C(u)$. Then

$$\begin{aligned} \iota_s(U) &= \min_{x\in U}\Big\{\min_{x'}\{s(x,x') : x' \in U\} - \max_{x''}\{s(x,x'') : x'' \notin U\}\Big\} \\ &= \min_{x\in U}\Big\{m[u,x] - M[u,x]\Big\}. \end{aligned}$$

The Apresjan clusters are then exactly those clusters of T with positive isolation index. ∎

The strong cluster construction generalises the method of Apresjan by allowing an arbitrary isolation weighting w. We show that this more general construction can be performed using Apresjan clustering method.

LEMMA 2.2. *Let w be an isolation weighting. For each $a, b \in X$ put*

$$s(a,b) = \Big|\{ab|y : w(ab|y) > 0\}\Big| \tag{12}$$

Then every strong clusters of w is an Apresjan cluster of s.

Proof. Put $R = \{ab|c : w(ab|c) > 0\}$. Let A be a strong cluster for w. Fix $a, a' \in A$ and $x \in X - A$. For all $y \in X - A$ we have $w(aa'|y) \geq \iota_w(A) > 0$ so

$$s(a,a') = |\{y : aa'|y \in R\}| \geq |X - A|.$$

For all $a'' \in A$, $w(aa''|x) \geq \iota_w(A) > 0$ so $w(ax|a'') < 0$ and $ax|a'' \notin R$. Since also $w(ax|x) \leq 0$ we have that $ax|z \in R$ implies $z \notin A \cup \{x\}$. Thus

$$s(a,x) = |\{z : ax|z \in R\}| \leq |X - A - \{x\}|.$$

We therefore have $s(a,x) \leq |X-A|-1 < s(a,a')$. By the same argument, $s(a',x) < s(a,a')$.

It follows that $s(a,a') > s(a,x)$ for all $a,a' \in A$ and $x \in X-A$. Hence A is an Apresjan cluster of s. ▮

The converse of Lemma 2.2 is not true. For example, if $X = \{a,b,c,d\}$, $w(ab|d) = w(ac|d) = w(bc|a) = 1$ and all other triples have weight -1 then $\{a,b,c\}$ is a strong cluster of s but not of w.

COROLLARY 2.2. *The strong clusters of an isolation weighting can be constructed in $O(n^3)$ time, where $n = |X|$.*

Proof. First construct $s(a,b) = |\{ab|y : w(ab|y) > 0\}|$ for all a,b, taking $O(n^3)$ time, and compute the collection $\mathcal{C}$ of Apresjan clusters of s ($O(n^2)$ time). The isolation indices $\iota_w(A)$ for the clusters in $\mathcal{C}$ can be computed in $O(n^3)$ time using a modification of the quartet path algorithms of [7]. ▮

2.4. Properties and construction of clean clusters

We first examine the special case when $w = \rho_s$ for some similarity function s on X (eq. (4)).

By Theorem 2.1, every Apresjan cluster of s is a clean cluster of s. The converse is not true. Consider the similarity function s on $X = \{a,b,c,d\}$ given in figure 1.

INSERT FIGURE 1 NEAR HERE

There are two nontrivial clean clusters, $\{a,b\}$ and $\{a,b,c\}$, but only $\{a,b\}$ is an Apresjan cluster. Furthermore, the unique single linkage tree for s contains non-trivial clusters $\{a,b\}$ and $\{a,b,d\}$, so the clean clusters of s are not necessarily present in a single linkage tree for s. However, as we now show, the clean cluster construction is closely connected with another classical clustering method: the average linkage tree [24] (also known as the group average linkage tree [14]).

THEOREM 2.4. *Every stable cluster of s is a cluster in every average linkage tree for s. Hence every clean cluster of s is a cluster in every average linkage tree for s.*

Proof. For any two disjoint subsets A,B define $\overline{s}(A,B) = \text{av}\{s(a,b) : a \in A,\ b \in B\}$. The average linkage tree algorithm begins with a collection of singleton clusters $\mathcal{A}_1 = \{\{x\} : x \in X\}$. At each iteration $k > 1$ it chooses the pair of maximal (by inclusion) clusters $A,B \in \mathcal{A}_{k-1}$ for which

$\overline{s}(A,B)$ is largest, and puts $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{A \cup B\}$. Ties can be broken randomly. The procedure terminates when $\mathcal{A}_k$ contains X.

Let $\mathcal{A}$ be the clusters in the average linkage tree for s and let A be a stable cluster that is not a cluster in $\mathcal{A}$. Since $\mathcal{A}$ is a maximal hierarchy, $\mathcal{A} \cup \{A\}$ is incompatible. Let k be the first iteration of the average linkage tree algorithm for which $\mathcal{A}_k$ contains a cluster B that is incompatible with A. There are maximal clusters V and Z in $\mathcal{A}_{k-1}$ such that V and Z are amalgamated to give B. Hence $B = V \uplus Z$ and $\overline{s}(V, A_i) \leq \overline{s}(V,Z)$ for all maximal $A_i \in \mathcal{A}_{k-1}$.

Every cluster in $\mathcal{A}_{k-1}$ is compatible with A, so every maximal cluster in $\mathcal{A}_{k-1}$ is either contained in A or disjoint with A. W.l.o.g. suppose that $V \subset A$ and $Z \cap A = \emptyset$. Let $U_1, U_2, \ldots, U_p$ be the maximal clusters of $\mathcal{A}_{k-1} - \{V\}$ that are also contained in A. Thus $\overline{s}(U_l, V) \leq \overline{s}(V,Z)$ for all $l = 1, 2, \ldots p$. Let $U = A - V$, so that $U = \cup_{l=1}^{p} U_l$ and

$$\begin{aligned} |U|\overline{s}(U,V) &= \sum_{l=1}^{p} |U_i|\overline{s}(U_l, V) \\ &\leq \sum_{l=1}^{p} |U_i|\overline{s}(V,Z) \\ &= |U|\overline{s}(V,Z), \end{aligned}$$

giving $\overline{s}(U,V) \leq \overline{s}(V,Z)$.

For all $u \in U$, $v \in V$, $z \in Z$ we have

$$\rho_s(uv|z) = s(u,v) - \max\{s(u,z), s(v,z)\} \leq s(u,v) - s(v,z).$$

Let $w = \rho_s$. Then

$$\begin{aligned} \iota_w^s(A) &\leq \overline{w}(UV|Z) \\ &= \tfrac{1}{|U||V||W|} \sum_{u \in U} \sum_{v \in V} \sum_{z \in Z} \rho_s(uv|z) \\ &\leq \tfrac{1}{|U||V||W|} \sum_{u \in U} \sum_{v \in V} \sum_{z \in Z} s(u,v) - s(v,z) \\ &= \overline{s}(U,V) - \overline{s}(V,Z) \\ &\leq 0, \end{aligned}$$

a contradiction. Hence A is contained in the average linkage tree.

By Theorem 2.1 all clean clusters are stable clusters and so are contained in every average linkage tree. ∎

Apart from providing a characterisation of a subset of clusters in the average linkage tree, Theorem 2.4 leads to an $O(n^3)$ time algorithm for

constructing clean clusters of a similarity function (c.f. algorithms 1 and 2.

COROLLARY 2.3. *The clean clusters of a similarity function s can be constructed in $O(n^3)$ time, where $n = |X|$.*

Proof. First compute an average linkage tree T for s using the $O(n^2)$ time algorithm of [21]. By Theorem 2.4, every clean cluster of s is a cluster of T.

For each pair of nodes u, v in T such that $u \prec_T v$, define

$$R[u,v] = \{xy|z : \mathrm{lca}(x,y) = u,\ \mathrm{lca}(x,z) = \mathrm{lca}(y,z) = v\} \tag{13}$$

and

$$R_{\preceq}[u,v] = \bigcup_{u' \preceq_T u} R[u',v] \tag{14}$$

Every rooted triple in $r(T)$ is in exactly one set $R[u,v]$, and these sets can be constructed in $O(n^3)$ time.

Let $R[u,v,n]$ denote the set of n minimum weight rooted triples of $R[u,v]$. It takes $O(|R[u,v]|)$ time to extract $R[u,v,n]$ from $R[u,v]$ using the linear time selection algorithm of [8]. It therefore takes $O(n^3)$ time to extract all of the sets $R[u,v,n]$.

Let $R_{\preceq}[u,v,n]$ denote the minimum n triples of $R_{\preceq}[u,v]$. We construct these sets using dynamic programming. Let $u_1, u_2, \ldots, u_m$ be the children of u. Then $R_{\preceq}[u,v,n]$ is equal to the set of n minimum weight triples in

$$R[u,v,n] \cup R_{\preceq}[u_1,v,n] \cup R_{\preceq}[u_2,v,n] \cup \cdots \cup R_{\preceq}[u_{m(u)},v,n]. \tag{15}$$

For each v, it takes $O(n^2)$ to compute all of these sets $R[u,v,n]$ using a post-order traversal. There are $O(n)$ choices for v so extracting all the sets $R_{\preceq}[u,v,n]$ takes $O(n^3)$ time.

Let $U = C(u)$. Then $r(U) = \bigcup_{v \succ_T u} R_{\preceq}[u,v]$ and so the minimum $|U|-1$ triples in $r(U)$ are the minimum $|U|-1$ triples in $\bigcup_{v \succ_T u} R_{\preceq}[u,v,n]$. Since there are at most n^2 triples in $|\bigcup_{v \succ_T u} R_{\preceq}[u,v,n]|$ we can compute

$$\iota_w^c(U) = \mathrm{avmin}\,(r(U), |U|-1)$$

in $O(n^2)$ time. It therefore takes $O(n^3)$ time to compute the $O(n)$ isolation indices. ∎

We now consider the generalisation from clean clusters of ρ_s to clean clusters of an arbitrary isolation weighting w. There is apparently no analogue of Lemma 2.2 for C-clusters. We have constructed an example for five

elements where a clean cluster of w is not a clean cluster for the similarity function $s(a,b) = |\{ab|y : w(ab|y) > 0\}|$ of eq. (12). We were also unable to find a variation on s for which a version of Lemma 2.2 might hold.

We follow an alternative approach. Given an isolation weighting w and $r \in X$, put

$$CH(w,r) = \{A \subseteq X : w(ar|x) > 0 \text{ for all } a \in A,\, x \in X - A\} \qquad (16)$$

LEMMA 2.3. *The set $CH(w,r)$ forms a chain, that is, for each $A, B \in CH(w,r)$ either $A \subseteq B$ or $B \subseteq A$.*

Proof. From the definition of an isolation weighting we deduce $w(xr|r) \leq 0$ for all $x \in X$ and so if $A \in CH(w,r)$ and $A \neq \emptyset$ then $r \in A$. Given any two clusters $A, B \in CH(w,r)$, if there is a, b such that $a \in A - B$ and $b \in B - A$ then we obtain $w(ar|b) > 0$ and $w(br|a) > 0$, a contradiction. ▮

LEMMA 2.4. *We can construct a chain containing $CH(w,r)$ in $O(n^2)$ time.*

Proof. Let G be the semi-complete digraph with vertex set X and edge set

$$E = \{(u,v) : w(ur|v) \leq 0\}.$$

Use the $O(n^2)$ algorithm of [16] to determine a (directed) Hamiltonian path $a_1, a_2, \ldots, a_n$ for G. For each $A \in CH(w,r)$ there is no edge (u,v) in E such that $u \in A$ and $v \notin A$. Hence there must be i such that $A = \{a_i, a_{i+1}, \ldots, a_n\}$. The collection of sets $\{a_i, a_{i+1}, \ldots, a_n\}$ for $i = 1, \ldots, n$ is a chain containing all of the clusters in $CH(w,r)$. ▮

Our method for constructing the clean clusters of an isolation weighting is iterative. We first restrict the isolation weighting to three elements, then incorporate one element at a time into the analysis. The basis for the iterative algorithm, and the connection between chains and clean clusters is provided by:

THEOREM 2.5. *Let w be an isolation weighting for X. If A is a clean cluster of w and $r \in X$ then either $A \in CH(w,r)$ or $A - \{r\}$ is a clean cluster of w restricted to $X - \{r\}$, or both.*

Proof. Suppose that A is not a clean cluster of w restricted to $X - \{r\}$. Then there is R such that

$$R \subseteq \{aa'|x : a, a' \in A - \{r\},\, x \in X - \{r\}\},$$

$|R| = |A - \{r\}| - 1$ and $\sum_{aa'|x \in R} w(aa'|x) \leq 0$. If A is also not a cluster in $CH(w, r)$ then there is $a \in A$ and $x \in X - A$ such that $w(ar|x) < 0$. Putting $R' = R \cup \{ar|x\}$ we obtain a subset of $r(A)$ containing at least $|A| - 1$ elements such that $\sum_{aa'|x \in R'} w(aa'|x) \leq 0$. Hence A is not a clean cluster of w. ∎

Theorem 2.5 leads directly to a polynomial time algorithm for constructing clean clusters of an isolation weighting. Pseudo code is presented in appendix A. (Algorithm 3).

COROLLARY 2.4. *The clean clusters of an isolation weighting can be computed in $O(n^4)$ time.*

Proof. We use an iterative algorithm. Order X arbitrarily as $X = \{x_1, x_2, \ldots, x_n\}$, putting $X_k = \{x_1, x_2, \ldots, x_k\}$ for each $k : 1 \leq k \leq n$. Let $w_{|X_k}$ denote the restriction of w to X_k and let $\mathcal{C}_k$ denote the clean clusters for $w_{|X_k}$. We compute $\mathcal{C}_2$ directly.

By Theorem 2.5,

$$\mathcal{C}_{k+1} \subseteq CH(w_{|X_{k+1}}, x_{k+1}) \cup \{A \subseteq X_{k+1} : A - \{x_{k+1}\} \in \mathcal{C}_k\}.$$

We construct $CH(w_{|X_{k+1}}, x_{k+1})$ using Lemma 2.4. The clean cluster indices of the clusters $A \in CH(w_{|X_{k+1}}, x_{k+1})$ can be computed in $O(n^3)$ time using the same technique as in the proof of Corollary 2.3. The clean cluster indices of the clusters in $\{A : A - \{x_{k+1}\} \in \mathcal{C}_k\}$ can be computed by maintaining a list $\hat{r}[A, k]$ of n minimum weight triples for each cluster A. After each element addition,

$$\hat{r}[A, k+1] \subseteq \hat{r}[A, k] \cup \{aa'|x_{k+1} : a, a' \in A\} \quad (17)$$

$$\hat{r}[A \cup \{x_{k+1}\}, k+1] \subseteq \hat{r}[A, k] \cup \{ax_{k+1}|y : a \in A, y \in X_k - A\} \quad (18)$$

Hence we can construct $\hat{r}[A, k+1]$ and $\hat{r}[A \cup \{x_{k+1}\}, k+1]$ in $O(n^2)$ time per cluster A. Each iteration step from $\mathcal{C}_k$ to $\mathcal{C}_{k+1}$ can be completed in $O(n^3)$ time, giving $O(n^4)$ time for the entire construction. ∎

2.5. Construction of stable clusters

The stable cluster method is the last, and most refined, method in the series. Let s be a similarity function. By Theorem 2.4 we know that any average linkage tree T for s contains all of the stable clusters of s. All that remains is to examine the $O(n)$ clusters in $clus(T)$ and discard those clusters that are not stable. This proves to be a significantly more difficult problem.

THEOREM 2.6. *It is an NP-hard problem to construct the set of stable clusters of a similarity function. Hence it is also NP-hard to construct the stable clusters of an arbitrary isolation weighting.*

Proof.
We provide a transformation from:

SPARSEST CUT
Instance: Vertex set V, cost $c(u,v)$ for every unordered pair $\{u,v\} \in V \times V$. Rational k.
Question: Is there $U \subsetneq V$ such that

$$\sum_{u \in U, v \in V-U} c(u,v) < k|U||V-U| \quad ?$$

SPARSEST CUT was shown to be NP-hard by [19].

Let V, c and k make up an arbitrary instance of SPARSEST CUT. Put $X = V \cup \{z\}$, where z is a new element. Define a similarity function s on X by $s(u,v) = c(u,v)$ for all $u, v \in V$ and $s(z,v) = k$ for all $v \in V$. For all $u, v \in V$ we have $\rho_s(uv|z) = c(u,v) - k$. We claim that $\iota_s^s(V) > 0$ if and only if there is no $U \subset V$ such that $\sum_{u \in U, v \in V-U} c(u,v) \leq k|U||V-U|$.

All clusters incompatible with V are of the form $U \cup \{z\}$ for some $U \subset V$. For such a cluster, we have

$$\begin{aligned} \overline{w}(U(V-U)|\{z\}) &= \tfrac{1}{|U||V-U|} \sum_{u \in U, v \in V-U} \rho_s(uv|z) \\ &= -k + \sum_{u \in U, v \in V-U} \tfrac{c(u,v)}{|U||V-U|} \end{aligned}$$

Hence $\iota_s^s(V) > 0$ if and only if there is no U such that

$$\sum_{u \in U, v \in V-U} c(u,v) \leq k|U||V-U|.$$

The result follows. ∎

3. A FAMILY OF SPLIT METHODS

In this section we present a series of methods for classification using splits. The series is analogous to the clustering constructions investigated in the previous section. Indeed, the duality between the cluster and split

cases fits nicely into the framework of [11]: the splits are to clusters what projective space is to affine space.

Our presentation of the split and unrooted tree constructions follows the same lines as the presentation of the clustering methods.

3.1. Definitions

Unrooted X-trees and splits were defined in Section 1. Put $n = |X|$. Let $splits(T)$ denote the set of splits of a tree T. A set of splits $\mathcal{S}$ of X is **compatible** if there is a tree T such that $\mathcal{S} \subseteq splits(T)$. Buneman showed that $\mathcal{S}$ is compatible if and only if for each pair $A|B$, $C|D$ of splits in $\mathcal{S}$ at least one of $A \cap C$, $A \cap D$, $B \cap C$, $B \cap D$ is empty [10].

A **quartet** $ab|cd$ defines a separation of a and b from c and d. The set of all possible quartets on X is denoted $\mathcal{Q}(X)$. The set of quartets of a split $A|B$ is defined by

$$q(A|B) = \{aa'|bb' : a, a' \in A,\ b, b' \in B\}.$$

We say that $ab|cd$ is a quartet of T if there is a split $A|B \in splits(T)$ such that $ab|cd \in q(A|B)$. The set of quartets of T is denoted $q(T)$.

A **dissimilarity function** is a symmetric function from $X \times X$ to $\Re^+$. A **separation weighting** is a function $\omega : \mathcal{Q}(X) \to \Re$ such that $\omega(ab|cd) + \omega(ac|bd) \leq 0$ for all $a, b, c, d \in X$. The weight $\omega(ab|cd)$ indicates the degree to which a and b are separated from c and d. Given a dissimilarity function δ, define

$$\beta_\delta(ab|cd) = \tfrac{1}{2}\left(\min\{\delta(a,c) + \delta(b,d), \delta(a,d) + \delta(b,c)\} - \delta(a,b) - \delta(c,d)\right) \qquad (19)$$

Then β_δ is a separation weighting.

Given a set Q of quartets, a separation weighting ω, and a positive integer $k \leq |Q|$, define

$$\mathrm{av}(Q) = \frac{1}{|Q|} \sum_{ab|cd \in Q} \omega(ab|cd) \qquad (20)$$

$$\mathrm{avmin}(Q, k) = \min_{Q'}\{av(Q') : Q' \subseteq Q,\ |Q'| = k\}. \qquad (21)$$

Given non-empty disjoint subsets U, V, Y, Z define

$$\overline{\omega}(UV|YZ) = \mathrm{av}\{uv|yz : u \in U,\ v \in V,\ y \in Y,\ z \in Z\}, \qquad (22)$$

the average weight of all quartets $uv|yz$ with $u \in U$, $v \in V$, $y \in Y$, and $z \in Z$.

3.2. A family of tree construction methods

In Section 2.2 we defined a collection of isolation indices and showed that, for each index, the clusters with positive index formed a hierarchy. In

this section we do the same for splits: we define a collection of separation indices and then show that the resulting collections of splits are compatible.

Definition of Split Indices *Let ω be a separation index and $A|B$ a split of X.*

(i) The **strong separation index** *of $A|B$ is defined*

$$\mu_\omega(A|B) = \min_{uv|yz} \{\omega(uv|yz) : uv|yz \in q(A|B)\}$$

and the **Buneman splits** *for ω are $\{A|B : \mu_\omega(A|B) > 0\}$.*

(ii) The **refined Buneman index** *of $A|B$ is defined*

$$\mu^r_\omega(A|B) = avmin(q(A|B), n-3)$$

and the **refined Buneman splits** *for ω are $\{A|B : \mu^r_\omega(A|B) > 0\}$.*

(iii) The **clean split index** *of $A|B$ is defined*

$$\mu^c_\omega(A|B) = avmin(q(A|B), (|A|-1)(|B|-1))$$

and the **clean splits** *for ω are $\{A|B : \mu^c_\omega(A|B) > 0\}$.*

(iv) The **split stability index** *of $A|B$ is defined*

$$\mu^s_\omega(A|B) = \min_{U,V,Y,Z \neq \emptyset} \{\overline{\omega}(UV|YZ) : A = U \uplus V,\ B = Y \uplus Z\}$$

and the **stable splits** *of ω are $\{A|B : \mu^s_\omega(A|B) > 0\}$.*

The Buneman splits of a dissimilarity function δ are the Buneman splits of $\omega = \beta_\delta$. Similarly for refined Buneman splits, clean splits, and stable splits.

The refinement relations between the different methods are presented in Theorem 3.1. Theorem 3.2 establishes that each construction gives compatible splits. Both theorems use the following bounds on the size of the conflict set for incompatible splits. The indices, and the construction complexities, for all of the split methods are summarised in table 2.

LEMMA 3.1. *If U, V, Y, Z are non-empty subsets of X such that $A = U \uplus V$ and $B = Y \uplus Z$ then*

$$|U||V||Y||Z| \geq (|A|-1)(|B|-1) \geq n-3.$$

Proof. Put $p = |V|$ and $q = |Z|$. Then

$$|U||V||Y||Z| = (|A|-p)p(|B|-q)q. \tag{23}$$

Using straightforward calculus we can show that $(|A| - p)p \geq (|A| - 1)$ and $(|B| - q)q \geq (|B| - 1)$, giving the first inequality. For the second, put $m = (|B| - 1)$ so that $|A| - 1 = n - 2 - m$ and $(|A| - 1)(|B| - 1) = (n - 2 - m)m$. The minimum $n - 3$ is obtained when $m = 1$. ∎

THEOREM 3.1. *Let ω be a separation weighting on X. For any split $A|B$ we have*

$$\mu_\omega(A|B) \leq \mu^r_\omega(A|B) \leq \mu^c_\omega(A|B) \leq \mu^s_\omega(A|B).$$

Hence the set of Buneman splits is contained within the set of refined Buneman splits which is contained within the set of clean splits which is contained within the set of stable splits.

Proof. The inequalities $\mu_\omega(A|B) \leq \mu^r_\omega(A|B) \leq \mu^c_\omega(A|B)$ follow from the observation that $k \leq k'$ implies $\text{avmin}(q(A|B), k) \leq \text{avmin}(q(A|B), k')$. Choose non-empty and disjoint subsets U, V, Y, Z such that $A = U \uplus V$, $B = Y \uplus Z$, and $\mu^s_w(A|B) = \overline{\omega}(UV|YZ)$. Then

$$\begin{aligned}
\mu^c_\omega(A|B) &= \text{avmin}(q(A|B), (|A| - 1)(|B| - 1)) && (24)\\
&\leq \text{avmin}(q(A|B), |U||V||Y||Z|) && (25)\\
&\leq \overline{\omega}(UV|YZ) && (26)\\
&= \mu^s_w(A|B). && (27)
\end{aligned}$$

∎

THEOREM 3.2. *Let ω be a separation weighting. The sets of Buneman splits, refined Buneman splits, clean splits, and stable splits of ω are all compatible collections of splits.*

Proof. We prove that the set of stable splits is compatible. The other compatibility results follow from Theorem 3.1.

Let $A|B$ be a stable split and let $C|D$ be a split incompatible with $A|B$. Put $U = A \cap C$, $V = A \cap D$, $Y = B \cap C$, $Z = B \cap D$. Since $A|B$ and $C|D$ are incompatible, U, V, Y, and Z are non-empty. Then

$$\begin{aligned}
\mu^s_\omega(A|B) + \mu^s_\omega(C|D) &\leq \overline{\omega}(UV|YZ) + \overline{\omega}(UY|VZ)\\
&= \frac{1}{|U||V||Y||Z|} \sum_{u \in U} \sum_{v V} \sum_{y \in Y} \sum_{z \in Z} \Big(\omega(uv|yz) + \omega(uy|vz)\Big)\\
&\leq 0
\end{aligned}$$

and so $\mu^s_\omega(A|B) > 0$ implies $\mu^s_\omega(C|D) < 0$. It follows that the set of stable splits is compatible. ∎

The relationship between the tree construction methods parallels that for the clustering methods. The Buneman split method is quite conservative. For $A|B$ to be a Buneman split of ω we must have $\omega(aa'|bb') > 0$ for all $aa'|bb' \in q(A|B)$. The refined Buneman index allows $n-3$ quartets in $q(A|B)$ to have negative average weight. The clean split method incorporates the size of A and B into the calculation of the index. The bound of $(|A|-1)(|B|-1)$ stems from Lemma 3.1. If we increase this bound then clean splits will no longer necessarily be stable splits and, additionally, we lose the guarantee of compatibility for the splits selected.

The stable split method allows the largest number of negative weight quartets. A split $A|B$ can have $O(n^4)$ negative weight quartets in $q(A|B)$ and yet still be stable. There will always, however, be more positive weight quartets in $q(A|B)$ than negative ones. The stability index of a split can be viewed as the measure of support for a split $A|B$ on exactly those quartets with which $A|B$ is in conflict with an incompatible split $C|D$.

3.3. Properties and construction of Buneman splits

We now formalise the connection between the Buneman split method and Apresjan clusters. Let δ be a dissimilarity function on X. The *Farris Transform* of δ with respect to $x \in X$ is the similarity function s_x on $X - \{x\}$ with

$$s_x(a,b) = \tfrac{1}{2}(\delta(a,x) + \delta(b,x) - \delta(a,b)) \tag{28}$$

for all $a, b \in X - \{x\}$ [13]. The inverse of the Farris transform is given by

$$\delta(a,b) = s_x(a,a) + s_x(b,b) - 2s_x(a,b)$$

for all $a, b \in X - \{x\}$, and $\delta(a,x) = s_x(a,a)$ as well as $\delta(x,x) = 0$.

The Farris transform links Buneman splits and Apresjan clusters. This connection leads to the suprising algorithmic result that Buneman splits of a dissimilarity function can be constructed in $O(n^3)$ time, even though the definition appears to imply that $O(n^4)$ quartets need to be considered.

THEOREM 3.3. *(i) Let δ be a dissimilarity function on X and let s_x denote the Farris transform of δ with respect to $x \in X$. For any split $A|B$ of X,*

$$\mu_\delta(A|B) = \min_b\{\iota_{s_b}(A) : b \in B\} = \min_a\{\iota_{s_a}(B) : a \in A\}. \tag{29}$$

Thus $A|B$ is a Buneman split for δ if and only if A is an Apresjan cluster of s_b for all $b \in B$, and this holds if and only if B is an Apresjan cluster of s_a for all $a \in A$. (ii) The Buneman splits of a dissimilarity function δ can be computed in $O(n^3)$ time.

Proof. Eq. (29) follows immediately from the observation that $\beta_\delta(ab|cx) = \rho_{s_x}(ab|c)$.

We now prove (ii). For each $x \in X$ we compute the Farris transform s_x of δ with respect to x ($O(n^2)$ time for each x) and then construct the Apresjan clusters of s_x in $O(n^2)$ time (Corollary 2.1). Let S_x be the set of splits $A|(X-A)$ such that A is a strong cluster of s_x. The Buneman splits are exactly those splits appearing in all of the sets of splits S_x, and the corresponding separation indices are given by eq. (29). Hence the entire computation takes $O(n^3)$ time. ▮

The Farris transform can be extended to a general separation weighting ω. For each $x \in X$, define an isolation weighting w_x on rooted triples of $X - \{x\}$ by $w_x(ab|c) = \omega(ab|cx)$ for all $a, b, c \in X - \{x\}$.

PROPOSITION 3.1. *For all splits $A|B$,*

$$\mu_\omega(A|B) = \min_b\{\iota_{w_b}(A) : b \in B\} = \min_a\{\iota_{\omega_a}(B) : a \in A\}.$$

Hence $A|B$ is a Buneman split for ω if and only if A is a strong cluster of w_b for all $b \in B$, if and only if B is a strong cluster of w_a for all $a \in A$.

Proof. Follows directly from the definition $w_x(ab|c) = \omega(ab|cx)$. ▮

Proposition 3.1 leads directly to a new $O(n^4)$ time algorithm for constructing the Buneman splits of an separation weighting. The first $O(n^4)$ time algorithm for this problem was given by [7].

3.4. Properties and construction of refined Buneman splits

Theorem 3.3 and Lemma 3.1 link the Buneman method for constructing unrooted trees with the strong cluster method for constructing rooted trees. There appears to be no analogous relationship between the refined Buneman tree and a clustering method. One can easily construct a dissimilarity matrix δ on five points for which there exists a refined Buneman split $A|B$ such that A is not a stable cluster of s_a for some $a \in A$. Hence A would also not be a C-cluster or strong cluster for s_a.

We can, however, exploit another kind of connection between refined Buneman splits and the clustering methods. Theorem 3.4, similar in structure to Theorem 2.5, was used by [9] to show that the refined Buneman tree can be constructed in polynomial time. For an arbitrary $r \in X$, let w_r denote the isolation weighting on $X - \{r\}$ given by $w_r(ab|c) = \omega(ab|cr)$ (c.f. Section 3.3). The basis of the iterative algorithm is

THEOREM 3.4. *[9] Let ω be an separation weighting for X. If $A|B$ is a refined Buneman split of ω and $r \in B$ then either A is a strong cluster of w_r or $A|(B - \{r\})$ is a refined Buneman split of ω restricted to $X - \{r\}$.*

The algorithm of [9] runs in $O(n^6)$ time. Here we improve this complexity to $O(n^5)$ time, first showing how refined Buneman indices can be quickly computed on a tree. Pseudocode for the computation of refined Buneman indices can be found in Appendix (algorithm 4).

LEMMA 3.2. *Let T be an X-tree. We can compute $\mu^r_\omega(A|B)$ for each split $A|B \in splits(T)$ in $O(n^4)$ time.*

Proof. We adapt the algorithm presented in the proof of corollary 2.3 to quartets. We first root T at an arbitrary node. For each pair of vertices u and v in T let $Q[u, v]$ be the set of quartets $ab|cd$ such that the path from a to c intersects the path from b to d exactly along the path from u to v. Then every quartet $ab|cd$ induced by the previously unrooted tree corresponding to T appears in exactly two sets $Q[u, v]$ (noting $Q[u, v] = Q[v, u]$), and we can construct these sets in $O(n^4)$ time.

Let $Q[u, v, n-3]$ denote the $n-3$ minimum weight quartets in $Q[u, v]$. Using the algorithm of [8] we can compute $Q[u, v, n-3]$ from $Q[u, v]$ in $O(|Q[u, v]|)$ time. Hence we can compute all of the sets $Q[u, v, n-3]$ in $O(n^4)$ time.

For each pair (u, v) such that $v \not\preceq u$ we define

$$Q_\preceq[u, v] = \bigcup_{u' \preceq_T u} Q[u', v]$$

and let $Q_\preceq[u, v, n-3]$ denote the set of $n-3$ minimum weight quartets in $Q_\preceq[u, v]$. We can compute the sets $Q_\preceq[u, v, n-3]$ using one post-order traversal for each choice of v: Let $u_1, u_2, \ldots, u_m$ be the children of u. Then $Q_\preceq[u, v, n-3]$ equals the set of $n-3$ minimum weight quartets in

$$Q[u, v, n-3] \cup Q_\preceq[u_1, v, n-3] \cup \cdots \cup Q_\preceq[u_m, v, n-3] \tag{30}$$

For each v it takes $O(n^2)$ time to compute all of these sets $Q_\preceq[u, v, n-3]$ using a post-order traversal. There are $O(n)$ choices for v so extracting all of the sets $Q_\preceq[u, v, n-3]$ takes $O(n^3)$ time.

Given a vertex u, let U be the set of leaves that are descendents of u and put $V = X - U$. Then $q(U|V) = \bigcup_{v: v \not\preceq_T u} Q_\preceq[u, v]$ and so the $n-3$ minimum weight quartets in $q(U|V)$ are the $n-3$ minimum weight quartets

in

$$Q_u = \bigcup_{v \not\prec_T u} Q_{\preceq}[u, v, n-3].$$

The refined Buneman index $\mu^r_w(U|V)$ can therefore be computed in $O(n^2)$ time. The entire computation takes $O(n^4)$ time. ▌

COROLLARY 3.1. *The refined Buneman splits of a separation weighting ω can be constructed in $O(n^5)$ time.*

Proof. We use an iterative algorithm, whose pseudocode is given in appendix (Algorithm 5). Order X arbitrarily as $X = \{x_1, x_2, \ldots, x_n\}$, putting $X_k = \{x_1, x_2, \ldots, x_k\}$ for each $k : 1 \leq k \leq n$. Let $\omega_{|X_k}$ denote the restriction of ω to X_k and let $\mathcal{R}_k$ denote the refined Buneman splits for $\omega_{|X_k}$. We compute $\mathcal{R}_2$ directly.

Put $r = x_{k+1}$. Let $\mathcal{C}_{k+1}$ denote the set of strong clusters of the isolation weighting w_r on X_k given by $w_r(ab|c) = \omega(ab|cr)$. By Theorem 3.4,

$$\mathcal{R}_{k+1} \subseteq \{(A \cup \{r\})|B : A|B \in \mathcal{R}_k\} \cup \{A|(X_{k+1} - A) : A \in \mathcal{C}_{k+1}\}.$$

All refined Buneman indices of splits in $\{(A \cup \{r\})|B : A|B \in \mathcal{R}_k\}$ can be computed in $O(n^4)$ time in total by maintaining a list $q_{min}[A|B, k]$ of $n-3$ minimum quartets for each split $A|B \in \mathcal{R}_k$. After each element addition

$$q_{min}[A \cup \{x_{k+1}\}|B, k+1] \subseteq q_{\min}[A|B, k] \cup \{ax_{k+1}|bb' : a \in A, b, b' \in B\} \tag{31}$$

and so $q_{min}[A \cup \{x_{k+1}\}|B, k+1]$ can be obtained in $O(n^3)$ time. Similarly for $q_{min}[A|B \cup \{x_{k+1}\}, k+1]$. The refined Buneman indices of splits in $\{A|(X_{k+1} - A) : A \in \mathcal{C}_{k+1}\}$ can be computed in $O(n^4)$ time using Lemma 3.2. Hence we can compute $\mathcal{R}_{k+1}$ from $\mathcal{R}_k$ in $O(n^4)$ time and the entire construction takes $O(n^5)$ time. ▌

3.5. Properties and construction of clean splits

There is apparently no analogue of Theorem 3.3 for clean splits. We have constructed an example on eight elements where δ has a clean split $A|B$ such that A is not in the average linkage tree for s_b for *any* $b \in B$.

In Section 2.4 we showed how clean clusters can be constructed iteratively using a connection with rooted chains (Theorem 2.5). Here we show that an similar connection applies to the clean split case. Let w_r denote the isolation weighting on $X - \{r\}$ given by $w_r(ab|c) = \omega(ab|cr)$ (c.f. Section 3.3).

THEOREM 3.5. *Let ω be a separation weighting for X. If $A|B$ is a clean split of ω and $r \in B$ then either A is a clean cluster of w_r or $A|(B - \{r\})$ is a clean split of ω restricted to $X - \{r\}$.*

Proof. Suppose that $A|(B - r)$ is not a clean split of ω restricted to $X - \{r\}$. Then there is $Q \subseteq q(A|B - \{r\})$ such that $|Q| = (|A| - 1)(|B - \{r\}| - 1)$ and $\sum_{aa'|bb' \in Q} \omega(aa'|bb') \leq 0$. If A is not a clean cluster of w_r then there is R such that

$$R \subseteq \{aa'|x : a, a' \in A,\ x \in X - \{r\}\},$$

$|R| = |A| - 1$ and $\sum_{aa'|x \in R} w_r(aa'|x) \leq 0$. Put $Q_R = \{aa'|xr : aa'|x \in R\}$, so

$$\sum_{aa'|xr \in Q_R} \omega(aa'|xr) \leq 0.$$

Put $Q' = Q \cup Q_R$. Then Q' is a subset of $q(A|B)$ containing $(|A| - 1)(|B| - 2) + (|A| - 1) = (|A| - 1)(|B| - 1)$ quartets such that $\sum_{ab|cd \in Q'} \omega(ab|cd) \leq 0$. Hence $A|B$ is not a clean split of ω. ∎

Theorem 3.5 provides the reduction step of a polynomial time iterative algorithm for constructing the set of clean splits.

THEOREM 3.6. *The clean splits of a separation weighting ω can be constructed in $O(n^5)$ time.*

Proof. First note that we can compute $\mu^c_\omega(A|B)$ for each split $A|B \in splits(T)$ in $O(n^4)$ time by using a variant of Lemma 3.2. The $O(n^5)$ time algorithm is almost identical to that for refined Buneman splits described in Corollary 3.1 (c.f. Algorithms 4 and 5). ∎

The Quartet Cleaning method of [6] is a special case of the unrooted C-tree construction. Let Q be a set of quartets such that for each $a, b, c, d \in X$ at most one of $ab|cd$, $ac|bd$, $ad|bc$ is in Q. Define a separation weighting ω by putting $\omega(ab|cd) = 1$ if $ab|cd \in Q$ and $\omega(ab|cd) = -1$ if $ab|cd \notin Q$. Then $A|B$ is a clean split of ω if and only if $|q(A|B) - Q| < (|A| - 1)(|B| - 1)/2$, the condition for $A|B$ to be a "clean" split with respect to the terminology of [6].

3.6. Construction of stable splits

Like the stable clusters, the stable splits are the last, and most refined, in the series of constructions. As with stable clusters, the stable splits are difficult to construct:

THEOREM 3.7. *It is an NP-hard problem to construct the set of stable splits of a dissimilarity function. Hence it is also NP-hard to construct the stable clusters of an arbitrary separation weighting.*

Proof. Let V, c and k make up an arbitrary instance of SPARSEST CUT. We construct a dissimilarity function δ such that $\beta_\delta(uv|xy) = c(u,v) - k$ for all $u, v \in V$. The result then follows by the same arguments used to prove Theorem 2.6. ▌

4. DISCUSSION

We have presented two parallel series of tree based classification methods, one giving clusters and rooted trees, the other giving splits and unrooted trees. The progression began with two classical constructions, Apresjan clusters and Buneman splits, and finished with two general (but NP-hard) constructions, stable clusters and stable splits. We explored links between the different constructions and with well-known classification methods.

We see two major directions for future work. The first is to investigate the performance of these methods in an applied field of classification such as phylogenetics. Three variants of the methods described here [6, 7, 17] have already been employed by computational biologists. The appeal of combinatorial approaches in phylogenetics is that quartet weights can be determined using complex phylogenetic criteria and models that are computationally infeasible for larger sets of taxa. Quartet weights are then used to construct a tree in a reasonable amount of time. The clean split construction is particularly well-suited for incorporation in a combinatorial strategy for phylogenetic reconstruction. The method, like others presented here, is continuous, clearly defined, has a polynomial time algorithm and can be used to analyse both dissimilarity data and quartet weights.

A second direction of future work is further generalisation. Bandelt and Dress [3] established a duality between weak hierarchies and weakly compatible splits [3]. We have already used this to provide a more efficient algorithm for split decomposition [5]. A series of general clustering methods, extending work on weak hierarchies, was explored in [11]. The question arises of whether these two methods can be refined like the strong clusters and Buneman tree, and whether the connections with single linkage/average linkage can be generalised to agglomerative methods for non-hierarchical clustering.

APPENDIX C
Algorithms

In the following algorithms we let $\mathcal{L}(u)$ denote the labels of the leaves that are descendents of the node u.

ALGORITHM 1 (*Compute* $\iota_w^c(U)$ *for all* $U \in clus(T)$*).*

Compute and tabulate $\text{lca}(x, x')$ for all leaves x, x' in T.

Iterate through all rooted triples $xy|z \in r(T)$ and use lca tables to construct $R[u, v]$ for all $u, v \in V(T)$ such that $u \prec_T v$.

For each pair $u, v \in V(T)$ such that $u \prec_T v$ use the algorithm of [8] to determine the set $R[u, v, n]$ of n minimum weight triples in $R[u, v]$.

For all vertices $v \in V(T)$ do

  For all vertices u in a post-order traversal of T do

    Use eq. (15) and [8] to construct $R_{\preceq}[u, v, n]$.

For all vertices $u \in V(T)$ compute

  $R_u \leftarrow \cup_{v \succ_T u} R_{\preceq}[u, v, n]$

  $\iota_w^c(\mathcal{L}(u)) \leftarrow \text{avmin}(\{w(xy|z) : xy|z \in R_u\}, |\mathcal{L}(u) - 1|)$, using [8].

Output cluster indices $\iota_w^c(\mathcal{L}(u))$ for all $u \in V(T)$.

ALGORITHM 2 (*Rooted C-tree for a similarity function* s *on* X*).*

Compute an average linkage tree T for s.

Use algorithm 0 with $w = \beta_s$ to compute $\iota_w^c(U)$ for all $U \in clus(T)$

Output $RC(\beta_s) = \{U \in clus(T) : \iota_w(U) > 0\}$.

ALGORITHM 3 (*Rooted C-tree for an isolation weighting w).*

Order X arbitrarily as $x_1, x_2, \ldots, x_n$

$\mathcal{C}_2 \leftarrow \{\{x_1\}, \{x_2\}\}$

$\hat{r}[\{x_1\}, 2] \leftarrow \{x_1x_1|x_2\}$; $\hat{r}[\{x_2\}, 2] \leftarrow \{x_2x_2|x_1\}$.

For k from 2 to $n-1$ do

Construct $CH(w_{|X_{k+1}}, x_{k+1})$ using proof of Lemma 2.4.

Use Algorithm 1 to compute $\iota_w^c(U)$ for all $U \in CH(w_{|X_{k+1}}, x_{k+1})$.

$\mathcal{C}_{k+1} \leftarrow \{U \in CH(w_{|X_{k+1}}, x_{k+1}) : \iota_w^c(U) > 0\}$

For all $A \in \mathcal{C}_k$ do

Compute $\hat{r}[A, k+1]$ from $\hat{r}[A, k] \cup \{aa'|x_{k+1} : a, a' \in A\}$.

Compute $\iota_w^c(A)$ using $\hat{r}[A, k+1]$ and [8].

If $\iota_w^c(A) > 0$ then add A to $\mathcal{C}_{k+1}$.

Compute $\hat{r}[A \cup \{x_{k+1}\}, k+1]$ from $\hat{r}[A, k] \cup \{ax_{k+1}|y : a \in A, y \notin A\}$.

Compute $\iota_w^c(A \cup \{x_{k+1}\})$ using $\hat{r}[A \cup \{x_{k+1}\}, k+1]$ and [8].

If $\iota_w^c(A \cup \{x_{k+1}\}) > 0$ then add $A \cup \{x_{k+1}\}$ to $\mathcal{C}_{k+1}$.

Output $\mathcal{C}_n$.

ALGORITHM 4 (*Compute* $\mu^r_w(U|X-U)$ *for all splits* $U|(X-U)$ *of* T*).*

Root T at an arbitrary vertex.

Compute and tabulate $\mathrm{lca}(x, x')$ for all leaves x, x' in T.

Iterate through all $a, b, c, d \in X$ and use lca tables to

construct $Q[u, v]$ for all $u, v \in V(T)$.

For each pair $u, v \in V(T)$ use the algorithm of [8] to

extract $Q[u, v, n-3] \subseteq Q[u, v]$.

For all vertices $v \in V(T)$ do

For all vertices u for which $v \not\preceq u$ in a post-order traversal of T do

Use eq. (30) and [8] to construct $Q_{\preceq}[u, v, n-3]$.

For all vertices $u \in V(T)$ put $U = \mathcal{L}(u)$ and compute

$Q_u \leftarrow \bigcup_{v \not\preceq_T u} Q_{\preceq}[u, v, n-3]$

$\mu^r_w(U|X-U) \leftarrow \mathrm{avmin}(\{w(ab|cd) : ab|cd \in Q_u\}, n-3)$.

Output split indices $\mu^r_w(U|X-U)$ for all $U|(X-U) \in \mathit{splits}(T)$.

ALGORITHM 5 (*Refined Buneman tree for w).*

Order X arbitrarily as $x_1, x_2, \ldots, x_n$

$\mathcal{R}_3 \leftarrow \{x_i|X_3-x_i : i = 1,2,3\}$

For $i = 1,2,3$ do

$q_{min}[x_i|X_3-x_i] = q(x_i|X_3-x_i)$

For k from 3 to $n-1$ do

Let T be the strong cluster tree for $w_{|X_{k+1}}$.

Use Algorithm 4 to compute $\mu_w^r(U|X_{k+1}-U)$ for all $U \in clus(T)$.

$\mathcal{R}_{k+1} \leftarrow \{U|X_{k+1}-U : U \in clus(T)$ and $\mu_w^r(U|X_{k+1}-U) > 0\}$.

For all $U|V \in \mathcal{R}_k$ do

Construct $q_{min}[U\cup\{x_{k+1}\}|V, k+1]$ using eq. (31) and [8].

Compute $\mu_w^r(U\cup\{x_{k+1}\}|V)$ using $q_{min}[U\cup\{x_{k+1}\}|V, k+1]$ and [8].

If $\mu_w^r(U\cup\{x_{k+1}\}|V) > 0$ then add $U\cup\{x_{k+1}\}|V$ to $\mathcal{R}_{k+1}$.

Construct $q_{min}[U|V\cup\{x_{k+1}\}, k+1]$ using eq. (31) and [8].

Compute $\mu_w^r(U|V\cup\{x_{k+1}\})$ using $q_{min}[U|V\cup\{x_{k+1}\}, k+1]$ and [8].

If $\mu_w^r(U|V\cup\{x_{k+1}\}) > 0$ then add $U|V\cup\{x_{k+1}\}$ to $\mathcal{R}_{k+1}$.

Output $\mathcal{R}_n$.

ACKNOWLEDGMENT

We would like to thank Olivier Gascuel, Vincent Moulton and Mike Steel for reading through versions of this manuscript.

REFERENCES

1. J.D. Apresjan. An algorithm for constructing clusters from a distance matrix. *Mashinnyi perevod: prikladnaja lingvistika*, 9:3–18, 1966.
2. H.-J. Bandelt and A.W.M. Dress. Weak hierarchies associated with similarity measures - an additive clustering technique. *Bulletin of Mathematical Biology*, 51(1):133–166, 1989.
3. H-J Bandelt and A.W.M Dress. A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, 92:47–105, 1992.
4. J.-P. Barthélémy and A. Guénoche. *Trees and proximity representations*. John Wiley & Sons Ltd., Chichester, 1991.

5. V. Berry and D. Bryant. Faster reliable phylogenetic analysis. In *Proc. 3rd international conference on comp. mol. biol. (RECOMB)*, volume 3, pages 59–68, 1999.

6. V. Berry, T. Jiang, P. Kearney, and M. Li. Quartet cleaning: Improved algorithms and simulations. *Lecture Notes in Computer Science*, 1643:313–324, 1999.

7. Vincent Berry and Olivier Gascuel. Inferring evolutionary trees with strong combinatorial evidence. *Theoretical computer science*, 240(2):271–298, 2000.

8. M. Blum, V. Pratt, R.E. Tarjan, R.W. Floyd, and R.L. Rivest. Time bounds for selection. *J. Comput. System Sci.*, 7:448–461, 1973. Fourth Annual ACM Symposium on the Theory of Computing (Denver, Colo., 1972).

9. D. Bryant and V. Moulton. A polynomial time algorithm for constructing the refined Buneman tree. *Applied Mathematics Letters*, 12:51–56, 1999.

10. P. Buneman. The recovery of trees from measures of dissimilarity. In F.R. Hodson, D.G. Kendall, and P. Tautu, editors, *Mathematics in the Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, Edinburgh, 1971.

11. A. Dress. Towards a theory of holistic clustering. In B. Mirkin, F.R. McMorris, F. Roberts, and A. Rzhetsky, editors, *Mathematical Hierarchies and Biology*, DIMACS series in Discete Math. and Theoretical Comp. Science, pages 271–290. AMS, 1997.

12. M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13:155–179, 1995.

13. J.S. Farris, A.G. Kluge, and M.J. Eckart. A numerical approach to phylogenetic systematics. *Systematic Zool.*, 19:172–189, 1970.

14. A.D. Gordon. Hierarchial classification. In P. Arabie, L.J. Hubert, and G. DeSoete, editors, *Clustering and Classification*, pages 65–122. World Scientific, London, 1996.

15. J.C. Gower and G.J.S. Ross. Minimum spanning tree and single linkage cluster analysis. *Applied Statistics*, 18:54–64, 1969.

16. P. Hell and M. Rosenfeld. The complexity of finding generalized paths in tournaments. *Journal of Algorithms*, 4:303–309, 1982.

17. D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast converging method for phylogenetic tree reconstruction. *J. Comp. Biol.*, 6(3/4):369–386, 1999.

18. N. Jardine. Towards a general theory of clustering. *Biometrics*, 25:609–610, 1969.

19. D. W. Matula and F. Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Appl. Math.*, 27(1-2):113–123, 1990. Computational algorithms, operations research and computer science (Burnaby, BC, 1987).

20. V. Moulton and M. Steel. Retractions of finite distance functions onto tree metrics. *Discrete Appl. Math.*, 91(1-3):215–233, 1999.

21. F. Murtagh. Complexities of hierarchic clustering algorithms: state of the art. *CSQ*, 1(2):101–113, 1984.

22. A.F. Parker-Rhodes and R.M. Needham. A reduction method for non-arithmetic data, and its application to thesauric translation. In *Information Processing, Proceedings of the International Conference on Information Processing*, pages 321–325, Paris, 1960. UNESCO.

23. F.J. Rohlf. Algorithm 76: Hierarchical clustering using the minimum spanning tree. *Computer Journal*, 16:93–95, 1973.

24. R.R. Sokal and C.D. Michener. A statistical method for evaluating systematic relationships. *Univ. Kansas Science Bull.*, 38:1409–1438, 1958.

TABLE 1.
A summary of the cluster indices and construction complexities for the family of clustering methods.

Construction	Index	Complexity
Strong clusters	$\iota_w(A) = \min_{uv\|z}\{w(uv\|z) : uv\|z \in r(A)\}$	$O(n^2)$ (sim.[a])
		$O(n^3)$ (iso. wt.[b])
Clean clusters	$\iota_w^c(A) = \text{avmin}(r(A), \|A\| - 1)$	$O(n^3)$ (sim.)
		$O(n^4)$ (iso. wt.)
Stable clusters	$\iota_w^s(A) = \min_{U,V,Z}\{\overline{w}(UV\|Z) : A = U \uplus V, Z \subseteq X - A\}$	NP-hard.

[a] Complexity when input is a similarity function
[b] Complexity when input is an isolation weighting

TABLE 2.
A summary of the split indices and construction complexities for the family of split methods.

Construction	Index	Complexity
Buneman splits	$\mu_\omega(A\|B) = \min_{uv\|yz} \{\omega(uv\|yz) : uv\|yz \in q(A\|B)\}$	$O(n^3)$ (dis.[a])
		$O(n^4)$ (sep. wt. [b])
Ref. Buneman splits	$\text{avmin}(q(A\|B), n-3)$	$O(n^5)$
Clean splits	$\mu^c_\omega(A\|B) = \text{avmin}(q(A\|B), (\|A\|-1)(\|B\|-1))$	$O(n^5)$
Stable splits	$\mu^s_\omega(A\|B) = \min_{U,V,Y,Z} \{\overline{\omega}(UV\|YZ) : A = U \uplus V,\ B = Y \uplus Z\}$	NP-hard.

[a] Complexity when input is a dissimilarity function
[b] Complexity when input is a separation weighting

s	a	b	c	d
a	5	4	2	3
b		5	2	1
c			5	1
d				5

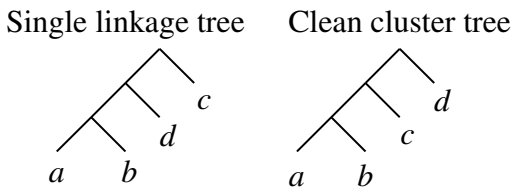

FIG. 1. A similarity function s on four points, its single linkage tree and clean cluster tree.

.

Available online at www.sciencedirect.com
ScienceDirect
Journal of Discrete Algorithms ••• (••••) •••–•••
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

Maximum agreement and compatible supertrees ☆,☆☆

Vincent Berry [a,*], François Nicolas [b]

[a] *Équipe Méthodes et Algorithmes pour la Bioinformatique, LIRMM, CNRS-Université Montpellier 2, France*
[b] *Department of Computer Science, P.O. Box 68 (Gustaf Hällströmin katu 2 b), 00014 University of Helsinki, Finland*

Received 12 January 2006; accepted 3 August 2006

Abstract

Given a set of leaf-labelled trees with identical leaf sets, the MAST problem, respectively MCT problem, consists of finding a largest subset of leaves such that all input trees restricted to these leaves are isomorphic, respectively compatible. In this paper, we propose extensions of these problems to the context of supertree inference, where input trees have non-identical leaf sets. This situation is of particular interest in phylogenetics. The resulting problems are called SMAST and SMCT.

A sufficient condition is given that identifies cases where these problems can be solved by resorting to MAST and MCT as subproblems. This condition is met, for instance, when only two input trees are considered. Then we give algorithms for SMAST and SMCT that benefit from the link with the subtree problems. These algorithms run in time linear to the time needed to solve MAST, respectively MCT, on an instance of the same or smaller size.

It is shown that arbitrary instances of SMAST and SMCT can be turned in polynomial time into instances composed of trees with a bounded number of leaves.

SMAST is shown to be W[2]-hard when the considered parameter is the number of input leaves that have to be removed to obtain the agreement of the input trees. A similar result holds for SMCT. Moreover, the corresponding optimization problems, that is the complements of SMAST and SMCT, cannot be approximated in polynomial time within any constant factor, unless P = NP. These results also hold when the input trees have a bounded number of leaves.

The presented results apply to both collections of rooted and unrooted trees.

Keywords: Algorithms; Trees; Isomorphism and refinement relations; Supertrees; Computational biology

1. Introduction

1.1. Supertree problems and methods

This paper proposes two new methods for building *supertrees*, i.e. trees inferred from other trees. Building supertrees is a problem whose importance increased markedly in the last decade in phylogenetics. Trees considered

☆ Part of these results were briefly covered in a conference paper [V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, in: S.C. Sahinalp, S. Muthukrishnan, U. Dogrusoz (Eds.), Proceedings of the 15th Combinatorial Pattern Matching Symposium (CPM'O4), in: Lecture Notes in Computer Science, vol. 3109, Springer, Berlin, 2004, pp. 205–219].

☆☆ Supported by the *Action Incitative Informatique-Mathématique-Physique en Biologie Moléculaire* [ACI IMP-Bio].

* Corresponding author.
E-mail address: vberry@lirmm.fr (V. Berry).

1570-8667/$ – see front matter
doi:10.1016/j.jda.2006.08.005

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

in this field are called *phylogenies* or *evolutionary trees* because any such tree is an estimation of the evolutionary history of a set of species or sequences (e.g. genes) called *taxa*: the leaves of the tree are each labelled by a current taxon and the branching pattern of the tree describes a speciation scenario leading from ancestral taxa to current ones. In phylogenetics, the major work in progress is the building of the so-called *Tree of Life*, a huge tree interrelating all species of the living realm (see e.g. [38]). Currently, some trees of life are still assembled by hand. According to systematic biologists, the key problem remains to obtain reliable computational methods to assemble several source phylogenies into a single supertree [9].

The input of any supertree building method is a collection of trees with different but overlapping sets of leaves. The output is a tree whose leaf set includes all (or most) species of the input trees and that displays as much as possible of the branching pattern of the input trees on these leaves. The input trees, usually inferred from different datasets, often differ upon the position of some leaves or groups of leaves. Current supertree methods can be divided into two categories depending on the way they handle these conflicts: (i) optimization methods tend to resolve conflicts, i.e. choose one of the proposed scenarios, according to a specified optimization criterion (e.g. [3,32,34]); (ii) consensus methods produce supertrees displaying only the parts of the species' history for which the input trees agree. The drawback of approach (i) is that output supertrees sometimes contain undesirable or unjustified resolutions of conflicts [32]. Approach (ii) has been poorly investigated in the supertree context, in contrast with the many consensus methods available to deal with collection of trees having identical leaf sets. The two known supertree methods of this kind are the pioneering *strict consensus* [21] and *reduced consensus* [37]. Unfortunately, strict consensus usually produces a supertree with a scant amount of information [10,31] and only applies to the rare case of compatible input trees: the trees can differ from one another but not actually conflict [9]. Moreover, the use of reduced consensus is not widespread because a few conflicts only will likely result in a whole set of complementary partial trees as output [37, Section 4], instead of the single synthetic supertree that is sought. Below, we propose alternative methods affiliated to approach (ii) that do not suffer from the drawbacks just mentioned.

1.2. *Extending* MAST *and* MCT *to the supertree context*

Almost all supertree methods proposed so far focus on *clusters* (sets of leaves under internal nodes). This is a problem whenever the input trees contain some "rogue" leaves, i.e. leaves whose position differs greatly from one input tree to the other. Indeed, changing the position of just one leaf in a tree can lead to a completely different set of clusters. Unfortunately, this phenomenon does happen in real supertree instances. Thus, several authors have suggested that an alternative in designing supertree methods would be to focus on leaves individually rather than to consider clusters of leaves [10,21,31]. The rationale for this is that, in a number of cases, removing a few leaves upon whose positions the input trees disagree is sufficient to produce a single informative supertree.

Here we respond to this suggestion by extending to the supertree context a well-known classical consensus problem and one of its variants. Given a set of leaf-labelled trees with identical leaf sets, the MAXIMUM AGREEMENT SUBTREE (MAST) problem consists of finding a subtree homeomorphically included in all input trees and with the largest number of leaves [2,14,17,23,30]. In other words, this involves selecting a largest set of input leaves such that the source trees are isomorphic (i.e. *agree*) when restricted to these leaves. Note that this problem is also considered in various domains other than computational biology. In phylogenetics, when input trees are non-binary, a node with more than two descendants usually represents uncertainty with respect to the branching pattern of its descendants rather than a multi-speciation event. The MAXIMUM COMPATIBLE TREE problem (MCT) is a variant of MAST that takes this into account by seeking a largest set of leaves such that the source trees restricted to these leaves are *compatible* [7,19,24,26] (note that this problem is also called MRST in [26]). Compatibility allows a high-degree node of a source tree to be resolved (split into several nodes) according to the information present in other source trees. Note that this is a weaker constraint than the isomorphism required by MAST, and thus allows inclusion of more input leaves in the output tree.

We call SMAST and SMCT the respective variants of MAST and MCT concerned with supertree inference, i.e. which allow input trees with differing leaf sets. The use of SMCT rather than SMAST can be advocated when the edges of the input trees are associated with confidence values (e.g. bootstrap values in phylogenetic analysis). To obtain a more reliable supertree, edges with insufficient support can be collapsed before the supertree inference is performed, which gives rise to nodes of higher degree in some input trees. In this case, SMCT is more propitious than SMAST for inferring a supertree, as it allows a high degree node of an input tree to be resolved according to the

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

highly supported branching patterns present in other input trees. In other words, highly supported clusters that remain in some input trees will not be contradicted by weakly supported alternatives collapsed in other input trees.

Apart from inferring a partial estimate of the species' history, SMAST and SMCT can also be used to achieve the following goals in phylogenetics:

- To measure the topological similarity between the input trees considered for a supertree reconstruction. The proportion of leaves not conserved in a produced supertree when solving SMAST and SMCT measures the intrinsic difficulty of the particular instance considered for supertree building. This difficulty is currently assessed only through indirect measures, such as the average number of triples or quartets common to two input trees.
- By explicitly indicating leaves upon whose position the input trees conflict, SMAST and SMCT help to identify leaves that may be involved in horizontal transfers of genes and to identify paralogous sequences in the original datasets.
- To increase the accuracy of other supertree building methods. For instance, the popular *matrix representation with parsimony* (MRP) method [3,34] has relatively low accuracy when the input trees overlap moderately and [11] recommend adding to the set of input trees a tree with leaves spanning most input trees, that they call a *seed* tree. Any supertree that is a solution of SMAST or SMCT most likely contains leaves from most, if not all, input trees (see Theorem 3) and, moreover, fully agrees with all of these trees by definition. Thus, it is a good candidate for being a seed tree.

1.3. Related work

We first review known results on the complexity of MAST and MCT. The MAST problem is NP-hard for three rooted trees of unbounded degree [2], while MCT is already NP-hard for two rooted trees of unbounded degree [26]. When k rooted trees with n leaves are given as input, MAST can be solved in $O(n^d + kn^3)$ time provided that the degree of one of the input trees is bounded by d [2,12], and MCT can be solved in $O(2^{2kd}n^k)$ time provided that all input trees have degree bounded by d [19]. MCT is also theoretically solvable in polynomial time provided that solely that the maximum degree of all input trees is bounded [22]. Polynomial-time algorithms with sub-quadratic running times have been obtained for MAST in the special case of two input trees [14,29,30].

MAST and MCT are known to be *fixed-parameter tractable* (*FPT*) in p, the smallest number of leaves to remove from the input set of leaves such that the input trees agree. The latest result being an $O(\min\{3^p kn, 2.27^p + kn^3\})$ time algorithm for the case of rooted trees (considering unrooted trees adds a p factor) [8]. The MAST problem (maximizing the number of leaves in an agreement subtree) is hard to approximate on a bounded number of trees [26] or on trees with a bounded height [20]. The same results hold for MCT [7]. However, the complement problem of MAST (i.e. minimizing the number of leaves to remove so that input trees are isomorphic) can be approximated within a constant ratio in polynomial time in both rooted and unrooted cases [2,7,27]. The same result holds for the complement problem of MCT [7,18].

The extension of MAST to supertree inference has also been considered in [23], and very recently in [28]. However, the "supertrees" considered in [23] (and subsequent papers) have a different meaning from that considered in phylogenetics and here. The work of [28] is independent of the results presented here, but studies an extension of MAST similar to the one we present. [28] give an algorithm for the case of two input rooted trees and present an approximation result that is complementary to the results shown here. However, they neither consider the case of unrooted trees, nor the extension of the MCT problem to the supertree context.

1.4. Results

We show how to extend MAST and MCT in a natural way to obtain the SMAST and SMCT problems on supertrees. We prove that the maximal degree d of input trees does not play any role in solving SMAST and SMCT as any instance of these problems can be reduced to an instance with small bounded degree. This contrasts with MAST and MCT problems, for which polynomial-time algorithms are available for input trees of bounded degree only (in the case of more than two input trees).

We show that any leaf appearing in a single input tree may definitely be included in all supertrees that are solutions of SMAST and SMCT. We give a sufficient condition for SMAST and SMCT to be solved by using MAST and MCT

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

as subproblems. This condition is always fulfilled for collections of only two input trees but also applies to instances including more trees.

We give algorithms that take advantage of the link between these problems and detail when this leads to polynomial cases for SMAST and SMCT. The presented algorithms run in time linear to the algorithm solving MAST and MCT, one of them generalizing the algorithm of [28]. The MERGETREES algorithm we propose also enables computation of the strict consensus supertree of two trees in $O(n)$ time (where n is the total number of input leaves), which improves the $O(n^3)$ bound stated in [21].

In general, SMAST and SMCT are NP-hard as they are equivalent to MAST, respectively MCT, in the case of input trees with identical leaf sets. However, by reduction from HITTING SET, we show that SMAST and SMCT are more difficult than MAST and MCT, as they are W[2]-hard for p (the minimum number of input leaves to remove from input trees to obtain their agreement, respectively compatibility). This holds even when the instance only consists of rooted triples (binary trees with three leaves) or unrooted quartets (trees with four leaves).

This suggests that heuristic algorithms may be required to solve these supertree problems in general. However, no heuristics with a tight approximation ratio can exist for these problems: SMAST and SMCT are hard to approximate (from the results of [20,26] for MAST), and the reduction from HITTING SET is approximation preserving, which proves that no polynomial-time algorithm can approximate within any *constant factor* the complement of the SMAST and SMCT problems (unless P = NP). Note in passing that, compared to the reduction from INDEPENDENT SET/VERTEX COVER given in [28], the reduction given here from HITTING SET leads to tighter results on the parameterized complexity and approximability of the complement of SMAST. Moreover, our result also applies to the complement of SMCT and to the unrooted case. Note that the strong limitations shown here on the approximability of SMAST and SMCT do not impede the existence of approximation algorithms with non-constant ratio. E.g. [28] provides a $(n/\log n)$-approximation algorithm for SMAST on rooted trees.

1.5. Organization of the paper

In the following, Section 2 reviews definitions of MAST and MCT with associated results, then introduces the SMAST and SMCT supertree problems. Section 3 presents algorithms to solve SMAST and SMCT in the particular cases where MAST and MCT can be used as subproblems. A sufficient condition for applying these algorithms is also stated there. Then, Section 4 details how general instances of SMAST and SMCT can be polynomially transformed into instances of trees having a bounded number of leaves (hence also a bounded degree). On the basis of such instances, Section 5 shows the intractability and inapproximability results.

2. Definitions and preliminaries

The trees we consider are *evolutionary trees* (also called *phylogenies*). Such a tree T has its leaf set $L(T)$ in bijection with a label set and is either *rooted* (at a node denoted $\text{root}(T)$), in which case all internal nodes have at least two children each, or *unrooted*, in which case internal nodes have a degree of at least three. In the following, trees are denoted T, respectively R, respectively U, in statements applying to both rooted and unrooted trees, respectively applying only to rooted trees, respectively applying only to unrooted trees. When there is no ambiguity, we identify leaves with their labels. Given a set S, $\text{Card}(S)$ denotes the cardinality of S. In particular, if L is a leaf set, $\text{Card}(L)$ denotes the number of leaves in L. The size $|T|$ of a tree T is the number of its leaves: $|T| = \text{Card}(L(T))$. For a node u in a rooted tree, we denote $S(u)$ the subtree rooted at u (i.e. u and its descendant nodes) and $L(u)$ the leaves of this subtree. See Fig. 2 for an example. The following definitions apply to rooted and unrooted trees.

Definition 1 *(Restriction of a tree).* Given a set L of labels and a tree T, the *restriction* of T to L, denoted $T|L$, is the tree obtained in the following way: take the smallest induced subgraph of T connecting leaves with labels in $L \cap L(T)$, then remove any degree two (non-root) node to make the tree homeomorphically irreducible. If $\mathcal{T}$ is a collection of trees, then define $\mathcal{T}|L := \{T|L\colon\ T \in \mathcal{T}\}$.

See trees U, U' in Fig. 1 for an example. Note that for any tree T and any two label sets L, L', $(T|L)|L' = T|(L \cap L') = (T|L')|L$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

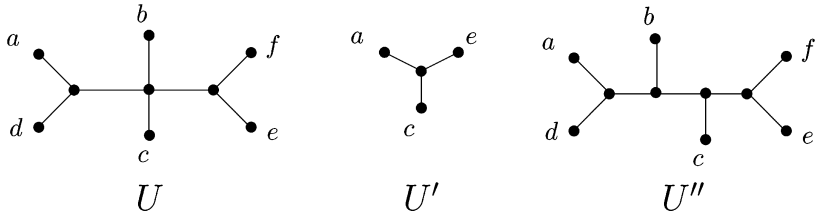

Fig. 1. Three unrooted trees. A tree U, a tree U' such that $U' = U|\{a, c, e\}$ and a tree U'' such that $U'' \trianglerighteq U$.

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

Definition 3 *(Tree refinement).* A tree T *refines* a tree T', and we write $T \trianglerighteq T'$, whenever T can be transformed into T' by collapsing some of its internal edges (*collapsing* an edge means removing it and merging its extremities). See Fig. 1 for an example. More generally, a tree T refines a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$, denoted $T \trianglerighteq \mathcal{T}$, whenever T refines all T_i's in $\mathcal{T}$.

When considering a set of trees with different leaf sets, the preceding definition can be extended [36]:

Definition 4 *(Tree compatibility).* Let T be a tree with leaf set L, let L' be a subset of L and T' be a tree with leaf set L'. We say that T *displays* T' whenever $T|L' \trianglerighteq T'$. Furthermore, a collection $\mathcal{T}$ of trees with different leaf sets is *compatible* if there is a tree T that displays every tree in $\mathcal{T}$. In that case, T is said to *display* $\mathcal{T}$.

Isomorphism and compatibility issues between rooted trees can also be expressed in terms of ancestor relationships. Given two nodes u and v in a rooted tree R, $u < v$ means that u is a proper ancestor of v and $u \leqslant v$ means that u is an ancestor of v, i.e. that u is either a proper ancestor of v, or v itself. The *least common ancestor* (or lca) of a set of leaves $L \subseteq L(R)$ is the unique node u such that $u \leqslant \ell$ for all $\ell \in L$, and $v < u$ for any other node v that is also an ancestor of every leaf in L. The lca of L in R is denoted $\mathrm{lca}_R(L)$. More particularly, the lca of any pair $\{\ell, \ell'\} \subseteq L(R)$ is denoted $\mathrm{lca}_R(\ell, \ell')$.

The following statements are directly derived from the definitions given previously and are implicitly or explicitly used in a number of works (for instance [18,33]).

Observation 1. *Let R and R' be two rooted trees on the same leaf set L. The following two statements are equivalent*:

(i) *R and R' are isomorphic*;
(ii) *$\forall \ell, \ell', \ell'' \in L$, the two following equations hold*

$$\mathrm{lca}_R(\ell, \ell') < \mathrm{lca}_R(\ell, \ell'') \iff \mathrm{lca}_{R'}(\ell, \ell') < \mathrm{lca}_{R'}(\ell, \ell'') \quad \textit{and} \tag{1}$$

$$\mathrm{lca}_R(\ell, \ell') = \mathrm{lca}_R(\ell, \ell'') \iff \mathrm{lca}_{R'}(\ell, \ell') = \mathrm{lca}_{R'}(\ell, \ell''). \tag{2}$$

Moreover, the following two statements are equivalent:

(iii) *R refines R'*;
(iv) *$\forall \ell, \ell', \ell'' \in L$, the following holds*:

$$\mathrm{lca}_{R'}(\ell, \ell') < \mathrm{lca}_{R'}(\ell, \ell'') \Rightarrow \mathrm{lca}_R(\ell, \ell') < \mathrm{lca}_R(\ell, \ell''). \tag{3}$$

2.1. Agreement problems for trees with identical leaf sets

The well-known MAST problem is defined as follows:

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

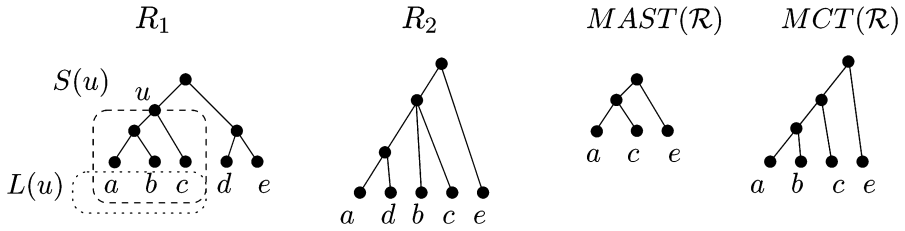

Fig. 2. A collection $\mathcal{R} = \{R_1, R_2\}$, one of the $MAST(\mathcal{R})$ trees and the $MCT(\mathcal{R})$ tree. $S(u)$ denotes the subtree induced by a node u and $L(u)$ the corresponding set of leaves.

Definition 5 *(MAST problem).* Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees with identical leaf sets L, an *agreement subtree* of $\mathcal{T}$ is any tree T with leaves in L such that $\forall T_i \in \mathcal{T}$, $T = T_i|L(T)$. The MAXIMUM AGREEMENT SUBTREE problem (MAST) consists in finding an agreement subtree of $\mathcal{T}$ with the largest number of leaves. Such a tree is denoted $MAST(\mathcal{T})$.

The MCT problem is a variant of MAST introduced in phylogenetics to deal with cases where high-degree nodes represent uncertainty with respect to the relative branching of their child subtrees.

Definition 6 *(MCT problem).* Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of input trees with identical leaf sets L, a tree T with leaves in L is said to be *compatible with* $\mathcal{T}$ if and only if $\forall T_i \in \mathcal{T}$, $T \trianglerighteq T_i|L(T)$. The MAXIMUM COMPATIBLE TREE problem (MCT) consists in finding a tree compatible with $\mathcal{T}$ having the largest number of leaves. Such a tree is denoted $MCT(\mathcal{T})$. If there is a tree T compatible with $\mathcal{T}$ such that $L(T) = L$, then the collection $\mathcal{T}$ is said to be *compatible*.

Note that an evolutionary tree T properly refining another tree T', agrees with the entire evolutionary history of T', while containing additional history absent from T': at least one high degree node of T' is replaced in T by several nodes, hence T specifies more speciation events than T'. Fig. 2 shows examples of trees $MAST(\mathcal{T})$ and $MCT(\mathcal{T})$ for a collection $\mathcal{T}$ of two rooted trees. Note that $\forall \mathcal{T}$, $|MCT(\mathcal{T})| \geqslant |MAST(\mathcal{T})|$ and that MCT is equivalent to MAST when all input trees are binary. Note also that some instances of the MAST and MCT problems have several optimum solutions.

2.2. Extending agreement problems to the supertree context

We now consider the case of supertree inference, where input trees are allowed to have different sets of leaves. We first show how to extend MAST and MCT to this context. Then we distinguish different kinds of leaves that appear in the input trees, depending on the overlap of these trees. Without loss of generality, the rest of the paper assumes that any input tree shares at least two leaves with other input trees.

Definition 7 *(Leaf set of a collection).* Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees, we denote $L(\mathcal{T}) := \bigcup_{T_i \in \mathcal{T}} L(T_i)$ the set of all leaves appearing in at least one tree of $\mathcal{T}$.

Definition 8 *(SMAST problem).* Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees, an *agreement supertree* of $\mathcal{T}$ is a tree T with $L(T) \subseteq L(\mathcal{T})$ such that $\forall T_i \in \mathcal{T}$, $T|L(T_i) = T_i|L(T)$. An agreement supertree of $\mathcal{T}$ that is of maximum size is called a *maximum agreement supertree* of $\mathcal{T}$ and is denoted $SMAST(\mathcal{T})$. The corresponding optimization problem is stated as follows:

Name: MAXIMUM AGREEMENT SUPERTREE (SMAST)
Instance: A finite collection $\mathcal{T}$ of trees (all rooted or all unrooted).
Solution: An agreement supertree T of $\mathcal{T}$.
Measure: $|T|$, to be maximized.

In a similar way, we define:

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

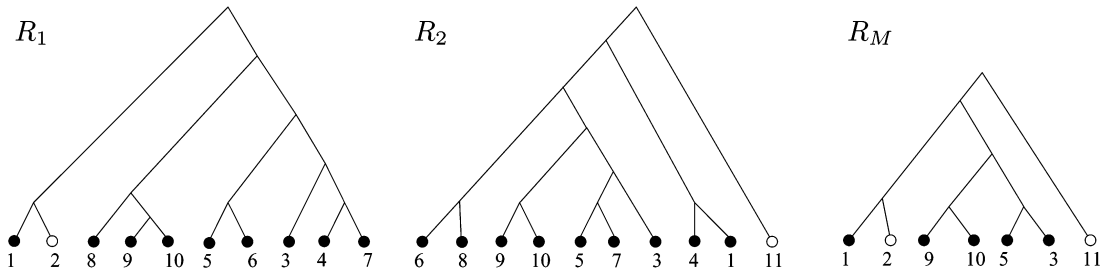

Fig. 3. A collection $\mathcal{R} = \{R_1, R_2\}$ of two source trees on tomatoes taken from [4] and a supertree R_M. In this example, the supertree both represents a $SMAST(\mathcal{R})$ and a $SMCT(\mathcal{R})$. Leaves appearing in only one source tree are displayed in white. Correspondence between numbers and species: 1—*L. lycopersicoides*, 2—*L. juglandifolium*, 3 – *L. peruvianum*, 4—*L. chilense*, 5—*L. pennellii*, 6—*L. hirsutum*, 7—*L. chmielewskii*, 8—*L. esculentum*, 9—*L. pimpinellifolium*, 10—*L. cheesmanii*, 11—*L. rickii*.

Definition 9 *(SMCT problem).* Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees, a *supertree compatible with* $\mathcal{T}$ is a tree T with $L(T) \subseteq L(\mathcal{T})$ such that $\forall T_i \in \mathcal{T}$, $T|L(T_i) \trianglerighteq T_i|L(T)$. A supertree compatible with $\mathcal{T}$ that is of maximum size is called a *maximum compatible supertree* of $\mathcal{T}$ and is denoted $SMCT(\mathcal{T})$. The corresponding optimization problem is as follows:

Name: MAXIMUM COMPATIBLE SUPERTREE (SMCT)
Instance: A finite collection $\mathcal{T}$ of trees (all rooted or all unrooted).
Solution: A supertree T compatible with $\mathcal{T}$.
Measure: $|T|$, to be maximized.

Fig. 3 shows a collection $\mathcal{R}$ with to source trees of tomato species (*Lycopersicon*) and a supertree, that both is a $SMAST(\mathcal{R})$ and a $SMCT(\mathcal{R})$. Fig. 7 shows an example where these two supertrees differ. The two problems stated above are natural extensions of the problems defined in Section 2.1. More precisely, SMAST, respectively SMCT, is equivalent to MAST, respectively MCT, when all input trees have the same set of leaves.

Remark 1. Let $\mathcal{T}$ be a collection of trees. Any restriction of an agreement supertree of $\mathcal{T}$ is also an agreement supertree of $\mathcal{T}$ and any restriction of a supertree compatible with $\mathcal{T}$ is also a supertree compatible with $\mathcal{T}$.

Hence, $MAST(\mathcal{T})$ and $SMCT(\mathcal{T})$ can potentially contain less leaves than some trees in $\mathcal{T}$.

Let $\mathcal{T}$ be a collection of trees with identical leaf set L. Given any subset $L' \subseteq L$, there can be only one agreement *subtree* of $\mathcal{T}$ with leaf set L'. This contrasts with what can happen for agreement *supertrees*, due to the lack of cross information between source trees:

Remark 2. Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees and a subset $L \subseteq L(\mathcal{T})$, there may be more than one agreement supertree, respectively compatible supertree, of $\mathcal{T}$ with leaf set L.

For instance, consider the collection $\mathcal{R} = \{R_1, R_2\}$ where $R_1 := ((a, c), b)$ and $R_2 = ((a, d), b)$, in parenthetical notation (i.e. Newick format). Any tree in the set $\{(((a, c), d), b), (((a, d), c), b), ((a, (c, d)), b), ((a, c, d), b)\}$ is a $SMAST(\mathcal{R})$ or a $SMCT(\mathcal{R})$.

Definition 10 *(Types of leaves).* Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of trees. Leaves in $L(\mathcal{T})$ can be partitioned in three subsets:

(1) leaves appearing in every tree of $\mathcal{T}$. We note $L_\cap(\mathcal{T}) := \bigcap_{T_i \in \mathcal{T}} L(T_i)$ this subset of leaves;
(2) leaves appearing in several but not all trees of $\mathcal{T}$. This subset of leaves is denoted $L_\Delta(\mathcal{T})$;
(3) leaves specific to a tree of $\mathcal{T}$, i.e. leaves appearing in a single tree of $\mathcal{T}$. This subset of leaves is denoted $L_S(\mathcal{T})$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

3. Computing SMAST and SMCT

We first study, in Section 3.1, the particular case of a collection of only two trees with different leaf sets but such that one refines (respectively, is equal to) the other when restricted to common leaves. In that case, it is always possible to efficiently produce a tree that displays the collection, i.e. a maximum compatible supertree (respectively, a maximum agreement supertree). We provide a linear-time algorithm that fulfills this purpose.

Then, Section 3.2 shows that for a collection containing an arbitrary number of trees, any maximum agreement supertree or maximum compatible supertree includes all specific leaves, i.e. leaves that appear in a single tree of the collection. Based on this property, Section 3.3 shows cases where the MAST and MCT problems can be used to solve SMAST and SMCT problems, respectively, describing appropriate algorithms. The link between subtree problems and supertree problems induces polynomial cases for the latter, as listed in Section 3.4.

3.1. Merging two rooted trees in linear time

Let $\mathcal{R} = \{R_I, R_A\}$ be a compatible collection of two rooted trees with $L(R_I) \neq L(R_A)$, such that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$. In other words, R_A (loosely or strictly) refines R_I when they are both restricted to their common leaves. This section describes an algorithm called MERGETREES that returns a tree R displaying $\mathcal{R}$. By definition, R contains all leaves of the two trees and R is a maximum compatible supertree of $\mathcal{R}$. Moreover, when $R_A|L(R_I)$ loosely refines $R_I|L(R_A)$, i.e. when both restricted trees are isomorphic, then the tree R output by MERGETREES is a maximum agreement supertree of $\mathcal{R}$. To compute a *SMCT*($\mathcal{R}$) or *SMAST*($\mathcal{R}$) of a collection $\mathcal{R}$ of more than two trees, repeated calls to MERGETREES operating on two trees will be used, as described in Section 3.3. For the rest of Section 3.1, trees are considered to be rooted.

Definition 11 *(Specific subtree).* Let $\mathcal{R}$ be a collection of trees and $R_I \in \mathcal{R}$. A *specific subtree* of R_I is any maximal tree of the form $S(v_i)$, where v_i is a node of R_I such that $L(v_i) \cap (L(\mathcal{R}) - L(R_I)) = \emptyset$. Here, *maximal* means that if p_i is the parent node of v_i, then $L(p_i) \cap (L(\mathcal{R}) - L(R_I)) \neq \emptyset$. A leaf in a specific subtree is called a *specific leaf*.

For instance, consider the collection $\{R_I, R_A\}$ displayed in Fig. 4. R_I hosts two specific subtrees: the leaf $\{y_1\}$ and the subtree rooted at node v_i''. The leaf $\{x\}$ is the only specific subtree of R_A. In a collection $\mathcal{R} = \{R_A, R_I\}$ of two trees, leaves are either specific to R_I, specific to R_A, or common to both trees. To obtain the desired tree R, MERGETREES proceeds by grafting specific subtrees of R_I into R_A. In a way, its goal is similar to the grafting step of the well-known algorithm of Gordon for computing a strict consensus supertree [21]. However, Gordon's algorithm attaches, one by one, specific *leaves* of the input trees to a "backbone" tree, while the algorithm detailed here proceeds by grafting each time a whole specific *subtree*. Using this idea and two simple data structures, we can achieve linear running time when the grafting step of [21] runs in cubic time. As the tree R output by MERGETREES has to display R_I, the specific subtrees of R_I have to be grafted in R_A so as to respect the ancestor-descendant constraints of R_I between lcas of leaves (see Observation 1). Sometimes there is a unique place where a subtree can be grafted in order to respect these relationships, and sometimes there can be several. However, a correspondence between some nodes of R_I and some nodes of R_A can be maintained such that a correct place is always easily identified. This correspondence is explained by the fact that, when restricted to common leaves $L_\cap(\{R_A, R_I\})$, R_I is refined by R_A. This ensures that for any node $v_i \in R_I|L(R_A)$ there is a unique node $v_a \in R_A|L(R_I)$, such that $L(v_i) = L(v_a)$. This correspondence

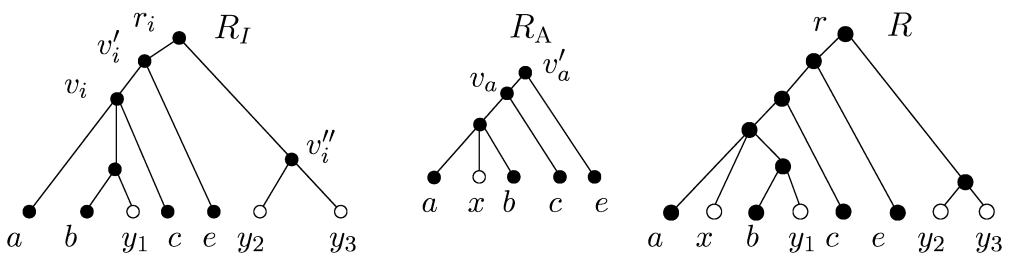

Fig. 4. Two rooted trees R_I and R_A with overlapping sets of leaves such that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$, and the tree R returned by the call MERGETREES(R_I, R_A). Specific leaves are indicated by white circles.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

between nodes of the restricted trees $R_A|L(R_I)$ and $R_I|L(R_A)$ can be translated as a correspondence between nodes of the complete input trees R_I and R_A:

Definition 12. Let R and R' be two trees. To any node v in R such that $v := \mathrm{lca}_R(S)$ with $S \subseteq L_\cap(\{R, R'\})$, $\mathrm{Card}(S) \geqslant 1$, we associate an *anchor node* v' in R', defined as $v' := \mathrm{lca}_{R'}(S)$. Nodes of R with an anchor in R' are called *anchored nodes*.

Note that the previous definition allows both internal nodes and leaf nodes of R to have an anchor in R'. However, note that some nodes of R can have no anchor in R' and that some nodes of R' are not anchors of any node in R.

Remark 3. Let R_I and R_A be two rooted trees such that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$ and $v_i \in R_I$ be a node with an anchor node $v_a \in R_A$. We have $L(v_i) \cap L_\cap(\{R_A, R_I\}) = L(v_a) \cap L_\cap(\{R_A, R_I\})$.

If the root r_i of R_I has no anchor in R_A according to the previous definition, then algorithm MERGETREES artificially anchors it to a node in R_A to enable grafting of specific subtrees hanging from r_i or between r_i and the highest anchored node in R_I. This artificial anchor is set in the following way: if the root r_a of R_A is not anchored to any node in R_I then it is used as the anchor for r_i. Otherwise, the anchor of r_i is set at a new node that is added as a parent of r_a (the algorithm will graft some specific subtrees to this new node that will hence not remain of degree 1). In Fig. 4, leaves $\{a, b, c, e\}$ of R_I are respectively anchored at similarly labelled leaves in R_I; the internal nodes v_i, respectively v'_i, is anchored at v_a, respectively v'_a. The root r_i of R_I is artificially anchored to a node added as parent of the original root v'_a of R_A (not shown in the figure).

The position of any specific subtree of R_I can be considered w.r.t. anchored nodes: either (i) the subtree is hanging from an anchored node, or (ii) it is hanging from a node that is in between two anchored nodes. Thanks to the corresponding anchors between R_I and R_A, any specific subtree of R_I can be grafted in R_A so as to respect ancestor/descendant relationships needed for the produced tree to display R_I (see Observation 1). Algorithm 1 details in pseudo-code how this is done given two preprocessed data structures:

- Anchor that associates nodes of R_I to their respective anchors in R_A.
- SpecificChildren that associates to each node of R_I the list of its children that are roots of specific subtrees.

Input: Two rooted trees R_I and R_A with overlapping sets of leaves and such that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$.
Result: A tree R on $L(R_A) \cup L(R_I)$ such that $R|L(R_A) = R_A$ and $R|L(R_I) \trianglerighteq R_I|L(R)$.
$R \leftarrow R_A$
1 Compute the Anchors of R_I in R and the SpecificChildren for R and R_I
***(i)* Graft specific subtrees to anchored nodes of R**
2 **for** *each node* $v \in R_I$ *such that* Anchor$(v) \neq \emptyset$ **do**
  for *each node* $c \in$ SpecificChildren(v) **do**
    Add a copy of $S(c)$ as a new child of Anchor(v) in R
***(ii)* Graft specific subtrees on the path between two anchored nodes of R**
3 **for** *each non-root node* $v \in R_I$ *such that* Anchor$(v) \neq \emptyset$ **do**
  $v_i \leftarrow$ parent node of v ; $v_a \leftarrow$ Anchor (v)
4 **while** Anchor(v_i) *is* $\emptyset$ **do**
    Insert a new node v_{new} between v_a and its parent node in R
5   **for** *each* $c \in$ SpecificChildren(v_i) **do**
      Add a copy of $S(c)$ as a new child subtree of v_{new}
    $v_a \leftarrow v_{\text{new}}$; $v_i \leftarrow$ parent node of v_i
return R

Algorithm 1. MERGETREES(R_I, R_A).

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

For an example of an execution of MERGETREES, consider the trees R_I and R_A displayed in Fig. 4. The algorithm progressively builds tree R, starting from a copy of the tree R_A. Then anchors between R_I and R are computed. During this process, the initial root of R (corresponding to the node labelled v'_a in R_A) is given a parent node (called r) to serve as an artificial anchor for $r_i \in R_I$. During step (i) of the algorithm (i.e. loop in line 2), a copy of the specific subtree $S(v''_i)$ of R_I is grafted to the anchor r of r_i in R. Then during step (ii) (i.e. loop in line 3), the specific leaf-subtree y_1 hanging in R_I from the parent of the anchored node b is grafted in R to a new node inserted between the leaf labelled b and its parent. Fig. 4 shows the resulting tree R.

Theorem 1. *Given a collection $\mathcal{R} = \{R_I, R_A\}$ of two rooted trees such that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$, the algorithm* MERGETREES*(R_I, R_A) returns a tree R such that $L(R) = L(\mathcal{R})$ and such that R is a SMCT($\mathcal{R}$). In the particular case where $R_A|L(R_I) = R_I|L(R_A)$, then R is a SMAST($\mathcal{R}$).*

Proof. First, it is easy to see that $L(R) = L(R_I) \cup L(R_A)$: R is initially set at R_A and copies of all specific subtrees of R_I, i.e. containing all leaves in $L(\mathcal{R}) - L(R_A)$, are then grafted into R. Hence, if R displays the two input trees, then it is a *SMCT*($\mathcal{R}$), because there is no tree larger than R with leaves in $L(\mathcal{R})$. The output tree R displays R_A, because R is initially set at R_A, and the only modifications made to this tree are additions of subtrees containing leaves not belonging to R_A, i.e. not changing $R|L(R_A)$. It remains to be proven that R displays R_I. To that aim, we prove that $R|L(R_I) \trianglerighteq R_I|L(R)$ because, together with $L(R_I) \subseteq L(R)$, this proves that R displays R_I. $R|L(R_I) \trianglerighteq R_I|L(R)$ is proven by induction on the number of grafts performed by the algorithm. The initial step of the induction holds as, before the first graft, $R = R_A$, and we know by assumption that $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$. Now suppose the result holds for the first $g \geqslant 0$ grafts and that a $(g+1)$th specific subtree S_p of R_I is grafted into R, and for the needs of the proof, let R' be the resulting tree. We have to prove that $R'|L(R_I) \trianglerighteq R_I|L(R')$ which, by Observation 1, is equivalent to showing that for all ℓ, ℓ', ℓ'' in $L(R') \cap L(R_I)$ the following holds:

$$\mathrm{lca}_{R_I}(\ell, \ell') < \mathrm{lca}_{R_I}(\ell, \ell'') \Rightarrow \mathrm{lca}_{R'}(\ell, \ell') < \mathrm{lca}_{R'}(\ell, \ell''). \tag{4}$$

There are several cases depending on the number of these leaves already present in R before S_p is grafted:

(1) $\mathrm{Card}(\{\ell, \ell', \ell''\} \cap L(R)) = 0$: then the three leaves belong to $S_p \in R_I$ and, since an exact copy of S_p is grafted into R, the relationships between lcas of leaves in $L(S_p)$ are reproduced in R' as they are in R_I, hence (4) holds.
(2) $\mathrm{Card}(\{\ell, \ell', \ell''\} \cap L(R)) = 3$: this means that the three leaves belong to R before the grafting of S_p, hence (4) holds by induction hypothesis since grafting a subtree does not alter lca relationships between already present leaves in R.
(3) $\mathrm{Card}(\{\ell, \ell', \ell''\} \cap L(R)) = 1$: w.l.o.g. suppose $\ell \in L(R)$, that is $\ell \notin L(S_p)$, hence $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell, \ell'') < \mathrm{lca}_{R_I}(\ell', \ell'')$. As R' is obtained by adding a copy of S_p (containing ℓ' and ℓ'') by a new edge (v, v') as a new child subtree of a node $v \in R$, this means that $\mathrm{lca}_{R'}(\ell, \ell') = \mathrm{lca}_{R'}(\ell, \ell'') \leqslant v < v' \leqslant \mathrm{lca}_{R'}(\ell', \ell'')$, hence (4) holds.
(4) $\mathrm{Card}(\{\ell, \ell', \ell''\} \cap L(R)) = 2$: numerous but simple sub-cases arise here. For the sake of readability, this part of the proof is presented in Appendix B.

In the particular case where $R_A|L(R_I) = R_I|L(R_A)$, we also have to show that the equivalent of Eq. (1) of Observation 1 holds for R' and R_I. This proof strictly follows the proof given above for (4), and is thus omitted. □

Theorem 2. *Algorithm* MERGETREES*(R_I, R_A) runs in $O(n)$ time, where $n = \mathrm{Card}(L(R_I) \cup L(R_A))$.*

Proof. We first detail the cost of preprocessing the data structures used by the algorithm (line 1). To initialize the SpecificChildren data structure, a simple $O(n)$ search of each tree R_I and R_A enables us to know which leaves of each tree are specific. Then an $O(n)$ postorder search of each tree enables us to identify the children of each node that are specific. To initialize the Anchor data structure, leaf-nodes of R_I with a label in $L_\cap(\{R_A, R_I\})$ are directly anchored at nodes of R_I sharing the same label. Then lca relationships are preprocessed in R_I and R_A in $O(n)$ time [25]. Let $\mathcal{O}$ be the left-right order in which the leaves of $L_\cap(\{R_A, R_I\})$ appear in R_I. The $O(n)$ pairs (ℓ_i, ℓ_{i+1}) of successive leaves in $\mathcal{O}$ are then considered. For each of these pairs, a single query for $v_i := \mathrm{lca}_{R_I}(\ell_i, \ell_{i+1})$ and for $v_a := \mathrm{lca}_{R_A}(\ell_i, \ell_{i+1})$ is performed, each time costing only $O(1)$ thanks to the preprocessing step. Only these pairs of leaves have to be

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

considered to span all internal nodes v_i of R_I for which an anchor has to be determined, and to obtain the anchor v_a for each of them (see Appendix A for more details). Hence, initializing the Anchor data structure costs $O(n)$ time.

Step (i) of the MERGETREES algorithm is performed by a recursive search of the tree R_I, during which $O(n)$ nodes v_i are considered. Knowing whether a given node v_i has an anchor in R_A is $O(1)$ time, thanks to the Anchor data structure. If so, knowing each specific child of v_i is also $O(1)$ time thanks to the SpecificChildren data structure. Note that each tree contains $O(n)$ specific children. For each specific child c of v_i, a copy of $S(c)$ is grafted under Anchor(v_i), which costs a time proportional to the size of this subtree, $O(|L(c)|)$. Since non-intersecting subtrees $S(c)$ are considered over all examined nodes v_i, the total size of grafted subtrees is bounded by the number of nodes in the tree, i.e. by $O(n)$, which is then the cost of step (i).

Step (ii) is performed by a recursive postorder search of R_I. Every time an anchored node v is met, the edges on the path from v_i to its closest ascendant that is also anchored are explored (note that $O(n)$ such edges exist in R_I). During this upward walk, each time a non-anchored node v_i is met, copies of specific subtrees hanging from v_i are grafted into R_A to a place identified in $O(1)$ (nodes v_a and v_{new}). Specific subtrees $S(c)$ to be grafted are each identified in $O(1)$ and grafted in $O(|L(c)|)$. This costs $\sum_c |L(c)| \in O(n)$ total time. Hence, step (ii) also costs $O(n)$ time. □

Corollary 1. *The strict consensus supertree [21] of two trees containing n leaves in total can be computed in* $O(n)$ *time.*

Proof. Computing the strict consensus *supertree* T of a collection $\mathcal{T} = \{T_1, T_2\}$ of two trees involves four steps:

(1) computing the restrictions T_1' and T_2' of the input trees to $L_\cap(\mathcal{T})$;
(2) computing the strict consensus *tree* T' of the trees T_1', T_2' (having the same set of leaves);
(3) collapsing suitable edges (by joining their two extremities) in T_1 and T_2 such that $T_1|L_\cap(\mathcal{T}) = T'$ and $T_2|L_\cap(\mathcal{T}) = T'$;
(4) T is obtained by grafting specific subtrees of the modified T_1 and T_2 in tree T', then collapsing edges in the parts of T' where both specific subtrees of T_1 and T_2 have been inserted.

Step 1 is clearly done in $O(n)$ by traversals of the trees. Several $O(n)$ algorithms are known for Step 2 (e.g. [6]). Step 3 is done by first anchoring nodes of T_1' and T_2' in T', then jointly traversing T_1 and T' and similarly for T_2 and T', hence requiring $O(n)$ time. Step 4 first performs two calls to MERGETREES, obtaining a tree T in $O(n)$ time. When keeping track of which input tree the specific subtrees originate from, a single traversal of T (requiring $O(n)$ time) is then enough to decide which specific subtrees have to be collapsed. Collapsing a subtree is linear in the number of its edges, i.e. all collapsing operations in T require $O(n)$ total time. □

When considering collections of more than two trees, the MERGETREES algorithm will be used several times to attach specific subtrees from the different input trees to an initial backbone tree. The order in which input trees are processed does not change the set of leaves of the produced supertree. However, the shape of the supertree can vary depending on this order. This is not relevant to solve SMAST and SMCT, but can lead the supertree to possess some edges that can be considered as arbitrary from a phylogenetic standpoint. However, such edges can easily be detected and collapsed through known algorithms [35].

3.2. The inclusion of specific leaves

The following result states that all specific leaves of a collection are systematically included in any maximum agreement supertree or maximum compatible supertree of the collection. Intuitively, this result is not surprising since the information for positioning a specific leaf comes only from one input tree. Thus, no disagreement or incompatibility arises by positioning the leaf according to this input tree. Nonetheless, the proof requires handling a certain number of restrictions of trees and intersection of leaf sets.

Theorem 3. *Let* $\mathcal{R}$ *be a collection of rooted trees with overlapping sets of leaves. All specific leaves of* $\mathcal{R}$ *appear in any SMAST*$(\mathcal{R})$ *and in any SMCT*$(\mathcal{R})$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

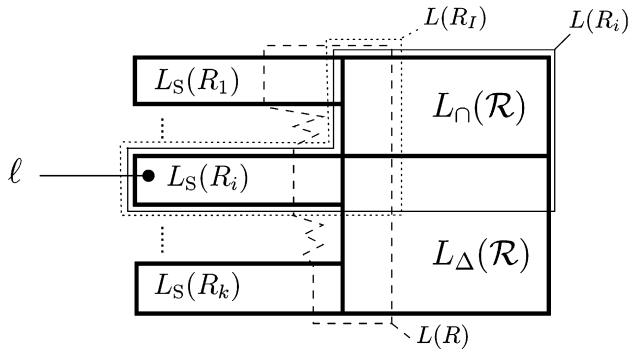

Fig. 5. The set of leaves $L(\mathcal{R})$ of a tree collection $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$, decomposed into the three sets $L_\cap(\mathcal{R}), L_\Delta(\mathcal{R})$ and $L_S(\mathcal{R}) = \bigcup_{R_i \in \mathcal{R}} L_S(R_i)$, all displayed in bold lines. The figure also displays the leaf set $L(R_i)$ of a tree $R_i \in \mathcal{R}$, in plain thin lines, and leaf sets $L(R_I)$, respectively $L(R)$, in dotted, respectively dashed lines, mentioned in the proof of Theorem 3, where R is a $SMCT(\mathcal{R})$ assumed to not cover all leaves of $L_S(\mathcal{R})$ for the sake of contradiction.

Proof. The proof is given for the SMCT problem. The proof for SMAST is quite similar. Let R be a $SMCT(\mathcal{R})$. The proof proceeds by supposing that there is a specific leaf $\ell \in L_S(\mathcal{R})$ such that $\ell \notin L(R)$ and shows that a single run of algorithm MERGETREES gives a tree containing ℓ that is both a $SMCT(\mathcal{R})$ and larger than R, which is in contradiction with the maximality of R. Several leaf sets involved in the proof are exemplified in Fig. 5.

Let R_i be the tree of $\mathcal{R}$ from which ℓ originates and let $R_I := R_i|(L(R) \cup L_S(R_i))$. Basic set operations show that

$$L(R_I) \cap L(R) = L(R_i) \cap L(R). \tag{5}$$

By definition of R, $R|L(R_i) \trianglerighteq R_i|L(R)$, i.e.

$$R|\big(L(R) \cap L(R_i)\big) \trianglerighteq R_i|\big(L(R_i) \cap L(R)\big), \tag{6}$$

$$R|L(R_I) \trianglerighteq R_i|\big(L(R_I) \cap L(R)\big), \tag{7}$$

$$R|L(R_I) \trianglerighteq R_I|L(R), \tag{8}$$

where (7) results from the use of (5) on both sides of (6), and (8) derives from $R_i|L(R_I) = R_I$.

Let R' be the tree returned by the call to MERGETREES (R_I, R). From (8), Theorem 1 applies and gives

$$L(R') = L(R) \cup L(R_I). \tag{9}$$

As, $\ell \in L(R_I) - L(R)$, we deduce from (9) that

$$\mathrm{Card}\big(L(R')\big) > \mathrm{Card}\big(L(R)\big). \tag{10}$$

Also, by definition of R_I, we have $L(R_I) = L(R_i) \cap (L(R) \cup L_S(R_i)) = (L(R_i) \cap L(R)) \cup L_S(R_i)$. Combined with (9), we then have

$$L(R') = L(R) \cup L_S(R_i). \tag{11}$$

From (8) and Theorem 1, we also know that R' is a $SMCT(\{R_I, R\})$, thus

$$R'|L(R_I) \trianglerighteq R_I|L(R'). \tag{12}$$

From (9) and the definition of R_I, basic set operations show that $L(R') \cap L(R_I) = L(R') \cap L(R_i)$, thus the left term of (12) can be rewritten as $R'|L(R_i)$; its right term can be rewritten as $R_i|L(R')$ (replacing R_I by its definition and then using (9)), leading to

$$R'|L(R_i) \trianglerighteq R_i|L(R'). \tag{13}$$

Moreover, from Theorem 1, $R'|L(R) \trianglerighteq R|L(R')$. Since, $L(R) \subseteq L(R')$, this means that

$$R'|L(R) \trianglerighteq R. \tag{14}$$

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Now consider any $R_j \in \mathcal{R}$ such that $R_j \neq R_i$. From $L_S(R_i) \cap L(R_j) = \emptyset$ and (11) we obtain

$$L(R_j) \cap L(R) = L(R_j) \cap L(R'), \tag{15}$$

from which we deduce $(R'|L(R))|L(R_j) = R'|(L(R) \cap L(R_j)) = R'|L(R_j)$. Thus, restricting both terms of (14) to $L(R_j)$, we obtain $R'|L(R_j) \trianglerighteq R|L(R_j)$. Combining this with $R|L(R_j) \trianglerighteq R_j|L(R)$ (which holds by definition of R) shows by transitivity that $R'|L(R_j) \trianglerighteq R_j|L(R)$. This can be rewritten as

$$R'|L(R_j) \trianglerighteq R_j|L(R') \tag{16}$$

since (15) also implies $R_j|L(R) = R_j|L(R')$.

Eq. (13) for R_i and Eq. (16) for all $R_j \in \mathcal{R}$, $R_j \neq R_i$, show that R' is a supertree compatible with $\mathcal{R}$. Moreover, from (10), R' contains more leaves than $R := SMCT(\mathcal{R})$, a contradiction. □

3.3. Using MAST and MCT as subproblems

In the general case, it is not possible to solve SMAST, respectively SMCT, by considering MAST, respectively MCT, as a subproblem. For instance, Fig. 6 shows a collection $\mathcal{R}$ with only three rooted trees, where the trees $SMAST(\mathcal{R})$ and $SMCT(\mathcal{R})$ do not include $MAST(\mathcal{R})$ and $MCT(\mathcal{R})$ as restrictions. However, in the particular case where every leaf of the collection belongs either to a single tree or to all trees of the collection, the connection between subtree and supertree problems can be exploited. See algorithm BUILDSMCT (Algorithm 2) to solve SMCT. The algorithm proceeds from a maximum compatible tree of the input trees restricted to common leaves. Then, specific subtrees of each original input tree are added by successive calls to the MERGETREES algorithm.

To prove the correctness of BUILDSMCT, we first need to establish the three following invariants:

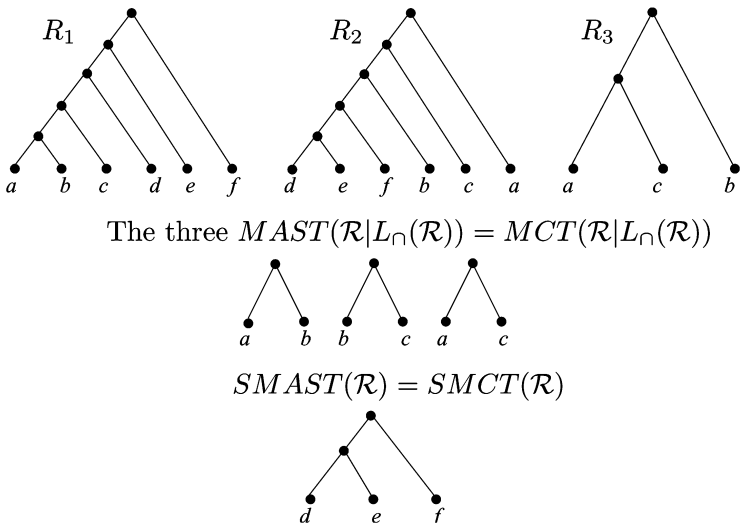

Fig. 6. A collection $\mathcal{R} = \{R_1, R_2, R_3\}$ of rooted input trees for which the trees $MAST(\mathcal{R})$ and $MCT(\mathcal{R})$ can not be used as backbones of $SMAST(\mathcal{R})$ and $SMCT(\mathcal{R})$.

Input: A collection $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of rooted trees such that $L_\Delta(\mathcal{R}) = \emptyset$.
Result: A tree $SMCT(\mathcal{R})$.
1 $R_M^0 \leftarrow MCT(\mathcal{R}|L_\cap(\mathcal{R}))$
for i *in* $1 \rightarrow k$ **do**
2 $\quad R_M^i \leftarrow$ MERGETREES $\left(R_i \middle| (L(R_M^0) \cup L_S(R_i)), R_M^{i-1}\right)$
return R_M^k.

Algorithm 2. BUILDSMCT ($\mathcal{R}$).

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Lemma 1. *Given a collection* $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ *such that* $L_\Delta(\mathcal{R}) = \emptyset$*, the following statements hold at each iteration* i $(1 \leqslant i \leqslant k)$ *of the loop of algorithm* BUILDSMCT:

(A) *let* $R_I := R_i|(L(R_M^{i-1}) \cup L_S(R_i))$ *and* $R_A := R_M^{i-1}$ *be the trees given as input to* MERGETREES *in line* 2, $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$ *holds*;
(B) $L(R_M^i) = L(R_M^0) \cup \bigcup_{j \leqslant i, R_j \in \mathcal{R}} L_S(R_j)$;
(C) R_M^i *is a supertree compatible with* $\mathcal{R}$.

The proof of the lemma is done by induction on the iterations of the loop in algorithm BUILDSMCT, and is included in Appendix C. The correctness of the algorithm BUILDSMCT directly derives from Lemma 1. Moreover, its running time mainly depends on that of the algorithm for solving MCT on an instance of the same or smaller size.

Theorem 4. *Let* $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ *be a collection of rooted trees such that* $L_\Delta(\mathcal{R}) = \emptyset$*. Algorithm* BUILDSMCT$(\mathcal{R})$ *computes a maximum compatible supertree of* $\mathcal{R}$ *in* $O(N + kn)$ *time where* n *is the maximum number of leaves in a tree of* $\mathcal{R}$*, and* N *is the time needed to compute a rooted maximum compatible tree of* $\mathcal{R}|L_\cap(\mathcal{R})$.

Proof. From Lemma 1(C), the tree R_M^k returned by the algorithm is a supertree compatible with $\mathcal{R}$. Moreover, it is of maximum size among such supertrees. Indeed, suppose there is a tree $R = SMCT(\mathcal{R})$ such that $|R| > |R_M^k|$. Since $L(R_M^k) = L_S(\mathcal{R}) \cup L(R_M^0)$ (from Lemma 1(B)) and $L(\mathcal{R}) = L_S(\mathcal{R}) \cup L_\cap(\mathcal{R})$ (from $L_\Delta(\mathcal{R}) = \emptyset$) then R contains more leaves of $L_\cap(\mathcal{R})$ than R_M^k, i.e.

$$\left|R|L_\cap(\mathcal{R})\right| > \left|R_M^k|L_\cap(\mathcal{R})\right| = \left|R_M^0|L_\cap(\mathcal{R})\right|. \tag{17}$$

However, as $\mathcal{R}|L_\cap(\mathcal{R})$ is a collection of trees on the same leaf set, $L_\cap(\mathcal{R})$, and $L(R|L_\cap(\mathcal{R})) \subseteq L_\cap(\mathcal{R})$, the fact that R is a supertree compatible with $\mathcal{R}$ implies that $R|L_\cap(\mathcal{R})$ is a tree compatible with the collection $\mathcal{R}|L_\cap(\mathcal{R})$. But then (17) is in contradiction with the maximality of R_M^0 among the trees compatible with this collection. Thus, R_M^k is a maximum compatible supertree of $\mathcal{R}$.

Concerning the running time, the kn term results from both the restrictions of input trees and from calls to MERGETREES: lines 1 and 2 restrict each of the k input trees to a subset of its leaves, necessitating a single $O(n)$ traversal of the tree each time; moreover, line 2 performs k calls to MERGETREES, each requiring a time proportional to the size $O(n)$ of the trees given as input to the call, by Theorem 2. The N term results from the computation of a maximum compatible tree of $\mathcal{R}|L_\cap(\mathcal{R})$ in line 1. □

Note that the kn term in the complexity of BUILDSMCT can be reduced to an n term by integrating this algorithm with MERGETREES and computing all anchors between the trees of $\mathcal{R}$ and R_M^0 before performing the grafts of specific subtrees. However, the N majoring term would remain.

A simple modification of the algorithm BUILDSMCT yields an algorithm, BUILDSMAST, that solves SMAST: in line 1, use a tree $MAST(\mathcal{R}|L_\cap(\mathcal{R}))$ instead of a tree $MCT(\mathcal{R}|L_\cap(\mathcal{R}))$ to initialize R_M^0.

Theorem 5. *Let* $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ *be a collection of rooted trees such that* $L_\Delta(\mathcal{R}) = \emptyset$*. Algorithm* BUILDSMAST$(\mathcal{R})$ *computes a maximum agreement supertree of* $\mathcal{R}$ *in* $O(N' + kn)$ *time where* n *is the maximum number of leaves in a tree of* $\mathcal{R}$ *and* N' *is the time needed to compute a rooted maximum agreement subtree of* $\mathcal{R}|L_\cap(\mathcal{R})$.

Proof. The correctness of BUILDSMAST is shown in a very similar way as that of BUILDSMCT, replacing in the above results (e.g. in Lemma 1(A)) each refinement relation between trees by an equality (i.e. isomorphism) of these trees. The complexity proof is quite similar to that of Theorem 4, with N being replaced with N'. □

Now consider the case where the input trees are unrooted. By rooting unrooted trees on the edge leading to a common leaf, isomorphism and refinement relations between unrooted trees translate into the same relations between corresponding rooted trees [8, Lemma 4]. Hence, the SMAST and SMCT problems on unrooted trees can be easily reduced to the same problems on rooted trees.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Theorem 6. *Let $\mathcal{U} = \{U_1, U_2, \ldots, U_k\}$ be a collection of unrooted trees sets such that $L_\Delta(\mathcal{U}) = \emptyset$.*

One can compute a tree SMAST($\mathcal{U}$), respectively SMCT($\mathcal{U}$), in $O(M + kn)$ time where M is the time needed to compute an unrooted *tree MAST($\mathcal{U}|L_\cap(\mathcal{U})$), respectively MCT($\mathcal{U}|L_\cap(\mathcal{U})$).*

Proof. Consider the case of the SMCT problem (the proof for SMAST is similar). Make the following modifications to BUILDSMCT: first compute $U_M^0 = MCT(\mathcal{U}|L_\cap(\mathcal{U}))$ by applying an algorithm to solve the problem on unrooted trees [19]. Then choose an arbitrary leaf $\ell \in L(U_M^0)$, compute the collection $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of rooted trees such that $R_i \in \mathcal{R}$ is obtained by rooting $U_i \in \mathcal{U}$ (inserting a new node) on the external edge leading to leaf ℓ. Similarly, R_M^0 is initialized as the tree obtained by rooting U_M^0 on the edge leading to ℓ. Then the `for` loop remains the same. The last modification is to unroot the obtained tree R_M^k before returning it.

Concerning the correctness of the modified algorithm, first note that $\mathcal{R}_M^0 = MCT(\mathcal{R}|L_\cap(\mathcal{R}))$ [8, Lemma 5]. Now, the k calls to algorithm MERGETREES give a tree R_M^k such that $R_M^k = SMCT(\mathcal{R})$ (Theorem 4) and such that $L(R_M^k) = L(R_M^0) \cup L_S(\mathcal{R})$ (Lemma 1(B)). Let U_M^k be the tree obtained by unrooting R_M^k. Since refinement relations are preserved by unrooting trees [8, Lemma 4], U_M^k is a supertree compatible with $\mathcal{U}$. Moreover, it is of maximum size. Indeed, a maximum compatible supertree U' of $\mathcal{U}$ including more leaves than U_M^k would necessarily contain more leaves of $L_\cap(\mathcal{U})$ than U_M^k (because $L(\mathcal{U}) = L_\cap(\mathcal{U}) \cup L_S(\mathcal{U})$ and $L_S(\mathcal{U}) \subseteq L(U_M^k) = L(R_M^k) = L(R_M^0) \cup L_S(\mathcal{R}) = L(U_M^0) \cup L_S(\mathcal{U})$). Thus, U' would contain more leaves of $L_\cap(\mathcal{U})$ than U_M^0 does, implying that $U'|L_\cap(\mathcal{U})$ would be a tree compatible with $\mathcal{U}|L_\cap(\mathcal{U})$ of larger size than U_M^0, which is a contradiction with the definition of the latter.

The running time differs from the original BUILDSMCT by the fact that the MCT is computed on unrooted trees, requiring $O(M)$ time instead of $O(N)$. Choosing ℓ is $O(1)$ time, computing $\mathcal{R}$ is $O(kn)$ time, and unrooting R_M^k is $O(1)$ time. Taking restrictions of trees in line 2 is $O(kn)$. Thus, the modified algorithm requires $O(M + kn)$ time. □

The previous theorems enable to state the relationships between subtree and supertree problems for a collection $\mathcal{T}$ when $L_\Delta(\mathcal{T}) = \emptyset$.

Corollary 2. *Let $\mathcal{T}$ be a collection of trees such that $L_\Delta(\mathcal{T}) = \emptyset$. Any tree MAST($\mathcal{T}|L_\cap(\mathcal{T})$), respectively MCT($\mathcal{T}|L_\cap(\mathcal{T})$), is the restriction to $L_\cap(\mathcal{T})$ of some tree SMAST($\mathcal{T}$), respectively SMCT($\mathcal{T}$).*

Note that the condition required for Corollary 2 to apply is always fulfilled for collections $\mathcal{T}$ of only two trees, because $L(\mathcal{T}) = L_\cap(\mathcal{T}) \cup L_S(\mathcal{T})$. Fig. 7 shows an illustration of the corollary in such a case. Lastly, note that it is also true that in this case the restriction to $L_\cap(\mathcal{T})$ of any tree *SMAST*($\mathcal{T}$), respectively *SMCT*($\mathcal{T}$), is a tree *MAST*($\mathcal{T}|L_\cap(\mathcal{T})$), respectively *MCT*($\mathcal{T}|L_\cap(\mathcal{T})$).

3.4. Polynomial cases

Particular cases where SMAST and SMCT problems can be solved in polynomial time are deduced from the above results and from works on MAST and MCT.

Corollary 3. *Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of trees* (*all rooted or all unrooted*).

(i) *The SMAST problem on two trees can be solved in polynomial time.*

Moreover, when $L_\Delta(\mathcal{T}) = \emptyset$,

(ii) *the SMAST problem can be solved in polynomial time whenever the maximum degree of an input tree is bounded,*
(iii) *the SMCT problem on $\mathcal{T}$ can be solved in polynomial time whenever the degree of all input trees is bounded.*

Proof. Consider the case where trees in $\mathcal{T}$ are rooted. In this case: (i) derives from Theorem 5 and from efficient algorithms that solve MAST when $\mathcal{T}$ contains only two trees [14,29,30]; (ii) follows from Theorem 5 and the algorithms of [2,12]; (iii) follows from Theorem 4 and [22]. If $\mathcal{T}$ is a collection of unrooted trees, then the result follows from Theorem 6 and the works cited in the rooted case.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

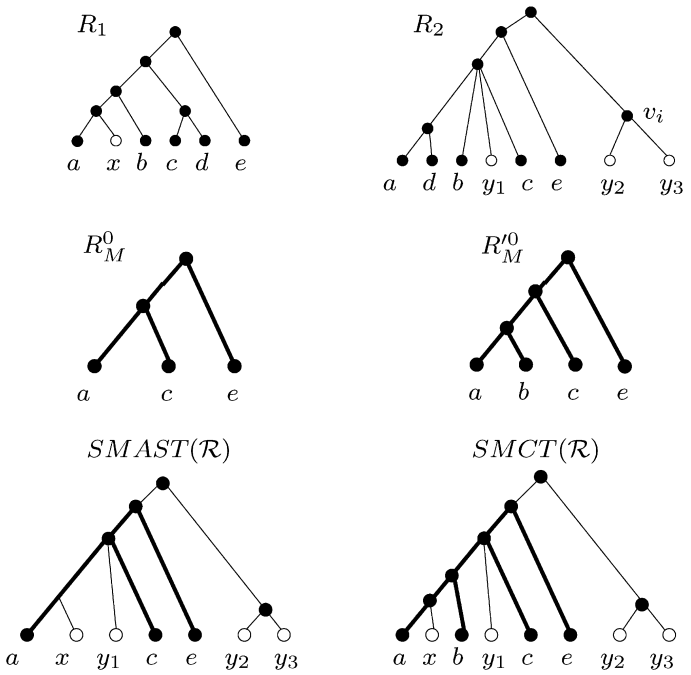

Fig. 7. A collection $\mathcal{R} = \{R_1, R_2\}$ of two input trees ($L_\cap(\mathcal{R}) = \{a, b, c, d, e\}$), two trees $R^0_{\mathrm{M}} := MAST(\mathcal{R}|L_\cap(\mathcal{R}))$ and $R'^0_{\mathrm{M}} := MCT(\mathcal{R}|L_\cap(\mathcal{R}))$ and two trees $SMAST(\mathcal{R})$ and $SMCT(\mathcal{R})$, in which the structure of R^0_{M}, respectively R'^0_{M}, is displayed in bold lines.

4. Reduction to instances involving smaller trees

We now show how trees of arbitrary size can be described by subtrees of small bounded size. This decomposition will be used to prove intractability results in the next section. Rooted trees of arbitrary size can be described by rooted trees on three leaves.

Definition 13 *(Rooted triples and fans).* A *rooted triple* (or *resolved triple*) is a binary rooted tree on three leaves. A *fan* (also called *unresolved triple*) is a rooted tree on three leaves with only one internal node. On three given distinct leaves a, b and c, there are three possible rooted triples, denoted $bc|a$, respectively $ac|b$, respectively $ab|c$, depending on their innermost grouping of two leaves (bc, respectively ac, respectively ab). E.g. tree R^0_M of Fig. 7 is the rooted triple $ac|e$. The only one possible fan on this set of leaves is denoted (a, b, c).

Let R be a rooted tree. For any set $\{a, b, c\}$ of three leaves in $L(R)$, $R|\{a, b, c\}$ is either a rooted triple or a fan. We define $rt(R)$, respectively $f(R)$, as the set of rooted triples, respectively fans, of R induced by the 3-leaf subsets of $L(R)$.

For instance, in Fig. 2,

$$rt(R_2) = \{ad|b, ad|c, ad|e, ab|e, ac|e, bd|e, cd|e, bc|e\},$$
$$f(R_2) = \{(a, b, c), (b, c, d)\}.$$

The basic building stones of unrooted trees are *quartets* and *stars*:

Definition 14 *(Unrooted quartets and stars).* A *quartet* is a binary unrooted tree on four leaves. A *star* is an unrooted tree with only one internal node to which four leaves are connected. Given four distinct leaves a, b, c and d, there are three possible quartets, respectively denoted $ab|cd$ (corresponding to the binary tree in which the path from a to b does not intersect the path from c to d), $ac|bd$ and $ad|bc$, and only one possible star denoted (a, b, c, d).

Let U be an unrooted tree. For any set Q of four leaves appearing in U, $U|Q$ is either a quartet or a star. We define $q(U)$, respectively $s(U)$, as the set of quartets, respectively stars, of U induced by 4-leaf subsets of $L(U)$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

For instance, in Fig. 1,

$$q(U) = \{ad|bc, ad|be, ad|bf, ad|ce, ad|cf, ad|ef, bc|ef, bd|ef, cd|ef, ab|ef, ac|ef\},$$
$$s(U) = \{(a, b, c, e), (a, b, c, f), (b, c, d, e), (b, c, d, f)\}.$$

The following well-known results show that isomorphism and compatibility of rooted, respectively unrooted, trees can be expressed by relations on sets of induced rooted triples and fans, respectively quartets and stars:

Lemma 2. *Let R, R' be two rooted trees.*

(i) *R is isomorphic to R' if and only if $rt(R) = rt(R')$ and $f(R) = f(R')$.*
(ii) *R refines R' if and only if $rt(R') \subseteq rt(R)$ and $L(R) = L(R')$.*

Let U, U' be two unrooted trees.

(iii) *U is isomorphic to U' if and only if $q(U) = q(U')$ and $s(U) = s(U')$.*
(iv) *U refines U' if and only if $q(U') \subseteq q(U)$ and $L(U) = L(U')$.*

Proof. (i) derives from [12, Lemma 6.6], (iii) is [2, Theorem 2] and [13, Theorem 1] yields (ii) and (iv). □

We can now show that solving SMAST and SMCT on instances with input trees of arbitrary degree is equivalent to solving the same problems on trees with both degree and size bounded by a small constant. This contrasts with MAST and MCT which are trivial when the input trees contain a bounded number of leaves. Moreover, MAST is polynomial in the case where an input tree has a bounded degrees [2,12]. Note also that having trees with bounded degree is a sufficient condition for the algorithm of [22] to solve MCT in polynomial time.

We first define collections of rooted triples and fans, respectively unrooted quartets and stars that can be obtained from a collection of general rooted trees, respectively general unrooted trees:

Definition 15. Given a collection $\mathcal{R}$ of rooted trees, define:

$$rt_{\cup}(\mathcal{R}) = \bigcup_{R_i \in \mathcal{R}} rt(R_i) \quad \text{and} \quad f_{\cup}(\mathcal{R}) = \bigcup_{R_i \in \mathcal{R}} f(R_i).$$

Similarly, given a collection $\mathcal{U}$ of unrooted trees, define:

$$q_{\cup}(U) = \bigcup_{U_i \in \mathcal{U}} q(U_i) \quad \text{and} \quad s_{\cup}(U) = \bigcup_{U_i \in \mathcal{U}} s(U_i).$$

From Lemma 2, we deduce:

Corollary 4. *Let $\mathcal{R}$ be a collection of rooted trees and let R be a rooted tree with $L(R) \subseteq L(\mathcal{R})$.*

(i) *R is an agreement supertree of $\mathcal{R}$ if and only if R is an agreement supertree of $rt_{\cup}(\mathcal{R}) \cup f_{\cup}(\mathcal{R})$,*
(ii) *R is a supertree compatible with $\mathcal{R}$ if and only if R is a supertree compatible with $rt_{\cup}(\mathcal{R})$.*

Let $\mathcal{U}$ be a collection of unrooted trees and let U be an unrooted tree with $L(U) \subseteq L(\mathcal{U})$.

(iii) *U is an agreement supertree of $\mathcal{U}$ if and only if U is an agreement supertree of $q_{\cup}(\mathcal{U}) \cup s_{\cup}(\mathcal{U})$,*
(iv) *U is a supertree compatible with $\mathcal{U}$ if and only if U is a supertree compatible with $q_{\cup}(\mathcal{U})$.*

Proof. Assertions (i), (ii), (iii) and (iv) of Corollary 4 are easily deduced from statements (i), (ii), (iii) and (iv) of Lemma 2, respectively. □

If k and n respectively denote the number of trees and the number of leaves in the collection, note that

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

- $rt_{\cup}(\mathcal{R})$ and $f_{\cup}(\mathcal{R})$ are computable in $O(kn^3)$ time from $\mathcal{R}$;
- $q_{\cup}(\mathcal{U})$ and $s_{\cup}(\mathcal{U})$ are computable in $O(kn^4)$ time from $\mathcal{U}$.

Since approximating MAST on rooted trees is at least as hard as approximating MAXIMUM CLIQUE [20], approximating SMAST on rooted triples and fans is at least as hard as approximating MAXIMUM CLIQUE. In the same way, building on a result of [7], approximating SMCT on rooted triples is at least as hard as approximating MAXIMUM INDEPENDENT SET.

5. Intractability of SMAST and SMCT

In this section, we show that there is a substantial gap in complexity between MAST and SMAST, respectively MCT and SMCT. The complement of the SMAST problem, denoted CSMAST, is defined as the minimization problem obtained from SMAST by changing, in Definition 8, "**Measure:** $|T|$, *to be maximized*", into "**Measure:** $\mathrm{Card}(L(\mathcal{T})) - |T|$, *to be minimized*". The complement of SMCT, denoted CSMCT, is obtained in the same way from Definition 9. Note that trees involved in practical phylogenetic instances are expected to conflict on a small proportion of leaves. Thus, $\mathrm{Card}(L(\mathcal{T})) - |SMAST(\mathcal{T})|$ and $\mathrm{Card}(L(\mathcal{T})) - |SMCT(\mathcal{T})|$ are expected to be small. Hence, approximating CSMAST and CSMCT is more interesting than approximating SMAST and SMCT. The complement of MAST, respectively MCT, is defined to be the restriction of CSMAST, respectively CSMCT, to instances consisting in collections of trees sharing the same leaf set. The complement of MCT is approximable within ratio 3 [18], as is also well-known for the complement of MAST [2,7]. The latter result was also recently improved to a ratio $3 - \frac{6 \log\log n}{\log n}$ [27]. In contrast to these positive results, CSMAST, respectively CSMCT, in its general form is NP-hard to approximate within any constant ratio, as shown below in Theorem 9.

Moreover, consider the decision problem corresponding to CSMAST:

Instance: A finite collection $\mathcal{T}$ of trees and an integer $p \geqslant 0$.
Question: Is there an agreement supertree of $\mathcal{T}$ of size at least $\mathrm{Card}(L(\mathcal{T})) - p$?

The decision problem corresponding to CSMCT is defined in the same way (replace "agreement supertree of $\mathcal{T}$" by "supertree compatible with $\mathcal{T}$" in the above statement of the "**Question:**"). Theorem 8 below shows that CSMAST and CSMCT are hard for parameter p unlike the complements of MAST and MCT which are FPT in p (see [8] for the latest algorithms).

5.1. *The* HITTING SET *problem*

As often done in previous works (e.g. [2,12]), we exploit links between the MAST problem and the HITTING SET problem. A *hitting set* of a collection of sets $\mathcal{C}$ is a set H such that for all $C \in \mathcal{C}$, $H \cap C$ is non-empty. Consider the decision problem:

Name: HITTING SET
Instance: A finite collection $\mathcal{C}$ of finite sets and an integer $p \geqslant 0$.
Question: Is there a hitting set of $\mathcal{C}$ of cardinality at most p?

HITTING SET is an alternative formulation of SET COVER. It is thus NP-complete [15], and W[2]-complete for parameter p [16, Proposition 10]. Moreover, its optimization version can not be approximated within any constant ratio unless P = NP [5].

5.2. *A graph representing rooted triples*

Definition 16. [1,13,36] Let $\mathcal{R}$ be a finite collection of rooted triples and let $L \subseteq L(\mathcal{R})$. Let $[\mathcal{R}, L]$ be the undirected graph such that:

- there is a vertex for every element of L,

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

- there is an edge between two vertices u and v if and only if there exists $\ell \in L$ such that $uv|\ell \in \mathcal{R}$.

Theorem 2 in [13] can be restated as follows:

Theorem 7. [13] *Let $\mathcal{R}$ be a collection of rooted triples and $L \subseteq L(\mathcal{R})$. There is an agreement supertree of $\mathcal{R}$ with leaf set L if and only if for each subset $L' \subseteq L$ of cardinality at least 3, the graph $[\mathcal{R}, L']$ is disconnected.*

5.3. The gadget

Definition 17. We recursively define the function *rake* associating a rooted tree to a given non-empty ordered sequence of rooted trees with non-intersecting leaf sets:

- $\text{rake}(R_1) = R_1$ for any rooted tree R_1 (sequence of length 1).
- $\text{rake}(R_1, R_2, \ldots, R_k)$ is the rooted tree whose root has R_1 and $\text{rake}(R_2, R_3, \ldots, R_k)$ as two child subtrees for any sequence of rooted trees $R_1, R_2, \ldots, R_k$ of length $k \geqslant 2$ such that

$$\forall i, j \in [1, k] \quad i \neq j \iff L(R_i) \cap L(R_j) = \emptyset.$$

Fig. 8 illustrates the previous definition. We now describe the gadget that is used to reduce HITTING SET to SMAST:

Definition 18 *(Gadget).* Let m be an integer such that $m \geqslant 1$ and let $x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m$ be $2m$ distinct labels. We define $\mathcal{G} = \mathcal{G}(x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m)$ to be the following collection of rooted triples:

$$\left\{y^{h+1}x^{h+1}|y^h, x^{h+1}x^{h+2}|y^h\right\}_{h\in[1,m]}$$

setting $x^{m+1} := x^1$, $x^{m+2} := x^2$ and $y^{m+1} := y^1$.

Lemma 3. *$\mathcal{G}$ has the following properties*:

(i) *There is no agreement supertree of $\mathcal{G}$ having leaf set $L(\mathcal{G})$.*
(ii) *Let $j \in [1, m]$. The following trees with leaf set $L(\mathcal{G}) - \{x^j\}$ are agreement supertrees of $\mathcal{G}$:*

$$\text{rake}(y^j, y^{j+1}, \ldots, y^m, y^1, y^2, \ldots, y^{j-1}, R^*)$$

where R^* is any rooted tree on $\{x^1, x^2, \ldots, x^m\} - \{x^j\}$.

Proof. (i) The graph $[\mathcal{G}, L(\mathcal{G})]$ associated with $\mathcal{G}$ is connected (see Fig. 9). Therefore, by Theorem 7, there is no agreement supertree of $\mathcal{G}$ having leaf set $L(\mathcal{G})$.

(ii) Assume w.l.o.g. that $j = 1$ ($\mathcal{G}$ is not altered by a common circular permutation of the two sequences $x^1, x^2, \ldots, x^m$ and $y^1, y^2, \ldots, y^m$). Fixing an arbitrary rooted tree R^* on $\{x^2, x^3, \ldots, x^m\}$, we have to show that the tree

$$R_A := \text{rake}(y^1, y^2, \ldots, y^m, R^*)$$

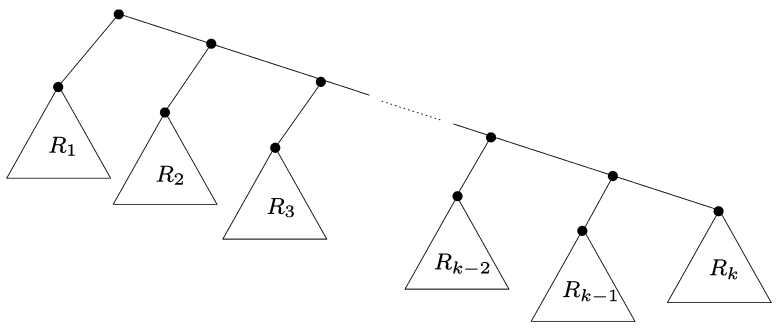

Fig. 8. The tree $\text{rake}(R_1, R_2, \ldots, R_k)$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

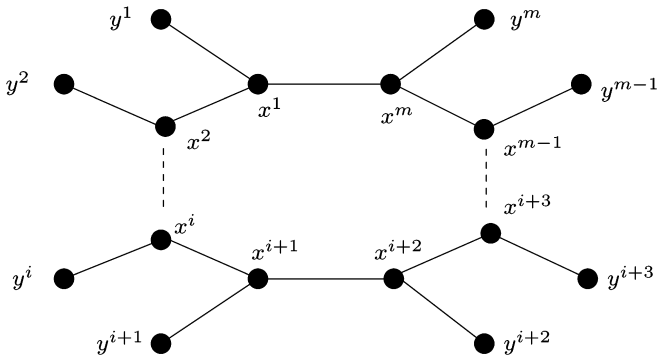

Fig. 9. The graph $[\mathcal{G}, L(\mathcal{G})]$ induced by the gadget $\mathcal{G}$.

on $L(\mathcal{G}) - \{x^1\}$ is an agreement supertree of $\mathcal{G}$. For this purpose, we distinguish trees in $\mathcal{G}$ that do not contain x^1 from those that do.

In the one hand, it is easily seen that

$$\forall h \in [1, m-1] \quad y^{h+1}x^{h+1}|y^h = R_A|\{y^h, y^{h+1}, x^{h+1}\},$$
$$\forall h \in [1, m-2] \quad x^{h+1}x^{h+2}|y^h = R_A|\{y^h, x^{h+1}, x^{h+2}\}.$$

On the other hand, $x^1 = x^{m+1}$ is a leaf of $y^{m+1}x^{m+1}|y^m$ and of $x^{h+1}x^{h+2}|y^h$ for $h \in \{m-1, m\}$. Hence, restricting these three trees to $L(\mathcal{G}) - \{x^1\}$ reduces them to only two leaves, belonging to R_A. We have shown that $\forall G_i \in \mathcal{G}$, $R_A|L(G_i) = G_i|L(R_A)$. As also $L(R_A) \subseteq L(\mathcal{G})$, this proves that R_A is an agreement supertree of $\mathcal{G}$. □

In other words, $\mathcal{G}$ is a collection of conflicting trees in the sense that there is no tree R with the entire $L(\mathcal{G})$ as leaf set and displaying all trees of $\mathcal{G}$. However, choosing only one leaf x^j (any one) and removing from $\mathcal{G}$ all triples containing x^j guarantees that such a tree exists. It is formed by making leaves y^h (with $h \in [1, m]$) pending in a specific order from the successive internal nodes of the tree (starting from the root and going downward), last appending a subtree containing the leaves x^h (with $h \in [1, m]$, $h \neq j$) but which can have any shape.

5.4. The reductions

Theorem 8. *The CSMAST problem is NP-hard, and* W[2]*-hard for parameter p, even for instances $\mathcal{T}$ only composed of rooted triples, respectively unrooted quartets.*

Proof. *Rooted case*. We reduce HITTING SET to CSMAST, polynomially and preserving the parameter p. Let $(\mathcal{C}, p)$ be an instance of HITTING SET,

$$\begin{aligned}\mathcal{C} &= \{X_1, X_2, \ldots, X_c\} \\ &= \big\{\{x_1^1, x_1^2, \ldots, x_1^{m_1}t\}, \{x_2^1, x_2^2, \ldots, x_2^{m_2}\}, \ldots, \{x_c^1, x_c^2, \ldots, x_c^{m_c}\}\big\},\end{aligned}$$

where $c := \mathrm{Card}\,(\mathcal{C})$ and $m_i := \mathrm{Card}(X_i)$. Then let (y_i^j) be an injective family of labels not appearing in $X_1 \cup X_2 \cup \cdots \cup X_c$, indexed on the set of ordered pairs (i, j) with $i \in [1, c]$, $j \in [1, m_i]$. Based on the model of $\mathcal{C}$, we build a collection of non-intersecting sets

$$\big\{\{y_1^1, y_1^2, \ldots, y_1^{m_1}\}, \{y_2^1, y_2^2, \ldots, y_2^{m_2}\}, \ldots, \{y_c^1, y_c^2, \ldots, y_c^{m_c}\}\big\}$$

whose elements are distinct from those of $\mathcal{C}$. Let

$$\mathcal{G}_i := \mathcal{G}(x_i^1, x_i^2, \ldots, x_i^{m_i}, y_i^1, y_i^2, \ldots, y_i^{m_i})$$

for all $i \in [1, c]$ and consider the collection of trees

$$\mathcal{R} := \mathcal{G}_1 \cup \mathcal{G}_2 \cup \cdots \cup \mathcal{G}_c.$$

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

From the definition of gadgets (Definition 18), the transformation of the instance $(\mathcal{C}, p)$ of HITTING SET into the instance $(\mathcal{R}, p)$ of the decision problem CSMAST obviously takes a polynomial time ($\mathcal{R}$ is of cardinality $2(m_1 + m_2 + \cdots + m_c)$) and preserves parameter p. By construction, all trees in $\mathcal{R}$ are rooted triples. It remains to be proven that the following two statements are equivalent:

(1) $\mathcal{C}$ admits a hitting set of size at most p and
(2) there is an agreement supertree of $\mathcal{R}$ whose size is at least $\mathrm{Card}(L(\mathcal{R})) - p$.

(2) $\Rightarrow$ (1). Let R_A be an agreement supertree of $\mathcal{R}$ of size at least $\mathrm{Card}(L(\mathcal{R})) - p$. Thus, $H := L(\mathcal{R}) - L(R_A)$ is a set of cardinality at most p. Moreover, for any $i \in [1, c]$, we know that $L(\mathcal{G}_i)$ is not a subset of $L(R_A)$ from Lemma 3(i). Then at least one element of $L(\mathcal{G}_i)$ is not a leaf in R_A, hence is in H. This shows that H is a hitting set of $\{L(\mathcal{G}_1), L(\mathcal{G}_2), \ldots, L(\mathcal{G}_c)\}$.

Now change H in the following way: replace each $y_i^j \in H$ (which only hits the set $L(\mathcal{G}_i)$) with any element in X_i. H is then a hitting set of $\mathcal{C}$ of size at most p.

(1) $\Rightarrow$ (2). Given $i \in [1, c]$ and $j \in [1, m_i]$, we denote σ_i^j the $(j-1)$th cyclic shift of the sequence $y_i^1, y_i^2, \ldots, y_i^{m_i}$:

$$\sigma_i^j = y_i^j, y_i^{j+1}, \ldots, y_i^{m_i}, y_i^1, y_i^2, \ldots, y_i^{j-1}.$$

Let H be a hitting set of $\mathcal{C}$ of cardinality at most p. For each $i \in [1, c]$, H contains at least an element of X_i, that we denote $x_i^{j_i}$, with $j_i \in [1, m_i]$. Concatenate the c sequences $\sigma_1^{j_1}, \sigma_2^{j_2}, \ldots, \sigma_c^{j_c}$: this yields a label sequence $z_1, z_2, \ldots, z_m$ of length $m := m_1 + m_2 + \cdots + m_c$. Then form the tree

$$R_A := \mathrm{rake}(z_1, z_2, \ldots, z_m, R^*)$$

where R^* is any rooted tree on $(X_1 \cup X_2 \cup \cdots \cup X_c) - H$.

By construction, R_A is a tree on $L(\mathcal{R}) - H$ and thus of size at least $\mathrm{Card}(L(\mathcal{R})) - p$. Moreover, for each $i \in [1, c]$, R_A (consisting of leaves of the kind y_i^j hanging one by one from internal nodes on the path from the root of R_A to the root of subtree R^*) is such that $R_A|L(\mathcal{G}_i) = \mathrm{rake}(y_i^{j_i}, y_i^{j_i+1}, \ldots, y_i^{m_i}, y_i^1, y_i^2, \ldots, y_i^{j_i-1}, R^*|X_i)$, which is an agreement supertree of $\mathcal{G}_i$ by Lemma 3(ii) (note that $x_i^{j_i} \in H$ is not a leaf in R^*, hence not in $R^*|X_i$). Thus, R_A is an agreement supertree of $\mathcal{R}$ of size at least $\mathrm{Card}(L(\mathcal{R})) - p$.

Unrooted case. We reduce below the version of CSMAST using a collection of rooted triples as input, to the version of CSMAST using a collection of binary unrooted trees as input. Note that solving CSMAST on binary unrooted trees is equivalent to solving this problem on unrooted quartets (Corollary 4(iii)).

Let $(\mathcal{R}, p)$ be an instance of CSMAST where $\mathcal{R}$ is a collection of rooted triples and p a non-negative integer.

Let R' be a rooted binary tree containing $\mathrm{Card}(L(\mathcal{R}))$ *new* leaves: $|R'| = \mathrm{Card}(L(\mathcal{R}))$ and $L(R') \cap L(\mathcal{R}) = \emptyset$. Consider the collection $\mathcal{U}$ of unrooted trees where the trees $U_i \in \mathcal{U}$ are in one to one correspondence with the trees $R_i \in \mathcal{R}$: given a tree $R_i \in \mathcal{R}$, the corresponding tree U_i is the unrooted tree obtained by adding an edge between the root of R_i and the root of a copy of R'. We have $L(\mathcal{U}) = L(R') \cup L(\mathcal{R})$ and $\mathrm{Card}(L(\mathcal{U})) = 2\,\mathrm{Card}(L(\mathcal{R}))$.

The instance $(\mathcal{R}, p)$ of CSMAST is transformed into an another instance $(\mathcal{U}, p)$ of CSMAST where all trees in $\mathcal{U}$ are *unrooted*. This transformation is clearly done in polynomial time and preserves parameter p.

We now prove that $(\mathcal{R}, p)$ is a positive instance of CSMAST if and only if $(\mathcal{U}, p)$ is a positive instance of CSMAST.

First, suppose that there is an agreement supertree R_A of $\mathcal{R}$ with $|R_A| \geqslant \mathrm{Card}(L(\mathcal{R})) - p$. Then the unrooted tree U_A, obtained by connecting the roots of R_A and a copy of R' by an edge, is an agreement supertree of $\mathcal{U}$ of size $|R_A| + |R'| \geqslant \mathrm{Card}(L(\mathcal{R})) - p + |R'| = \mathrm{Card}(L(\mathcal{U})) - p$.

Conversely, assume there is an agreement supertree U_A of $\mathcal{U}$ with $|U_A| \geqslant \mathrm{Card}(L(\mathcal{U})) - p$. We can assume that $p < \mathrm{Card}(L(\mathcal{R}))$ since otherwise, the empty tree is clearly an agreement supertree of $\mathcal{R}$ of size at least $\mathrm{Card}(L(\mathcal{R})) - p$. Hence, at most $\mathrm{Card}(L(\mathcal{R})) - 1$ leaves appearing in $\mathcal{U}$ are not leaves of U_A. Since $\mathrm{Card}(L(\mathcal{R}))$ distinct leaves of $\mathcal{U}$ appear in R', there is a leaf ℓ' of R' that is also a leaf of U_A. Let $U' := U_A|(L(\mathcal{R}) \cup \{\ell'\})$. Note that U' is an agreement supertree of $\mathcal{U}$ and, as for U_A, contains at least $\mathrm{Card}(L(\mathcal{R})) - p$ leaves from $L(\mathcal{R})$. Let R_A be the tree with leaves in $L(\mathcal{R})$, obtained by rooting U' at the leaf ℓ', and then deleting ℓ' and its incident edge. R_A is an agreement supertree of $\mathcal{R}$ and as $\ell' \notin L(\mathcal{R})$, R_A has at least $\mathrm{Card}(L(\mathcal{R})) - p$ leaves from $L(\mathcal{R})$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Theorem 9. *CSMAST is not approximable within any constant factor unless* P = NP*, even for instances* $\mathcal{T}$ *only composed of rooted triples, respectively unrooted quartets.*

Proof. The reduction from HITTING SET to CSMAST on rooted triples and the reduction from CSMAST on rooted triples to CSMAST on unrooted quartets described in Theorem 8 can be seen as approximation preserving reductions.

Hence, the result of [5] stated in Section 5.1 for HITTING SET also applies to CSMAST on rooted triples and to CSMAST on unrooted quartets. □

The above two intractability and inapproximability results also hold for the CSMCT problem, as the reductions use collections of binary trees, i.e. cases in which this problem is equivalent to CSMAST.

Appendix A. Determining anchors between two rooted trees in linear time

In this section, we show how *anchors* from nodes of R_I to nodes of R_A can be determined in $O(n)$ where $n := \text{Card}(L(R_I) \cup L(R_A))$.

Note that nodes of R_I for which an anchor has to be determined are nodes $v_i := \text{lca}_{R_I}(S)$ for some set $S \subseteq L_\cap(\{R_A, R_I\})$. Here such nodes are called *anchorable*.

The *leaf*-nodes in R_I whose label is in $L_\cap(\{R_A, R_I\})$ are anchorable. Their corresponding set S contains the single leaf of R_A having the same label. A single traversal of R_i and one of R_A is thus enough to establish the anchors of leaves of R_i. This costs $O(n)$.

Now consider anchorable *internal* nodes (i.e. for which $\text{Card}(S) > 1$). To anchor these nodes, we proceed by considering specific couples of leaves in $L_\cap(\{R_A, R_I\})$. Let $\mathcal{O}$ be the left-right order with which the leaves of $L_\cap(\{R_A, R_I\})$ appear in the tree R_I, and denote the ℓth element in $\mathcal{O}$ as $\mathcal{O}_\ell$. Let m be the cardinality of $L_\cap(\{R_A, R_I\})$. Then consider the $m = O(n)$ pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ of consecutive leaves in $\mathcal{O}$. For each such pair, two lca queries are performed to identify $v_i := \text{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_a := \text{lca}_{R_A}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. For some of these pairs, v_a is the anchor of v_i. As lca relationships can be preprocessed in trees R_I and R_A in $O(n)$ time to answer each lca query in $O(1)$ [25], considering the $O(n)$ pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ requires $O(n)$ running time. The correctness of this approach is demonstrated below. More precisely, it is shown that considering all pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$, $1 \leqslant \ell < m$, is sufficient to determine the anchor of all anchorable internal nodes of R_I.

First, by definition, any anchorable internal node $v_i \in R_I$ is the lca of a set $S \subseteq L_\cap(\{R_A, R_I\})$ with $\text{Card}(S) \geqslant 2$, hence v_i has leaves of S in $s \geqslant 2$ child subtrees, denoted $c_1, \ldots, c_s$. Now, by definition of $\mathcal{O}$, for all couples $(c_j, c_j + 1)$, $1 \leqslant j < s$, the right-most leaf of $L(c_j) \cap S$ just precedes in $\mathcal{O}$ the left-most leaf of $S \cap L(c_{j+1})$, and v_i is the lca of these two leaves. Hence, examining the lca in R_I of all couples $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$, $1 \leqslant \ell < m$ of consecutive leaves in $\mathcal{O}$ ensures that v_i is considered at some step.

Proposition 1. *Given a node* v_i *in* R_I *such that* $v_i = \text{lca}_{R_I}(S)$ *with* $S \subseteq L_\cap(\{R_A, R_I\})$ *and* $\text{Card}(S) \geqslant 2$*. Let* $\mathcal{O}_{s_1}, \ldots, \mathcal{O}_{s_j}$ *be the leaves of* S*, with* $1 \leqslant s_1 < s_2 < \cdots < s_j \leqslant n$*. Let* v_a *be the node of* R_A *such that* $v_a := \text{lca}_{R_A}(S)$*. Then there is an integer* ℓ*, such that* $s_1 \leqslant \ell < s_j$ *and* $v_a = \text{lca}_{R_A}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$.

Proof. First note that the elements of S are consecutive in $\mathcal{O}$. Let c be the child subtree of v_a that contains $\mathcal{O}_{s_1}$. Let ℓ be the smallest integer such that $s_1 \leqslant \ell < s_j$ and $\mathcal{O}_{\ell+1} \notin L(c)$. Note that ℓ exists, since by definition of v_a, this node has leaves of S in at least two different child subtrees. Since $\mathcal{O}_\ell, \mathcal{O}_{\ell+1}$ are in different child subtrees of v_a, then $v_a = \text{lca}_{R_A}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. □

This proposition ensures that while examining all pairs of leaves $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ with $1 \leqslant \ell < m$, all nodes of R_A that are anchors of nodes in R_I are considered.

Now we know that all anchorable nodes v_i of R_I will be considered, as will all anchor nodes v_a of R_A. The following shows that, at some point, each such node in R_I is considered *at the same time* as its anchor v_a in R_A.

Proposition 2. *Let* $S = \{\mathcal{O}_{s_1}, \ldots, \mathcal{O}_{s_j}\} \subseteq L_\cap(\{R_A, R_I\})$ *be a set of consecutive leaves in* $\mathcal{O}$ *with* $\text{Card}(S) \geqslant 2$ *such that there exists a node* v_i *in* R_I *with* $v_i = \text{lca}_{R_I}(S)$*. Let* v_a *be the node of* R_A *such that* $v_a = \text{lca}_{R_A}(S)$*.*

Then ℓ *exists,* $s_1 \leqslant \ell < s_j$*, such that* $v_a = \text{lca}_{R_A}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ *and* $v_i = \text{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Proof. If $m = 2$ then S contains only one pair of leaves. Thus, for $\ell = 1$ we have $S = \{\mathcal{O}_\ell, \mathcal{O}_{\ell+1}\}$, so $\mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \mathrm{lca}_{R_I}(S)$ and $\mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \mathrm{lca}_{R_\mathrm{A}}(S)$.

Consider now the case where $m > 3$. Let ℓ with $s_1 \leqslant \ell \leqslant s_j$ be the smallest integer such that $\mathcal{O}_\ell$ and $\mathcal{O}_{\ell+1}$ belong to two different child subtrees of $v_a = \mathrm{lca}_{R_\mathrm{A}}(S)$. Such an ℓ exists because of Proposition 1. Now, $\{\mathcal{O}_\ell, \mathcal{O}_{\ell+1}\} \in L(v_i)$ by definition of ℓ and $S = \{\mathcal{O}_{s_1}, \ldots, \mathcal{O}_{s_j}\}$. More precisely, it can be shown that $v_i = \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. Indeed, if this is not the case, i.e. these two leaves are in the same child subtree of v_i, then let $\ell'' \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$ be a leaf in S belonging to another child subtree of v_i. We have $v_i = \mathrm{lca}_{R_I}(\ell'', \mathcal{O}_\ell) < \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. Since $R_a|L(R_i) \trianglerighteq R_i|L(R_A)$, this implies $\mathrm{lca}_{R_\mathrm{A}}(\ell'', \mathcal{O}_\ell) < \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ by Observation 1. This in turn implies $\ell'' \notin L(v_a)$ because $v_a = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. Hence, $L(v_a) \cap L_\cap(\{R_\mathrm{A}, R_I\}) \neq L(v_i) \cap L_\cap(\{R_\mathrm{A}, R_I\})$, which is in contradiction with the definition of v_a and Remark 3. This shows that $\mathcal{O}_\ell$ and $\mathcal{O}_{\ell+1}$ are in different child subtrees of v_i, hence that $v_i = \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. By definition of v_a and v_i, this shows that for this precise ℓ, we have both $\mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \mathrm{lca}_{R_I}(S)$ and $R_\mathrm{A}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \mathrm{lca}_{R_\mathrm{A}}(S)$. □

Now, given that $R_\mathrm{A}|L(R_I)$ refines $R_I|L(R_\mathrm{A})$, the same anchorable internal node $v_i \in R_I$ might be considered in several pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$, sometimes together with a node $v_a = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ that is not its anchor in R_A. However, the anchor of v_i can be easily identified as it is the closest to the root (i.e. the one with the minimum depth), among the examined candidate nodes v_a in R_A. More formally:

Remark 4. Let v_i be a node of R_I such that $v_i = \mathrm{lca}_{R_I}(S)$ for a set $S \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$ with $\mathrm{Card}\,(S) \geqslant 2$. Suppose $v_i = \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_i = \mathrm{lca}_{R_I}(\mathcal{O}_{\ell'}, \mathcal{O}_{\ell'+1})$ with $\ell \neq \ell'$. Let $v_a^\ell = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_a^{\ell'} = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_{\ell'}, \mathcal{O}_{\ell'+1})$. If $\mathrm{depth}(v_a^\ell) > \mathrm{depth}(v_a^{\ell'})$ then v_a^ℓ is not the anchor of v_i.

Indeed, since $\mathrm{depth}(v_a^\ell) > \mathrm{depth}(\mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_{\ell'}, \mathcal{O}_{\ell'+1}))$, then $\mathcal{O}_{\ell'}$ or $\mathcal{O}_{\ell'+1}$ is not in $L(v_a^\ell)$. As both $\mathcal{O}_{\ell'}$ and $\mathcal{O}_{\ell'+1}$ belong to $L(v_i)$, this means that $L(v_a^\ell) \cap L_\cap(\{R_\mathrm{A}, R_I\}) \neq L(v_i) \cap L_\cap(\{R_\mathrm{A}, R_I\})$, i.e. that v_a^ℓ is not the anchor of v_i (from Remark 3).

Based on the previous remarks, the pseudo-code ANCHORS (see Algorithm 3) shows how the anchors for all anchorable nodes of R_I are determined in $\mathrm{O}(n)$ time. The artificial anchor of the root of R_I (when needed, see Section 3.1) is also described.

Input: Two rooted trees R_I and R_A s.t $R_\mathrm{A}|L(R_I) \trianglerighteq R_I|L(R_\mathrm{A})$.
Result: Determines the nodes of R_A that are anchors for nodes in R_I.
***(i)* Anchoring leaves:**
1 **for** *each leaf node* $\ell \in R_I$ *with a label in* $L_\cap(\{R_\mathrm{A}, R_I\})$ **do**
  set $\mathsf{Anchor}(\ell)$ to the leaf-node of R_A having the same label.
***(ii)* Anchoring internal nodes:**
for *each internal node* $v_i \in R_I$ **do** $\mathsf{Anchor}(v_i) \leftarrow \emptyset$
Let $\mathcal{O}$ be the left-right ordering in the tree R_I of the leaves in set $L_\cap(\{R_\mathrm{A}, R_I\})$
2 **for** *each pair* $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ *of consecutive leaves in* $\mathcal{O}$ **do**
  Let $v_i := \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_a := \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$
  if $\mathsf{Anchor}(v_i) = \emptyset$ *or* $\mathit{depth}(\mathsf{Anchor}(v_i)) > \mathit{depth}(v_a)$ **then**
    $\mathsf{Anchor}(v_i) \leftarrow v_a$
***(iii)* Anchor roots of R_I and R_A:**
if $\mathsf{Anchor}(\mathrm{root}(R_I)) = \emptyset$ **then**
  if $\mathrm{root}(R_\mathrm{A})$ *has been anchored with a node of* R_I **then**
    add a new node as parent of $\mathrm{root}(R_\mathrm{A})$ and consider it as the new root of R_A
  $\mathsf{Anchor}(\mathrm{root}(R_I)) \leftarrow \mathrm{root}(R_\mathrm{A})$

Algorithm 3. ANCHORS(R_I, R_A).

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Appendix B. Complement for the proof of Theorem 1

We give here the proof that (4) holds for the special case where $\mathrm{Card}(\{\ell, \ell', \ell''\} \cap L(R)) = 2$. This completes the proof of Theorem 1. Without loss of generality, let ℓ', ℓ'' be the two leaves present in R before the graft of the copy of S_p and let ℓ be the leaf in S_p. If $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell, \ell'') = \mathrm{lca}_{R_I}(\ell', \ell'')$, then (4) is verified by default. The three remaining possibilities for the relative positions of ℓ, ℓ, ℓ'' in R_i are as follows:

(1) $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell, \ell'') < \mathrm{lca}_{R_I}(\ell', \ell'')$.
(2) $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell', \ell'') < \mathrm{lca}_{R_I}(\ell, \ell'')$.
(3) $\mathrm{lca}_{R_I}(\ell, \ell'') = \mathrm{lca}_{R_I}(\ell', \ell'') < \mathrm{lca}_{R_I}(\ell, \ell')$.

The proof is only detailed below for case 1, as a very similar reasoning applies for cases 2 and 3.

Let v_i be the node from which S_p is hanging in R_I. Note that $\mathrm{lca}_{R_I}(\ell, \ell') \leqslant v_i$, otherwise $\{\ell, \ell', \ell''\} \in L(S_p)$, a contradiction with $\mathrm{Card}(\{\ell, \ell', \ell''\} \cap L(R)) = 2$.

(I) Suppose $\mathrm{lca}_{R_I}(\ell, \ell') = v_i$. Note that this does not imply that v_i has an anchor in R because $\ell \notin R$.

(I-A) Suppose that v_i has an anchor v_a in R. Let c be the child of v_i such that $\{\ell', \ell''\} \in S(c)$. Either (i) $L(c)$ contains no leaf of $L_\cap(\{R_\mathrm{A}, R_I\})$, or (ii) it contains at least such a leaf, say x. Case (i) occurs only when $S(c)$ is a specific subtree of R_i with respect to R_A, which means that a copy of $S(c)$ (containing $\{\ell', \ell''\}$) is grafted under v_a by loop of line 2, just before S_p is grafted. Hence, ℓ', ℓ'' belong to a same subtree $S(c')$ of a child c' of v_a. We now show that the same holds in case (ii). Indeed, in that case, $x \in L(v_i)$ and because v_i is an anchored node, then $\exists y \in L(v_i)$ such that $y \in L_\cap(\{R_\mathrm{A}, R_I\})$ and $v_i = \mathrm{lca}_{R_I}(x, y)$. Since $\{x, \ell', \ell''\} \in S(c)$, we have

$$v_i = \mathrm{lca}_{R_I}(x, y) < \mathrm{lca}_{R_I}(x, \ell') \quad \text{and} \quad v_i = \mathrm{lca}_{R_I}(x, y) < \mathrm{lca}_{R_I}(x, \ell'').$$

Since $\{x, \ell', \ell''\} \in L(R)$, we have by induction that

$$\mathrm{lca}_R(x, y) < \mathrm{lca}_R(x, \ell') \quad \text{and} \quad \mathrm{lca}_R(x, y) < \mathrm{lca}_R(x, \ell'').$$

Moreover, by definition of v_a, we know $\{x, y\} \in L(v_a)$, thus $v_a \leqslant \mathrm{lca}_R(x, y)$, and then from the previous equation

$$v_a < \mathrm{lca}_R(x, \ell') \quad \text{and} \quad v_a < \mathrm{lca}_R(x, \ell'').$$

This means that $\{x, \ell', \ell''\}$ belong to a same subtree $S(c')$ in R, with c' being a child of v_a.

Thus, in both (i) and (ii), when a copy of S_p (containing ℓ) is inserted as a new child subtree of v_a, by loop of line 2, then Eq. (4) holds.

(I-B) The other sub-case is when v_i is not an anchored node of R_i. This means that a copy of S_p is grafted in R as a subtree of a node v_new by loop of line 4. This loop inserts subtrees at nodes above the anchor v_a of a node v of R_I. Note that in R_I, $v_i < v$, and that there is no node with an anchor on the path from v to v_i (because the loop of line 4 would end before reaching v_i). There are several possible places for ℓ' and ℓ'' relative to v_i and v_0.

First, ℓ' and ℓ'', can be in the same specific subtree S'_p hanging from v_i. Then $\ell', \ell'' \in L(R)$ ensures that a copy of S'_p is inserted as a subtree of v_new by loop of line 5, before the same loop inserts a copy of S_p. Thus in that case, ℓ', ℓ'' are in the same subtree of v_new before S_p is inserted. The second possibility for ℓ', ℓ'' is that they are in the subtree of v_i that contains v. Then ℓ', respectively ℓ'', can be in a specific subtree S'_p, respectively S''_p, hanging from a node on the path between v and v_i, (with possibly $S'_p = S''_p$). But then, the loop of line 4, proceeding in a bottom-up way, ensures that S'_p, respectively S''_p, is inserted in the subtree of v_new that contains v_a. Alternatively, ℓ' and ℓ'' (or just one of them) can be in $S(v)$. A similar argument[1] as in (I-A) shows that in this case they are inserted in the subtree $S(v_a)$ of R. Thus, in this case also, they belong to the subtree of v_new containing v_a.

Thus, in all possible positions of ℓ', ℓ'' relative to v_i and v, they are grafted in R the same subtree of v_new. This means that when a copy of S_p (containing ℓ) is inserted as a different child subtree of v_new to obtain R', then (4) holds.

(II) The other main sub-case arises when $\mathrm{lca}_{R_I}(\ell, \ell') < v_i$. Let $u := \mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell, \ell'')$. Note that ℓ' and ℓ'' belong to the same child subtree of u, differing from that containing ℓ.

[1] Based on leaves $\{x, y\} \in L(v_a) \cap L_\cap(\{R_\mathrm{A}, R_I\})$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Let v be the node v_i, if v_i is anchored, otherwise it is the closest descendant of v_i that is anchored (v exists otherwise, $S(v_i)$ would be a specific subtree, contradicting the maximality of the specific subtree S_p). Moreover, there is a unique closest descendant of v_i that is anchored because, if two nodes are anchored, then their least common ancestor is also anchored).

Let v_a be the anchor of v in R. By definition of v and v_a, $x, y \in L_\cap(\{R_A, R_I\})$ exist such that $v = \mathrm{lca}_{R_I}(x, y)$ and $v_a = \mathrm{lca}_{R_A}(x, y)$. Note that $u = \mathrm{lca}_{R_I}(\ell', x) = \mathrm{lca}_{R_I}(\ell'', x) < \mathrm{lca}_{R_I}(x, y)$ because x, y are in the child subtree of u containing v_i, different from the one containing ℓ', ℓ''. Hence,

$$\mathrm{lca}_{R_I}(\ell', \ell'') > \mathrm{lca}_{R_I}(\ell', x) = \mathrm{lca}_{R_I}(\ell'', x) \quad \text{and}$$
$$\mathrm{lca}_{R_I}(x, y) > \mathrm{lca}_{R_I}(\ell', x) = \mathrm{lca}_{R_I}(\ell'', x).$$

As $\{\ell', \ell'', x, y\} \in L(R)$, induction applies to obtain

$$\mathrm{lca}_{R_A}(\ell', \ell'') > \mathrm{lca}_{R_A}(\ell', x) = \mathrm{lca}_{R_A}(\ell'', x) \quad \text{and}$$
$$\mathrm{lca}_{R_A}(x, y) = v_a > \mathrm{lca}_{R_A}(\ell', x) = \mathrm{lca}_{R_A}(\ell'', x).$$

Let v'_a be the node $\mathrm{lca}_{R_A}(\ell', x)$ of R_A. Previous equations indicate that v'_a has different children $c_{\ell'}$ and c_x, such that $\{\ell', \ell''\} \in S(c_{\ell'})$ and $x, y \in S(c_x)$. The node $v_a = \mathrm{lca}_{R_A}(x, y)$, anchor of v, is thus in $S(c_x)$. Moreover, by definition of v, the copy of S_p is inserted either as a new child subtree of v_a (in the case where $v = v_i$), either between v_a and its parent in R. In both cases, ℓ is inserted in $S(c_x)$. Thus in tree R', ℓ is in a subtree of v'_a differing from that containing ℓ', ℓ'', hence (4) holds.

Appendix C. Proof of Lemma 1

Proof. The three statements of the lemma are related to the call to MERGETREES issued in line 2 of the loop of BUILDSMCT, statement (A) applying before the execution of a call, while statements (B) and (C) apply to the result of this call. Thus, the three statements are strongly inter-dependent and we prove them by a joint induction. However, note that there is no circularity as

- the proof of (A) uses inductive hypothesis on previous iterations of statements (B) and (C), except for the basic step which is proved independently;
- the proof of (B) uses statement (A) of the same iteration and, except for the basic step, inductive hypothesis on the previous iteration of (B);
- the proof of (C) uses statements (A) and (B) of the same iteration and, except for the basic step, inductive hypothesis on the previous iteration of (C).

The basic step ($i = 1$) of each statement is proved as following:

(A) When $i = 1$, $R_I = R_1|(L(R^0_M) \cup L_S(R_1))$ and $R_A = R^0_M$. As $R^0_M = MCT(\mathcal{R}|L(\mathcal{R}))$, we have

$$L(R^0_M) \subseteq L_\cap(\mathcal{R}) \quad \text{and} \tag{C.1}$$
$$R^0_M \trianglerighteq R_1|L_\cap(\mathcal{R})|L(R^0_M). \tag{C.2}$$

From (C.1) we have $L(R^0_M) \subseteq L(R_1)$, thus

$$R^0_M = R^0_M|L(R_1) \quad \text{and} \tag{C.3}$$
$$\big(R_1|L_\cap(\mathcal{R})\big)|L(R^0_M) = R_1|L(R^0_M). \tag{C.4}$$

Rewriting (C.2) thanks to (C.3) and (C.4) gives $R_A|L(R_I) \trianglerighteq R_I|L(R^0_M) = R_I|L(R_A)$, the last equality resulting from the remark following Definition 1.

(B) From the basic step of (A), we know that Theorem 1 applies when MERGETREES is called. Thus, the tree R^1_M returned by this call is such that $L(R^1_M) = (L(R_1) \cap (L(R^0_M) \cup L_S(R_1))) \cup L(R^0_M)$, which simplifies into $L(R^0_M) \cup L_S(R_1)$ since $L_\Delta(\mathcal{R}) = \emptyset$.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

(C) From the basic step of statement (A) and Theorem 1 we know R_M^1 is a tree $SMCT(R_I, R_A)$, i.e.

$$R_M^1|L(R_I) \trianglerighteq R_I|L(R_M^1) \quad \text{and} \quad R_M^1|L(R_A) \trianglerighteq R_A|L(R_M^1) \tag{C.5}$$

with $R_I = R_1|(L(R_M^0) \cup L_S(R_1))$ and $R_A = R_M^0$. Moreover, from the basic step of statement (B), $L(R_M^1) = L(R_M^0) \cup L_S(R_1)$, thus for all $R_j \in \mathcal{R}$, $j > 1$, we have from (C.5) that

$$R_M^1|L(R_M^0) \trianglerighteq R_M^0|L(R_M^1) = R_M^0 \trianglerighteq R_j|L(R_M^0), \tag{C.6}$$

where the rightmost refinement relation results from the definition of R_M^0. Statement (B), $L_\Delta(\mathcal{R}) = \emptyset$, and $L(R_M^0) \subseteq L_\cap(\mathcal{R})$ imply $L(R_M^1) \cap L(R_M^0) = L(R_M^0) = L(R_j) \cap L(R_M^0) = L(R_M^1) \cap L(R_j)$ for all $R_j \in R$, $j > 1$. Thus, (C.6) rewrites as

$$R_M^1|L(R_j) \trianglerighteq R_j|L(R_M^1), \quad \forall j > 1. \tag{C.7}$$

From (C.5) we also have

$$R_M^1|L(R_1) \cap \big(L(R_M^0) \cup L_S(R_1)\big) \trianglerighteq R_1|L(R_M^0) \cup L_S(R_1)|L(R_M^1)$$

which rewrites as $R_M^1|L(R_1) \trianglerighteq R_1|L(R_M^1)$ using statement (B). Together with (C.7), this proves the basic step of statement (C).

Now suppose statements (A), (B) and (C) hold for the first $i-1$ iterations of the loop in line 2 and consider its ith iteration, with $i > 1$.

(A) We first rewrite $L(R_A)$, $L(R_M^i)$ and $L(R_I)$:

$$L(R_A) = L(R_M^{i-1}) = L(R_M^0) \cup \bigcup_{j<i} L_S(R_i), \tag{C.8}$$

$$L(R_i) = L_\cap(\mathcal{R}) \cup L_S(R_i), \tag{C.9}$$

$$L(R_I) = L(R_i) \cap \big(L(R_M^0) \cup L_S(R_i)\big) = L(R_M^0) \cup L_S(R_i), \tag{C.10}$$

where (C.8) results by inductive hypothesis from statement (B), (C.9) results from $L_\Delta(\mathcal{R}) = \emptyset$, and (C.10) follows from the fact that $L(R_M^0) \subseteq L_\cap(\mathcal{R})$ and $L_S(R_i)$ are both included into $L(R_i)$, as (C.9) shows. This three equations show that

$$L(R_I) \cap L(R_A) = L(R_M^0) = L(R_i) \cap L(R_A). \tag{C.11}$$

By inductive hypothesis, statement (C) says that R_M^{i-1} is a supertree compatible with $\mathcal{R}$, i.e. $R_M^{i-1}|L(R_i) \trianglerighteq R_i|L(R_M^{i-1})$, which rewrites as $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$ by definition of R_A, R_I and use of (C.11).

(B) From statement (A) and Theorem 1, the call to MERGETREES(R_I, R_A) performed at iteration i of the loop in BUILDSMCT with trees $R_I = R_i|(L(R_M^{i-1}) \cup L_S(R_i))$ and $R_A = R_M^{i-1}$ returns a tree R_M^i such that

$$L(R_M^i) = L(R_I) \cup L(R_A). \tag{C.12}$$

It is easy top see that $L(R_I) = L(R_M^0) \cup L_S(R_i)$ and that, by inductive hypothesis of (B), $L(R_A) = L(R_M^{i-1}) = L(R_M^0) \cup \bigcup_{j<i} L_S(R_j)$. Thus, from (C.12) we obtain $L(R_M^i) = L(R_M^0) \bigcup_{j\leqslant i} L_S(R_j)$.

(C) From statement (A) for the ith iteration, we know Theorem 1 applies to the call MERGETREES(R_I, R_A) performed at that iteration with trees $R_I = R_i|(L(R_M^0) \cup L_S(R_i))$ and $R_A = R_M^{i-1}$. Thus, the tree R_M^i is a $SMCT(R_I, R_A)$, i.e.

$$R_M^i|L(R_I) \trianglerighteq R_I|L(R_M^i) \quad \text{and} \quad R_M^i|L(R_A) \trianglerighteq R_A|L(R_M^i). \tag{C.13}$$

Moreover, statement (B) used for the ith iteration says that

$$L(R_M^i) = L(R_M^0) \cup \bigcup_{j\leqslant i} L_S(R_i) = L(R_M^{i-1}) \cup L_S(R_i). \tag{C.14}$$

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

Consider the case of trees $R_j \in \mathcal{R}$ with $j \neq i$, for which

$$L(R_j) \cap L(R_M^i) = L(R_j) \cap L(R_M^{i-1}), \tag{C.15}$$

from (C.14) and the fact that specific leaves of R_i do not appear in other input trees. By inductive hypothesis of statement (C),

$$R_M^{i-1}|L(R_j) \trianglerighteq R_j|L(R_M^{i-1}) = R_j|L(R_M^i). \tag{C.16}$$

From the second part of (C.13) we know another refinement relation:

$$R_M^i|L(R_M^{i-1}) \trianglerighteq R_M^{i-1}|L(R_M^i) = R_M^{i-1}, \tag{C.17}$$

the equality resulting from $L(R_M^{i-1}) \subseteq L(R_M^i)$ (from (C.14)). Thus, reducing both sides of (C.17) to leaves of a tree R_j, $j \neq i$, we obtain

$$R_M^i|L(R_M^{i-1})|L(R_j) \trianglerighteq R_M^{i-1}|L(R_j),$$

the left part of which rewrites as $(R_M^i|L(R_j))|L(R_M^{i-1}) = R_M^i|L(R_j)$ from (C.15). Combining this refinement relation with the one stated in (C.16) we obtain by transitivity that

$$R_M^i|L(R_j) \trianglerighteq R_j|L(R_M^i), \quad \forall j \neq i. \tag{C.18}$$

Now consider the case of the input tree $R_i \in \mathcal{R}$. Since $L_\Delta(\mathcal{R}) = \emptyset$, i.e. $L(R_i) = L_\cap(\mathcal{R}) \cup L_S(R_i)$, and $L(R_M^0) \subseteq L_\cap(\mathcal{R})$, we obtain

$$L(R_i) \cap L(R_M^i) = L(R_M^0) \cup L_S(R_i) \tag{C.19}$$

from (C.14). The refinement relation stated in the first part of (C.13) can be rewritten as

$$R_M^i|L(R_i) \cap \big(L(R_M^0) \cup L_S(R_i)\big) \trianglerighteq R_i|L(R_M^0) \cup L_S(R_i)|L(R_M^i)$$

which can be simplified into $R_M^i|L(R_i) \trianglerighteq R_i|L(R_M^i)$ using (C.19). Together with (C.18), this proves statement (C) for the ith iteration. □

References

[1] A.V. Aho, Y. Sagiv, T.G. Szymanski, J.D. Ullman, Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions, SIAM Journal on Computing 10 (3) (1981) 405–421.
[2] A. Amir, D. Keselman, Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithm, SIAM Journal on Computing 26 (6) (1997) 1656–1669.
[3] B.R. Baum, Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees, Taxon 41 (1992) 3–10.
[4] B.R. Baum, M.A. Ragan, The MRP method, in: O.R.P. Bininda-Emonds (Ed.), Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life, Kluwer, Dordrecht, 2004, pp. 17–34.
[5] M. Bellare, S. Goldwasser, C. Lund, A. Russeli, Efficient probabilistically checkable proofs and applications to approximations, in: Proc. of the 25th Annual A.C.M. Symposium on Theory of Computing (STOC'93), 1993, pp. 294–304.
[6] T. Berger-Wolf, Online consensus and agreement of phylogenetic trees, in: Proc. of the 4th Workshop on Algorithms in Bioinformatics (WABI'04), in: Lecture Notes in Computer Science, vol. 3240, Springer, Berlin, 2004, pp. 350–361.
[7] V. Berry, S. Guillemot, F. Nicolas, C. Paul, On the approximation of computing evolutionary trees, in: L. Wang (Ed.), Proc. of the 11th Annual International Conference on Computing and Combinatorics (COCOON'05), in: Lecture Notes in Computer Science, vol. 3595, Springer, Berlin, 2005, pp. 115–125.
[8] V. Berry, F. Nicolas, Improved parametrized complexity of the maximum agreement subtree and maximum compatible tree problems, IEEE/ACM Transactions on Computational Biology and Bioinformatics 3 (3) (2006) 289–302.
[9] O.R.P. Bininda-Edmonds (Ed.), Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life, Computational Biology Series, vol. 4, Kluwer Academic, Dordrecht, 2004.
[10] O.R.P. Bininda-Edmonds, J.L. Gittleman, M.A. Steel, The (super)tree of life: procedures, problems, and prospects, Annual Review of Ecology and Systematics 33 (2002) 265–289.
[11] O.R.P. Bininda-Edmonds, M.J. Sanderson, Assessment of the accuracy of matrix representation with parsimony analysis supertree construction, Systematic Biology 50 (4) (2001) 565–579.
[12] D. Bryant, Building trees, hunting for trees and comparing trees: theory and method in phylogenetic analysis, PhD thesis, University of Canterbury, Department of Mathematics, 1997.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

[13] D. Bryant, M.A. Steel, Extension operations on sets of leaf-labelled trees, Advances in Applied Mathematics 16 (4) (1995) 425–453.
[14] R. Cole, M. Farach-Colton, R. Hariharan, T.M. Przytycka, M. Thorup, An O($n \log n$) algorithm for the Maximum Agreement SubTree problem for binary trees, SIAM Journal on Computing 30 (5) (2001) 1385–1404.
[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, Cambridge, MA, 2001.
[16] U. Feige, M.M. Halldórsson, G. Kortsarz, Approximating the domatic number, in: Proc. of the 32nd Annual A.C.M. Symposium on Theory of Computing (STOC'00), 2000, pp. 134–143.
[17] C.R. Finden, A.D. Gordon, Obtaining common pruned trees, Journal of Classification 2 (1985) 255–276.
[18] G. Ganapathy, T.J. Warnow, Approximating the complement of the maximum compatible subset of leaves of k trees, in: Proc. of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02), 2002, pp. 122–134.
[19] G. Ganapathysaravanabavan, T.J. Warnow, Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time, in: O. Gascuel, B.M.E. Moret, (Eds.), Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI'01), 2001, pp. 156–163.
[20] L. Gąsieniec, J. Jansson, A. Lingas, A. Östlin, On the complexity of constructing evolutionary trees, Journal of Combinatorial Optimization 3 (2–3) (1999) 183–197.
[21] A.G. Gordon, Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labelled leaves, Journal of Classification 3 (1986) 335–348.
[22] S. Guillemot, F. Nicolas, Tight bounds on the complexity of the maximum agreement subtree and maximum compatible tree problems, Technical Report, LIRMM, Univ. of Montpellier, 2005.
[23] A. Gupta, N. Nishimura, Finding largest subtrees and smallest supertrees, Algorithmica 21 (2) (1998) 183–210.
[24] A.M. Hamel, M.A. Steel, Finding a maximum compatible tree is NP-hard for sequences and trees, Applied Mathematics Letters 9 (2) (1996) 55–59.
[25] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestor, SIAM Journal on Computing 13 (2) (1984) 338–355.
[26] J. Hein, T. Jiang, L. Wang, K. Zhang, On the complexity of comparing evolutionary trees, Discrete Applied Mathematics 71 (1–3) (1996) 153–169.
[27] J. Jansson, A. Lingas, E.-L. Lundell, A triplet approach to approximations of evolutionary trees, in: Proc. of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB'04), 2004.
[28] J. Jansson, J.H.-K. Ng, K. Sadakane, W.-K. Sung, Rooted maximum agreement supertrees, Algorithmica 43 (4) (2005) 293–307.
[29] M.-Y. Kao, T.W. Lam, W.-K. Sung, H.-F. Ting, A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees, in: Proc. of the 7th Annual European Symposium on Algorithms (ESA'99), 1999, pp. 438–449.
[30] M.-Y. Kao, T.W. Lam, W.-K. Sung, H.-F. Ting, An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings, Journal of Algorithms 40 (2) (2001) 212–233.
[31] D.T.J. Littlewood, R.A. Bray (Eds.), Interrelationships of the Platyhelminthes, Systematics Association Special Volumes, vol. 60, CRC Press, London, 2000 (Chapter 27, Towards a phylogenetic supertree of Platyhelminthes?).
[32] R.D.M. Page, Modified mincut supertrees, in: R. Guigó, D. Gusfield, (Eds.), Proc. of the 2nd International Workshop on Algorithms in Bioinformatics (WABI'02), 2002, pp. 537–552.
[33] Z.S. Peng, H.F. Ting, An O($n \log n$)-time algorithm for the maximum constrained agreement subtree problem for binary trees, in: Proc. of the 15th International Symposium on Algorithms and Computations (ISAAC'04), 2004, pp. 754–765.
[34] M.A. Ragan, Matrix representation in reconstructing phylogenetic relationships among the eukaryots, Biosystems 28 (1–3) (1992) 47–55.
[35] V. Ranwez, V. Berry, S. Guillemot, A. Criscuolo, E.J.P. Douzery, Vote or veto: desirable properties for supertree methods, LIRMM, ISEM, Université Montpellier 2, 2006, submitted for publication.
[36] C. Semple, M. Steel, Phylogenetics, Oxford Lecture Series in Mathematics and its Applications, vol. 24, Oxford University Press, Oxford, 2003.
[37] J.L. Thorley, M. Wilkinson, A view of supertrees methods, in: M.F. Janowitz, H.F.-J. Lapointe, F.R. McMorris, F.S. Roberts (Eds.), Bioconsensus, in: Discrete Mathematics and Theoretical Computer Science, vol. 61, DIMACS, 2003, pp. 185–194.
[38] http://tolweb.org/tree/phylogeny.html.

Please cite this article as: V. Berry, F. Nicolas, Maximum agreement and compatible supertrees, Journal of Discrete Algorithms (2006), doi:10.1016/j.jda.2006.08.005

.

Syst. Biol. 56(5):798–817, 2007

ISSN: 1063-5157 print / 1076-836X online
DOI: 10.1080/10635150701639754

PhySIC: A Veto Supertree Method with Desirable Properties

VINCENT RANWEZ,[1] VINCENT BERRY,[2] ALEXIS CRISCUOLO,[1,2] PIERRE-HENRI FABRE,[1] SYLVAIN GUILLEMOT,[2] CELINE SCORNAVACCA,[1,2] AND EMMANUEL J. P. DOUZERY[1]

[1] *Institut des Sciences de l'Evolution (ISEM, UMR 5554 CNRS), Université Montpellier II, Place E. Bataillon, CC 064, 34095 Montpellier Cedex 5, France; E-mail: ranwez@isem.univ-montp2.fr (V.R.)*

[2] *Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM,UMR 5506, CNRS), Université Montpellier II 161, rue Ada, 34392 Montpellier Cedex 5, France*

Abstract.— This paper focuses on veto supertree methods; i.e., methods that aim at producing a conservative synthesis of the relationships agreed upon by all source trees. We propose desirable properties that a supertree should satisfy in this framework, namely the *non-contradiction* property (PC) and the *induction* property (PI). The former requires that the supertree does not contain relationships that contradict one or a combination of the source topologies, whereas the latter requires that all topological information contained in the supertree is present in a source tree or collectively induced by several source trees. We provide simple examples to illustrate their relevance and that allow a comparison with previously advocated properties. We show that these properties can be checked in polynomial time for any given rooted supertree. Moreover, we introduce the *PhySIC* method (PHYlogenetic Signal with Induction and non-Contradiction). For k input trees spanning a set of n taxa, this method produces a supertree that satisfies the above-mentioned properties in $O(kn^3 + n^4)$ computing time. The polytomies of the produced supertree are also tagged by labels indicating areas of conflict as well as those with insufficient overlap. As a whole, *PhySIC* enables the user to quickly summarize consensual information of a set of trees and localize groups of taxa for which the data require consolidation. Lastly, we illustrate the behaviour of *PhySIC* on primate data sets of various sizes, and propose a supertree covering 95% of all primate extant genera. The *PhySIC* algorithm is available at http://atgc.lirmm.fr/cgi-bin/PhySIC. [Formal properties; phylogenetics; polynomial-time algorithms; primates; software; supertrees; triplets; veto methods.]

Building Supertrees

Phylogenies are invaluable tools in various areas of biology to understand the evolution of genes and taxa. Trees that incorporate an exhaustive sampling of taxonomic biodiversity provide crucial information about systematics, genomics, and diversification patterns of species (e.g., Davies et al., 2004). Large trees can be built using various approaches, including supermatrices and supertrees. The former approach consists of combining the different source data sets into a *supermatrix* of characters and then analyzing it under standard phylogenetic reconstruction criteria (e.g., Delsuc et al., 2005). The supertree approach is an alternative methodology using trees rather than character data as a primary source of information. It first involves inferring partially overlapping, source phylogenetic trees from initial character data and then assembling them into a larger, more comprehensive *supertree* (Bininda-Emonds, 2004a). This approach is particularly convenient when dealing with heterogeneous character sources; e.g., those scored from morphological, transposable elements, DNA, or protein studies. Supertrees have become increasingly popular (e.g., Bininda-Emonds, 2004b), notably since the seminal work involving the reconstruction of the primate supertree (Purvis, 1995a). The widespread use of supertrees is explained by three useful applications (Wilkinson et al., 2004): (i) they provide large phylogenetic frameworks for broad comparative studies; (ii) they evaluate the congruence of sets of input trees, and reveal conflicts due to outlier/unstable taxa; and (iii) they identify insufficient overlap among leaf sets of input trees and assign priorities for choosing the taxa to be subsequently sampled.

Different Kinds of Supertree Methods

Supertree methods fall into three categories depending on their way of handling topological conflicts; i.e., different arrangements of the same leaves among labeled source trees.

The first suite of methods does not handle incompatible source trees. The pioneering methods that belong to this category are *Build* (Aho et al., 1981) and the strict consensus supertree (Gordon, 1986). Although they are important milestones, these methods appear "of limited use. As most systematists know, phylogenies usually conflict with one another" (Bininda-Emonds, 2004b:4).

The second suite of methods handles conflicts among input trees in a liberal way: they apply a *voting* procedure. In order to extract their main phylogenetic signal, source trees are asked to vote on various parts of the phylogeny to be inferred, with the most supported candidates being elected and composing the output supertree. Voting methods are said to *resolve* conflicts (Thorley and Wilkinson, 2003): for each conflict, they use some optimization criterion to make a decision in favor of one of the topological alternatives. Most conflicts among input trees are expected to be resolved because relationships displayed by the supertree are guided by source topologies on the basis of weight of evidence. The most widespread voting method is matrix representation with parsimony (MRP) whereby nodes of each source tree are encoded as binary characters of a matrix, which is then analyzed with the maximum

parsimony criterion to obtain the composite tree (Baum, 1992; Ragan, 1992). Analyzing this binary encoding of source topological information with other tree-building criteria leads to variants of MRP such as matrix representation with flipping (MRF; Chen et al., 2003) and matrix representation with compatibility (MRC; Ross and Rodrigo, 2004). Other methods of the voting kind, such as MinCut (MC; Semple and Steel, 2000) and Modified-MinCut (MMC; Page, 2002) extend *Build*. They encode source trees in a graph that is progressively decomposed to get supertree clades. When conflicts hinder the decomposition, the graph is cut by removing the least supported relationships. The Average Consensus Supertree (Lapointe and Cucumel, 1997) and super distance matrix (Criscuolo et al., 2006) methods implement the voting approach in an alternative way. They average the initial distance matrices, converted from source characters or valued topologies, into a *superdistance* matrix; a tree-building distance-based approach is then used to infer a supertree from this matrix. Interestingly, voting methods like MRP may generate *novel* clades; i.e., clades not present in any input tree alone (Purvis, 1995b; Bininda-Emonds and Bryant, 1998; Sanderson et al., 1998). Unfortunately, when source trees conflict, novel clades that are contradicted by each of the source trees can be present in the supertree inferred by MRP (Goloboff and Pol, 2002; Goloboff, 2005; Cotton et al., 2006) and by MRF (Goloboff, 2005). The importance of this phenomenon is still debated, Bininda-Emonds (2003) reporting, on the basis of simulations, that this situation is not very frequent for MRP, whereas Goloboff (2005) shows selected case studies where "this situation is, clearly, not very unlikely."

The third suite of methods handles conflicts among input trees in a conservative way. They adopt a *veto* philosophy: the phylogenetic information of every source topology is to be respected, and the supertree is not allowed to contain clades that a source tree would vote against. These methods *remove* conflicts (Thorley and Wilkinson, 2003) because they either propose multifurcations in the supertree (Goloboff and Pol, 2002) or prune rogue taxa (Berry and Nicolas, 2004). In this framework, the supertree should not retain a single branching pattern within a given clade when several valid topological alternatives are present in the source trees. The full agreement required by veto methods provides an unambiguous phylogenetic framework that is, for instance, well suited for taxonomic revisions. More specifically, such a conservative approach may be applied to automatically build or update parts of the Tree of Life (http://tolweb.org). Several supertree methods akin to the veto philosophy have been proposed, all of which are inspired by consensus approaches that operate on trees with identical leaf sets. For example, extensions of the strict consensus (Gordon, 1986; Huson et al., 1999), semi-strict consensus (Goloboff and Pol, 2002), and maximum agreement subtree consensus (Berry and Nicolas, 2004) have been proposed to infer veto supertrees.

Properties of Supertree Methods

To assess the relevance of supertree methods, it is most useful to have properties characterizing the extent to which the supertrees they infer are reliable syntheses of source trees (Bininda-Emonds and Bryant, 1998; Steel et al., 2000; Wilkinson et al., 2004; Goloboff, 2005). For instance, Steel et al. (2000) suggest that the output supertree should (i) encompass every source tree when possible, (ii) always contain every leaf (taxon) that occurs in at least one source tree, and (iii) be computed under a running time that grows polynomially with respect to the total number of leaves. These authors also showed that rooted input trees are more appealing than unrooted ones for supertree methods that aim to satisfy several desirable properties simultaneously. Yet, even if supertree methods satisfy some desirable properties, the inferred supertrees often contain polytomies that actually intermix two distinct phenomena: either a lack of overlap in the topological information among source trees, or the occurrence of topological conflicts among them, or a combination of these. We thus decided to develop a method that proposes supertrees with unambiguous resolutions, and provides biologists with explanations about causes of polytomies. For this purpose, we rely on two new formal properties.

On the one hand, we think that supertree methods should avoid arbitrary resolutions; i.e., resolutions that are not entailed by the source topologies. Indeed, novel relationships displayed by a supertree "are worrying if they are not implied by combinations of the input trees" (Wilkinson et al., 2005), and "should be identified as such, to highlight their lack of any known justification" (Pisani and Wilkinson, 2002). Thus, we first request that every piece of phylogenetic information displayed in the supertree be present in one or several source topologies or be induced by their interaction; we call this the *induction* property.

On the other hand, we focus on unanimous clades, thus adopting a veto point of view. This means that the supertree is not allowed to contain a clade that conflicts either directly with a source tree or indirectly with a combination of them. We call this the *non-contradiction* property. Such a supertree, which incorporates only uncontradicted input relationships, provides a reliable baseline for subsequent analyses (Goloboff and Pol, 2002; Goloboff, 2005).

Goloboff and Pol (2002) mentioned similar properties in a formal characterization involving triplets. They provide examples showing that supertree methods of the voting kind, such as MRP and MC, understandably do not respect these properties. Although being appealing, the characterization proposed by Goloboff and Pol (2002) can at times be too restrictive or permissive (see following sections). Recently, Grunewald et al. (2006) provided another characterization of a property related to arbitrary resolutions contained in the supertree with respect to source trees. In both cases, there does not seem to be any straightforward algorithm that would always allow

for property verification. Note, however, that Goloboff and Pol (2002) proposed a supertree heuristic algorithm that satisfies the desired properties in most cases.

In this paper, we provide a characterization of non-contradiction and induction properties that differs from those of Goloboff and Pol (2002) and Grunewald et al. (2006). We also describe simple and polynomial-time algorithms that enable users to check whether or not a given supertree satisfies these properties. Then we propose an algorithm called *PhySIC* that improves the *Build* algorithm (Aho et al., 1981) by always inferring a supertree and which, moreover, satisfies the non-contradiction and induction properties. As far as we know, this is the first time that a polynomial-time method is proposed that always satisfies properties related to induction and non-contradiction. Moreover, improving the behavior of *Build* with respect to arbitrary decisions can benefit the various methods that extend this algorithm for supertree purposes; e.g., MC (Semple and Steel, 2000), MCC (Page, 2002), *AncestralBuild* (Daniel and Semple, 2004; Berry and Semple, 2006), and *RankedTree* (Bryant et al., 2004). Next, we pinpoint the difference between the behavior of *PhySIC* and that of well-known supertree methods on a biological case study on Primates. Lastly, we illustrate *PhySIC* on the reconstruction of the primate supertree at the genus level from various source trees based on mitochondrial DNA, nuclear DNA, and jumping gene sequences. The supertree reconstructed appears to be useful for displaying phylogenetic relationships among the major primate taxa (Goodman et al., 2005). Moreover, the produced supertree displays label(s) on each of its polytomous nodes that identify the cause(s) of these polytomies (lack of cross-information and/or presence of contradictions). The *PhySIC* method has been implemented in `C++` using the `Bio++` library (Dutheil et al., 2006) and is freely available as a web service and for download at http://atgc.lirmm.fr/cgi-bin/PhySIC/physic.cgi.

Non-Contradiction and Induction Properties

We first introduce vocabulary and notations required to formally define the properties of *non-contradiction* (PC) and of *induction* (PI). Simple examples are then used to illustrate the relevance of PC and PI as well as to relate them with previously proposed properties for supertree methods (Steel et al., 2000; Goloboff and Pol, 2002). Then we show how to check in polynomial time whether a supertree satisfies PC and PI for a given collection of source trees.

Topological Description of Trees

The definitions and notations used for trees and their topological description are mainly the same as those used by Semple and Steel (2003). We only consider rooted phylogenies, due to the fact that supertree methods cannot fulfill different desirable properties listed in Steel et al. (2000) when considering unrooted trees. Hereafter, the terms *phylogeny* and *tree* are considered synonymous. Given a tree T, $L(T)$ denotes the set of taxa associated to its leaves. More generally, given a collection $\mathcal{T}$ of trees, $L(\mathcal{T})$ denotes the set of taxa appearing in at least one tree of $\mathcal{T}$. Given two phylogenies T and T' on the same leaf set ($L(T) = L(T')$), we say that T refines T' whenever T contains all clades of T'. In other words, either T and T' are identical or T can be transformed into T' by collapsing some of its internal edges.

A rooted tree on three leaves A, B, C has only three possible binary shapes, called *triplets* and denoted by $AB|C$, respectively $AC|B$, respectively $BC|A$, depending on the innermost clade (AB, respectively AC, respectively BC). Given a triplet t, $\bar{t}$ denotes any of the two other triplets on the same set of leaves. Alternatively, a tree on three leaves can be a star tree; i.e., a unique internal node connected to the leaves. Any rooted tree T can be equivalently described by the set of triplets homeomorphic to subtrees of T connecting three leaves (e.g., Grunewald et al., 2006); $rt(T)$ denotes this set. Given a collection $\mathcal{T}$ of phylogenies, $rt(\mathcal{T}) = \bigcup_{T_i \in \mathcal{T}} rt(T_i)$ denotes the set of triplets present in these phylogenies. Note that it is possible that $rt(\mathcal{T})$ contains two triplets t and $\bar{t}$, namely when $\mathcal{T}$ hosts two incompatible phylogenies. Clearly, two such triplets cannot be combined into a single supertree of the collection.

Given a set $\mathcal{R}$ of triplets, $L(\mathcal{R})$ denotes the set of taxa appearing in at least one tree in $\mathcal{R}$. A tree T is said to *display* a set $\mathcal{R}$ of triplets when $\mathcal{R} \subseteq rt(T)$; moreover, T *strictly displays* $\mathcal{R}$ if additionally $L(T) = L(\mathcal{R})$. A set $\mathcal{R}$ of triplets is *compatible* if there is a tree T that displays $\mathcal{R}$. To find a tree displaying $\mathcal{R}$, it is useful to take into account that some triplets of the tree are *induced* by $\mathcal{R}$: a compatible set $\mathcal{R}$ of triplets *induces* a triplet t, denoted by $\mathcal{R} \vdash t$, if and only if $\mathcal{R} \cup \{\bar{t}\}$ is not compatible, or equivalently if any tree T that displays $\mathcal{R}$ contains t. For instance, any tree displaying $\{AB|C, BC|D\}$ also has to display the triplet $AC|D$; i.e., $\{AB|C, BC|D\} \vdash AC|D$. Bandelt and Dress (1986) and Dekker (1986) were among the first to investigate such induction rules. The set of all triplets induced by a compatible set $\mathcal{R}$ is called the *closure* of $\mathcal{R}$ and is denoted by $cl(\mathcal{R})$. Source trees considered for supertree building are sometimes incompatible, and then the set of triplets considered is incompatible. Nonetheless, we can characterize the set of triplets induced by these collections by extending the preceding definition: we will say that a set $\mathcal{R}$ of triplets *induces* a triplet t when there is a compatible subset $\mathcal{R}'$ of $\mathcal{R}$ that induces t.

Characterizing Non-Contradiction and Induction by Triplets

Here we describe two important properties that veto method supertrees should satisfy. They concern topological relationships that a supertree should not contain with respect to the input trees: first, it should not contain relationships *contradicting* the source trees (PC property); moreover, it should only contain relationships that are *induced* by the input trees (PI property). Below we detail these two properties.

There are several ways for a supertree to contradict a collection of source trees. The most direct contradiction

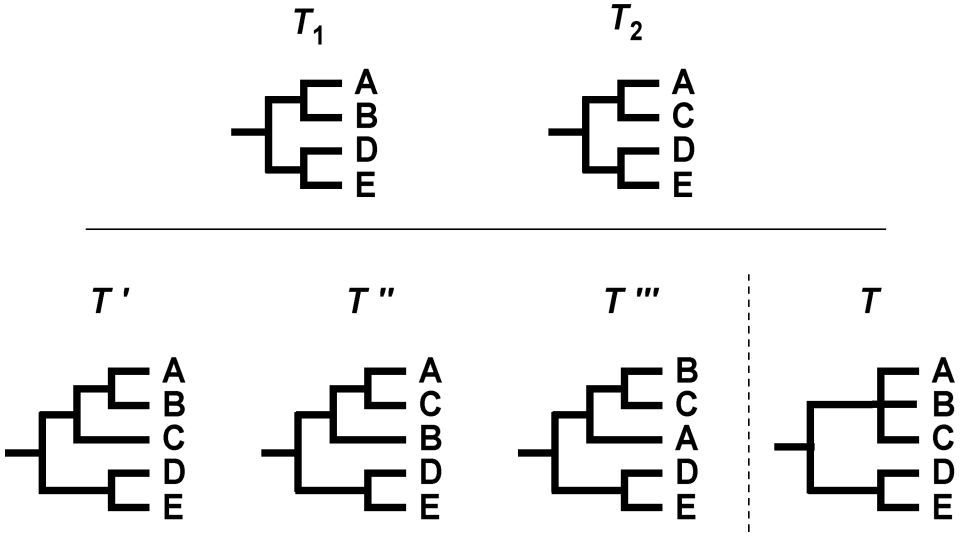

FIGURE 1. Supertrees can contain arbitrary resolution. Example of a collection $\mathcal{T} = \{T_1, T_2\}$ of two source trees and several possible supertrees T', T'', T''', and T. Unlike T, the supertrees T', T'', T''' propose an arbitrary resolution for the clade, A, B, C.

occurs when only one resolution appears for a group of taxa in one or several source trees, and the supertree contains a different resolution for the group. When different resolutions appear in source trees, as soon as the supertree proposes a resolution for the concerned taxa it contradicts at least one input tree. Contradictions are less direct when the supertree proposes a resolution that contradicts no single input tree but does contradict a combination of them.

A relationship contained in a supertree can present no contradiction with the input trees and still not be desirable. For instance, Figure 1 shows a collection of two source trees and four possible supertrees, T', T'', T''', and T, for this collection. Both B and C are sister taxa of A in the source trees, but no information is present in these trees to resolve the clade A, B, C. Thus, the fully resolved supertrees T', T'', T''' all take arbitrary decisions by proposing one of the possible resolutions for this clade. Here, T is the sole supertree not proposing an arbitrary resolution for the clade. Arbitrary resolutions are misleading as they display relationships that are not entailed by the input trees.

The above properties can be formalized in different ways depending on the kind of topological relationship considered; e.g., clades, nestings, triplets, etc. Following Goloboff and Pol (2002) and Grunewald et al. (2006), we chose to focus on triplets. Given a collection $\mathcal{T}$ of input trees and a candidate supertree T, $\mathcal{R}(T, \mathcal{T})$ denotes the set of triplets of $\mathcal{T}$ for which T proposes a resolution. More formally, $\mathcal{R}(T, \mathcal{T}) = \{AB|C \in rt(\mathcal{T})$ such that $\{AB|C, AC|B, BC|A\} \cap rt(T) \neq \emptyset\}$. The set $\mathcal{R}(T, \mathcal{T})$ corresponds to all topological information present in collection $\mathcal{T}$ that is related to the information present in supertree T. Using this notation, we can express the induction property (PI) and the non-contradiction property (PC) as follows:

- T satisfies PI for $\mathcal{T}$ if and only if for all $t \in rt(T)$, it holds that $\mathcal{R}(T, \mathcal{T}) \vdash t$. In other words, PI requires that each and every triplet of T is induced by $\mathcal{R}(T, \mathcal{T})$.
- T satisfies PC for $\mathcal{T}$ if and only if for all $t \in rt(T)$ and all $\bar{t}$, it holds that $\mathcal{R}(T, \mathcal{T}) \nvdash \bar{t}$. This means that, for each and every triplet of T, $\mathcal{R}(T, \mathcal{T})$ induces no alternative resolution.

For instance, considering collection $\mathcal{T} = \{T_1, T_2\}$ in Figure 2 and supertree T' of Figure 3, the set $\mathcal{R}(T', \mathcal{T})$ is $\{AC|E, AC|F, AB|E, AB|F, BC|E, BC|F, EF|A, EF|B, EF|C\}$. Note that the triplet $AD|C$ present in $rt(\mathcal{T})$ due to T_2 is not in this set because A, D, C are multifurcating in T'. When the source trees are incompatible, it is possible that $\mathcal{R}(T, \mathcal{T})$ contains two different triplets for the same three taxa. For example, consider the supertree T in Figure 3 proposed by the MC and MMC voting methods (Semple and Steel, 2000; Page, 2002) on the collection $\mathcal{T} = \{T_1, T_2\}$. $\mathcal{R}(T, \mathcal{T})$ contains both $AB|C$ (resulting from T_2) and $AC|B$ (resulting from T_1). In this case, the supertree T that contains the triplet $t = AB|C$ does not satisfy PC, since $\bar{t} = AC|B$ is in $\mathcal{R}(T, \mathcal{T})$ (hence, $\mathcal{R}(T, \mathcal{T}) \vdash \bar{t}$). Indeed, in this example, supertree T includes topological information contained in T_2 that contradicts that of T_1. This situation indirectly results from a difference in the sizes of the clades of T_1 and T_2, which are incompatible: the clade containing more taxa (here A, B, D) in T_2 versus (A, C) in T_1) is favored in the MC-MMC supertree. Such a size bias effect has been well-known in the field since Purvis (1995b) demonstrated it for the MRP voting method. Here it is illustrated for another voting method, and one might wonder whether this size bias is present in most voting methods. Note, however, that this size bias does not seem to have a major impact on MRP's accuracy (Baum and Ragan, 2004).

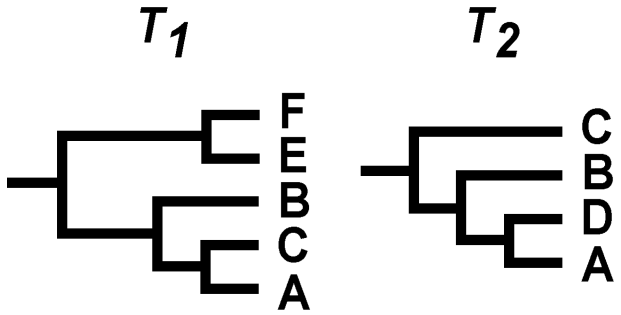

FIGURE 2. Example of a collection $\mathcal{T} = \{T_1, T_2\}$ of two source trees.

When the source trees are compatible, any reasonable method is expected to produce a supertree satisfying PC. However, some methods usually propose a supertree that does not satisfy PI. Indeed, compatible source trees can sometimes be displayed by an exponential number of supertrees, and some methods arbitrarily propose only one of them, thus selecting some triplets to the exclusion of other possible triplets. For instance, when considering the trivial case of two source trees $AB|E$ and $CD|E$, both MC and MMC propose the supertree ((A,B),(C,D),E), whereas numerous supertrees are possible; e.g., ((A,C),(B,D),E). In such a case, it seems preferable to output a consensus of all possible supertrees, as done by MRP (e.g., Bininda-Emonds and Bryant, 1998). Unfortunately, some topological information of the source trees (e.g., triplets) can be absent from the obtained consensus as it can contain highly multifurcating nodes.

However, for some compatible collections of trees, it is possible to find a supertree that displays all triplets of the collection and is also refined by all other possible supertrees. More formally, a set $\mathcal{R}$ of triplets is said to *identify* a tree T if and only if T strictly displays $\mathcal{R}$ and T is refined by every tree T' that strictly displays $\mathcal{R}$. A set $\mathcal{R}$ can identify at most one tree; thus, when the triplet set $\mathcal{R} = rt(\mathcal{T})$ of a collection $\mathcal{T}$ of source trees identifies a tree, this tree is a *canonical* representation of all possible supertrees.

Considering practical collections $\mathcal{T}$ of source trees, $rt(\mathcal{T})$ will almost never identify a tree, either because this set is incompatible or because it does not identify a particular tree. Nevertheless, it is possible that a subset of the triplets in $rt(\mathcal{T})$ identifies a tree T, and then the topological information contained in T exactly corresponds to a subset of the topological information contained in $\mathcal{T}$. Such a subset is most interesting when the triplets t it contains do not have an alternative resolution $\bar{t}$ in $rt(\mathcal{T})$. This situation occurs for the subset $\mathcal{R}(T, \mathcal{T})$ of $rt(\mathcal{T})$ when the supertree T satisfies PI and PC.

Proposition 1 *A tree T satisfies PI and PC for a collection $\mathcal{T}$ of trees if and only if $\mathcal{R}(T, \mathcal{T})$ identifies T.*

The proof is given in Appendix 1. It is based on the fact that a set $\mathcal{R}$ identifies a tree T if and only if $rt(T) = cl(\mathcal{R})$ (Grunewald et al., 2006: lemma. 2.1). This proposition confirms the relevance of PI and PC: having a supertree T that satisfies both of them highlights a part of $rt(\mathcal{T})$ (namely $\mathcal{R}(T, \mathcal{T})$) that exactly corresponds to a tree, i.e., does not contain arbitrary topological information, and moreover does not contradict any input tree. Such a feature is most desirable for supertrees inferred by veto methods.

Links with Other Advocated Properties

Properties similar to PI and PC were described in Goloboff and Pol (2002: 519) as "the property of [the supertree] displaying $AB|C$ if it is found in some input tree or implied by some combination of input trees and no input tree or combination of input trees displays or implies $AC|B$ or $BC|A$." These properties were also pointed out as being desirable by Grunewald et al. (2006). Using our formalism, they can be translated as follows for a supertree T representing a collection $\mathcal{T}$:

- PI': for any $t \in rt(T)$, it holds that $rt(\mathcal{T}) \vdash t$
- PC': for any $t \in rt(T)$ and for all $\bar{t}$, it holds that $rt(\mathcal{T}) \nvdash \bar{t}$.

The essential difference between PI'-PC' and PI-PC is whether we evaluate supertrees based on triplets in the original set of trees, $rt(\mathcal{T})$, or on the triplets commonly resolved by the supertree and at least one of the source trees, $\mathcal{R}(T, \mathcal{T})$. From the statement of the properties, it is clear that PC' implies PC and PI implies PI'. It is thus natural to wonder which version of the properties is preferable. Below, we show an example where PC' is too restrictive, and an example where PI' is too permissive. In contrast, PI and PC behave correctly in these examples.

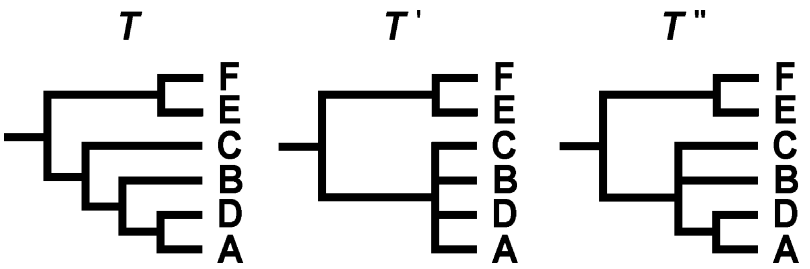

FIGURE 3. Various supertrees for the collection of Figure 2. T is the supertree proposed by the MC (Semple and Steel, 2000) and MMC methods (Page, 2002). T' and T'' are the supertrees respectively proposed by the $Build_{PC}$ and $PhySIC_{PC}$ algorithms described in this paper.

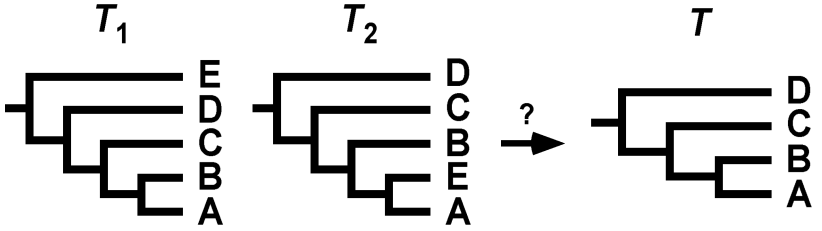

FIGURE 4. Excluding rogue taxa from the analysis can lead to informative supertrees.

Example 1 *Let $\mathcal{T} = \{T_1, T_2\}$ with T_1 and T_2 as shown in Figure 4. $rt(\mathcal{T})$ contains $AE|B$ and $AC|E$; therefore, $rt(\mathcal{T}) \vdash AC|B$. We also have $rt(\mathcal{T}) \vdash AB|C$ because $AB|C \in rt(T_1)$. Thus any tree providing a triplet on $\{A, B, C\}$ does not satisfy PC'. For analogous reasons PC' does not allow us to propose any triplet in the supertree. Thus, PC' rejects the tree T of Figure 4. Yet T is a reasonable and informative supertree for $\mathcal{T}$ and satisfies both PI and PC.*

We note that T is not a plenary supertree, i.e., it does not contain all input taxa, but this example shows that removing rogue taxa is a way in which more informative supertrees can be obtained. This is in line with the remark of Wilkinson et al. (2004), who stated that "non-plenary supertree methods might be most useful for identifying unstable leaves." For instance, such leaves might be involved in lateral transfers. This example easily generalizes to cases where the *supertree* actually contains more leaves than each source tree. Figure 5 depicts this generalization.

The next example shows a supertree satisfying both PI' and PC' while also displaying irrelevant triplets.

Example 2 *Let $\mathcal{T} = \{T_1, T_2\}$ with T_1 and T_2 as illustrated in Figure 6. $rt(\mathcal{T}) = \{AB|C, AB|X, BC|A\}$. The tree T of Figure 6 displays $\{AB|X, BC|X, AC|X\}$. $AB|X$ is present in (thus induced by) $rt(\mathcal{T})$ but the two other triplets can also be induced from $rt(\mathcal{T})$: $\{AB|X, BC|A\} \vdash \{BC|X, AC|X\}$. It follows that T satisfies PI'. Moreover, it is easily seen that no combination of triplets in $rt(\mathcal{T})$, other than $\{AB|X, BC|A\}$, induces triplets. However, T is clearly not an ideal supertree for $\mathcal{T}$ as no information in $\mathcal{T}$ induces group A, B, C to nest inside group A, B, C, X. The property PI, not satisfied by T, detects this problem: here $\mathcal{R}(T, \mathcal{T})$ only contains the triplet $AB|X$ and thus it does not induce the triplet $AC|X$ present in T.*

The PI' property quoted by Goloboff and Pol (2002) is stronger than the *Pareto* property (Neumann, 1983; Wilkinson et al., 2004) on triplets, which requires that the output tree contain all triplets and splits that occur in all source trees. The Pareto property is appealing in general and has also been advocated in the supertree context (property P6 of Steel et al., 2000). However, imposing the Pareto property on triplets may be problematic, even in the case of compatible source trees (Thorley and Wilkinson, 2003). This is due to the possibility of having *several* candidate supertrees that are both compatible with source trees and respect the Pareto property. In this case, no *single* supertree exists that satisfies the Pareto property while having no arbitrary resolution. The strict consensus of these supertrees does not necessarily satisfy the Pareto property. A solution is then to return several trees, either all candidate supertrees or their reduced consensus (Wilkinson, 1994). However, this solution may not well be suited when the aim is to summarize a collection of source trees into a single supertree that is more easily dealt with for further analysis by biologists.

When source trees are incompatible, it may even be impossible to have a supertree satisfying both the Pareto and non-contradiction properties (PC and PC') as shown in the following example.

Example 3 *Consider the collection $\mathcal{T} = \{T_1, T_2\}$ where $T_1 = (((A, D), B), ((C, F), E))$ and $T_2 = (((A, E), (B, F)), (C, D))$. Triplets $AB|C$ and $EF|D$ are displayed by both trees of $\mathcal{T}$. Thus, any supertree T for $\mathcal{T}$ must include all leaves in $\mathcal{T}$ in order to satisfy the Pareto*

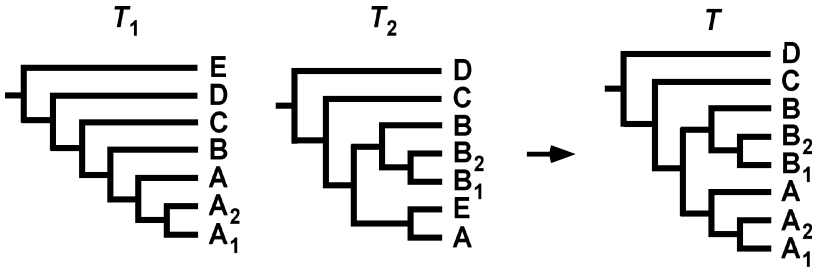

FIGURE 5. Another example with rogue taxa. This figure presents a generalization of the example displayed in Fig. 4 to the case of a supertree containing more taxa than each input tree.

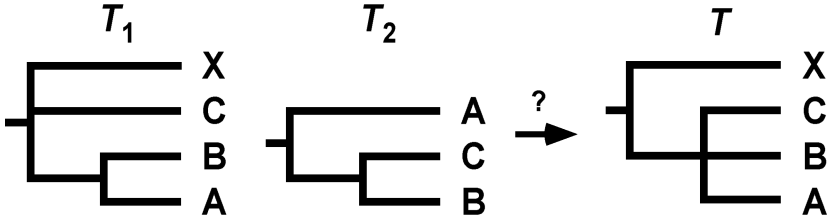

FIGURE 6. Contradiction in the source trees can lead to arbitrary resolution. An example where the presence of contradiction in the source trees (namely, $AB|C$ in T_1 versus $BC|A$ in T_2) can lead to the inferrence of arbitrary clades (namely, excluding X from the clade $\{A, B, C\}$ in the supertree T). This problem is detected by PI but not by PI′ nor PC′.

property. Since $rt(\mathcal{T})$ contains $AB|D$ and $AD|B$, any tree T displaying a triplet for the three leaves does not satisfy PC (hence PC′). For similar reasons, no supertree T can display a triplet on the taxa A, C, and D. Thus, any supertree satisfying PC (or PC′) and including all taxa of $\mathcal{T}$ contains a multifurcating node on taxa A, B, C, D, and hence does not display the triplet $AB|C$; i.e., does not satisfy the Pareto property.

In other words, imposing the Pareto property can lead the supertree to explicitly contradict relationships present in some input trees. This shows that the Pareto property on triplets is not compatible with the veto approach, where the proposed supertree must not contradict the source trees. However, the Pareto property can be considered for other topological relationships (Wilkinson et al., 2004). For example, there is always a supertree satisfying PI and PC as well as the Pareto property on partial or full splits contained in the source trees.

The Pareto property specifies relations that the supertree *must* contain. The complementary co-Pareto property specifies relations that the supertree *must not* contain. The co-Pareto property in the consensus context requires that the consensus tree contain no relationships that are not present in at least one input tree. However, Wilkinson et al. (2004) point out that this statement is not reasonable for supertrees, because "they might contain relashionships that are entailed by the input trees in combination, but are not present in any of them singly." Then they propose a weaker version that requires that the supertree does not contain relationships that are contradicted by all the input trees whose leaf set makes a contradiction possible. Note that, any supertree satisfying PC also satisfies the latter version of co-Pareto.

Steel et al. (2000) list five other properties that might be requested from supertree methods: changing the order of the trees in the input collection does not change the supertree (P1); renaming the taxa of the source trees gives the same supertree, but with the taxa renamed accordingly (P2); the output tree displays the source trees when they are compatible (P3); each leaf (taxon) that occurs in at least one source tree is in the supertree (P4); the running time of the method grows polynomially with respect to the total number of taxa (P5). The following example shows that ensuring P3 can force the supertree to contain arbirtrary clades. Thus, P3 can conflict with PI.

Example 4 *Let $\mathcal{T} = \{T_1, T_2\}$ with $T_1 = ((A, B), W)$ and $T_2 = ((A, B), (X, (Y, Z)))$. A supertree with taxon set $\{A, B, W, X, Y, Z\}$ that satisfies P3 must display T_2, hence must have a clade including Y, Z but not X. However, it will contain arbitrary clades, no matter where taxon W is attached. This is because any supertree satisfying PI must include a polytomy on W, X, Y, Z since source trees include no information on the relative position of W and the group X, Y, Z. For instance, the supertree $((A, B), (X, Y), W, Z)$ excludes the possibility for (A, B) and (X, Y) to be intermixed.*

Note that if polytomies of a supertree are interpreted in terms of an Adams consensus (Adams, 1972), then this example does not put P3 into question. However, this interpretation of polytomies does not prevail in phylogenetics, as we discuss in further detail in the case study paragraph.

Checking PC and PI in Polynomial Time

Existing supertree methods can sometimes output trees that do not satisfy PC or PI. For instance, the MC supertree obtained for the collection of Figure 2 does not satisfy PC, whereas on that of Example 4, it fails to satisfy PI. In contrast, for the collection $\{AB|C, BC|D\}$ the MC supertree satisfies both PI and PC. The MRP method sometimes outputs supertrees not satisying these properties (e.g., PC is not satisfied in Figure 1 of Bininda-Emonds and Bryant, 1998) and sometimes provides supertrees that satisfy them—e.g., when the source trees are compatible (Steel, 1992). We now describe an algorithm to decide whether a candidate supertree satisfies both PI and PC together. In case of a negative answer, it pinpoints those parts of the supertree contradicting these properties. This algorithm relies on two properties equivalent to PC and PI, whose formulation is less intuitive but whose checking is easy.

Definition 1 *Let $\mathcal{T}$ be the collection of source trees and T be a proposed supertree for $\mathcal{T}$. Define PC_{eq} and PI_{eq} to be the following properties*:

- *PC_{eq}: $rt(T) \cup \mathcal{R}(T, \mathcal{T})$ is compatible.*
- *PI_{eq}: for any $t \in rt(T)$ and for all $\bar{t}$, the set $\{\bar{t}\} \cup \mathcal{R}(T, \mathcal{T})$ is incompatible.*

Proposition 2 *(PI_{eq} and PC_{eq}) $\Leftrightarrow$ (PI and PC).*

Proof.

- **PC$_{eq}$ ⇒ PC**: PC$_{eq}$ ⇒ $\forall t \in rt(T)$, $\{t\} \cup \mathcal{R}(T, \mathcal{T})$ is compatible. This ensures that there is at least one tree T' that displays $\mathcal{R}(T, \mathcal{T}) \cup \{t\}$. It follows that T' displays $\mathcal{R}(T, \mathcal{T})$ but not t. As t is not displayed by every tree that displays the compatible set $\mathcal{R}(T, \mathcal{T})$, it follows that $\mathcal{R}(T, \mathcal{T}) \not\vdash t$.
- **PC ⇒ PC$_{eq}$**: $PC \Rightarrow \mathcal{R}(T, \mathcal{T}) \subseteq rt(T)$ (cf. proof of Proposition 1); this ensures that $rt(T) \cup \mathcal{R}(T, \mathcal{T})$ is compatible (since displayed by T).
- **PI + PC ⇒ PI$_{eq}$**: PI and PC ⇒ $cl(\mathcal{R}(T, \mathcal{T})) = rt(T)$ by Proposition 1. This ensures PI$_{eq}$.
- **PI$_{eq}$ + PC$_{eq}$ ⇒ PI**: PC$_{eq}$ ensures that $\mathcal{R}(T, \mathcal{T})$ is compatible. PI$_{eq}$ is exactly the definition of the induction for a compatible set, thus ensuring PI.

Note that we do not prove a direct equivalence between PI and PI$_{eq}$ in this general case. The two properties are only equivalent for a compatible set. In fact, PI$_{eq}$ is relatively uninformative without PC$_{eq}$, since PI$_{eq}$ holds as soon as $\mathcal{R}(T, \mathcal{T})$ is incompatible. Note also that another formulation of PC, closer to that of PI$_{eq}$ but less concise, is as follows: *for any $t \in rt(T)$, the set $\{t\} \cup \mathcal{R}(T, \mathcal{T})$ is compatible*.

PI$_{eq}$ and PC$_{eq}$ can easily be checked by using the *Build* algorithm, which indicates in polynomial time whether a set of rooted trees is compatible or not. A similar procedure was proposed by Steel (1992), and refined by Daniel (2004), to compute the strict consensus of all supertrees displaying a collection of compatible source trees. The following lemma provides us with an even faster way to check PC$_{eq}$.

Definition 2 (Direct contradiction) *A tree T* directly contradicts *a set of triplets $\mathcal{R}$ when there is a triplet t in $rt(T)$ such that $\exists t \in \mathcal{R}$. A supertree T is said to directly contradict a collection $\mathcal{T}$ of source trees if T directly contradicts $rt(\mathcal{T})$.*

Direct contradictions are linked with the PC property in the following way:

Lemma 1 *If a tree T does not directly contradict a collection $\mathcal{T}$ of source trees then the three following statements hold:*

1. *$\mathcal{R}(T, \mathcal{T}) \subseteq rt(T)$;*
2. *$\mathcal{R}(T, \mathcal{T})$ is compatible;*
3. *T satisfies PC$_{eq}$ for $\mathcal{T}$.*

Proof. By definition, $\mathcal{R}(T, \mathcal{T})$ only contains triplets on three-taxon sets for which there is a triplet in $rt(T)$. Because T does not directly contradict $\mathcal{T}$, the triplets of $\mathcal{R}(T, \mathcal{T})$ are resolved as those in T. It follows that $\mathcal{R}(T, \mathcal{T}) \subseteq rt(T)$ (proving 1). $\mathcal{R}(T, \mathcal{T})$ is therefore compatible (proving 2). Moreover, $\mathcal{R}(T, \mathcal{T}) \subseteq rt(T)$ ensures that $\mathcal{R}(T, \mathcal{T}) \cup rt(T)$ is compatible, which is exactly the formulation of PC$_{eq}$ (proving 3).

Thus, to check that a supertree T satisfies PC$_{eq}$, and hence PC, for a collection $\mathcal{T}$, it suffices to check that any triplet of $rt(T)$ is not resolved in a different way in a tree of $\mathcal{T}$. This can be done by computing the set $rt(T)$ of $O(n^3)$ triplets in T and then comparing $rt(T)$ with the set of triplets of each source tree T_i. If the collection $\mathcal{T}$ contains k source trees and a total of n taxa, then this simple implementation requires $O(kn^3)$ computing time. However, it is possible to check this condition in linear time for each pair T, T_i with $T_i \in \mathcal{T}$: first restrict in $O(n)$ time the trees T and T_i to the taxa they share; then apply the algorithm of Berry et al. (2005) that, given two trees with the same taxa, finds in $O(n)$ time a triplet resolved differently in the trees, or states that this situation does not arise. Thus, successively considering k source trees leads to a procedure that checks PC in $O(kn)$ computing time.

PhySIC: A POLYNOMIAL-TIME VETO SUPERTREE METHOD

We introduced above the PI and PC properties, showed their relevance, and described algorithms to check whether a given supertree T satisfies them. In this section, we show that it is possible to design a method that always produces supertrees that satisfy PI and PC. However, this aim is not precise enough, as the *star tree* (the tree whose leaves are all children of a single internal node) trivially satisfies these properties—simply because it does not resolve any triplet. Thus, a reasonable aim is to design a method that always infers supertrees that satisfy PI and PC *and* that contain as much resolution as possible; e.g., resolve as many triplets as possible. More precisely, we require a method that, given any collection $\mathcal{T}$, proposes a supertree T such that $\mathcal{R}(T, \mathcal{T})$ identifies T and $\mathcal{R}(T, \mathcal{T})$ has maximum size over all such subsets of $rt(\mathcal{T})$. Such a subset of $rt(\mathcal{T})$ is called a maximum identifying subset of triplets (MIST).

The difficulty of this problem cannot be simply deduced from previously known theoretical results for optimization problems on triplets. Indeed, the MIST problem is a middle term between the NP-hard problem that consists of finding a maximum-sized *compatible* subset of triplets (Bryant, 1997) and the polynomial-time problem that asks for the maximum-sized *tree-like* subset of a complete set of triplets (Berry and Gascuel, 2000; Bryant and Berry, 2001). Unfortunately, the MIST problem is NP-hard (Guillemot and Berry, 2007). This shows that it is highly unlikely that a polynomial-time algorithm exists that could find the most resolved supertree satisfying PI and PC. However, we can still rely on heuristic algorithms to find reasonable (but potentially suboptimal) solutions, as is commonly done for other NP-hard problems such as finding a most parsimonious tree or a maximum likelihood tree for a character matrix.

We present below a polynomial-time heuristic method that always outputs a supertree that satisfies PI and PC. The method tries to produce a supertree that contains as many input triplets as possible under this constraint. The method is a variant of the well-known *Build* algorithm

and is called *PhySIC—Phylogenetic Signal with Induction and non-Contradiction*. Supertrees inferred by the method have a degree of resolution that can be close to that of supertrees inferred by voting methods (see next section) while only containing clades that are not arbitrary with respect to the source trees nor contradicting them as detected by PC.

Inferring a Supertree that Satisfies PC

This section introduces algorithms, based on the *Build* algorithm, to produce nontrivial trees that satisfy PC.

The Build *Algorithm.*—The *Build* algorithm is a yes-or-no algorithm that tells whether a collection of triplets or larger trees is compatible or not. To achieve its goal, the algorithm tries to build a tree displaying the triplets; if the process is blocked at some step, this means that the input triplets are not compatible. This tree is built recursively, from the root to the leaves. First, the largest clades are identified, then clades included in the first ones, and so on. The composition of the clades is guided by the structure of a *graph*; that is, a set of objects (called *vertices*) with links (called *edges*) between pairs of them.

The graph used by *Build*, called here the *Aho Graph*, is defined as follows: let $\mathcal{R}$ be a collection of triplets on a taxon set X, the *Aho Graph* G for $\mathcal{R}$ is the undirected graph with vertices X and with edges (A, B) between two taxa A and B whenever there is a triplet $AB|C \in \mathcal{R}$. Thus, an edge between two taxa means that at least one triplet sees these two taxa in the same clade. The vertices of G are denoted by $v(G)$ (in the present description $v(G) = X$). For instance, Figure 7a shows the Aho graph built from $\mathcal{R} = rt(\{T_1, T_2\})$, where T_1, T_2 are the source trees of Figure 2 and, e.g., the edge between taxa B and D is due to the triplet $bd|c \in rt(T_2)$.

A *connected component* C_i of a graph is a maximal set of taxa linked to one another; i.e., such that for any pair A, B of taxa in C_i, there is a set of edges that links A to B. For instance, the graph in Figure 7a contains two connected components: $C_1 = \{E, F\}$ and $C_2 = \{A, B, C, D\}$. The connected components of graph G are denoted by $CC(G)$. The vertices of a component C_i of G are denoted by $v(C_i)$. When the Aho graph contains several connected components, each of them corresponds to a clade of the tree representing the input collection of triplets (if such a tree exists). Once these clades are known, the clades contained in each of these primary clades are found by recursively processing Aho graphs for subsets of triplets that respectively concern the taxa of these clades: the restriction of $\mathcal{R}$ to taxa of a component C_i is denoted by $\mathcal{R}|v(C_i)$ and defined as $\{AB|C \in \mathcal{R} \text{ such that } \{A, B, C\} \subseteq v(C_i)\}$. For example, Figure 7b shows the Aho graph obtained from $\mathcal{R}|v(C_2)$ where $\mathcal{R}$ is the set of triplets due to source trees in Figure 2, and C_2 is the component of the initial Aho graph shown in Figure 7a. The recursive calls stop when dealing with components containing less than 3 taxa, because there is no triplet (hence incompatibility) on so few taxa. However, if at some point in the recursive process, the Aho graph for a set of at least three taxa has only one connected component, this means that the input trees are conflicting on the resolution of these taxa. When this happens, the algorithm states that the collection of source trees is incompatible. Otherwise, when all recursive calls return, the algorithm concludes that the source trees are compatible. For instance, when run on the collection of Figure 2, *Build* first finds two connected components, $C_1 = \{E, F\}$ and $C_2 = \{A, B, C, D\}$, but the recursive call on C_2 leads to a graph containing only one connected component (Figure 7b), which leads the algorithm to detect the incompatibility of the source trees.

A First Simple Modification of Build.—We first describe here a simple modification of *Build* that infers a supertree from a collection of source trees $\mathcal{T}$. This subroutine, called $Build_{PC}$ (Figure 8), takes as input the triplet set $\mathcal{R} = rt(\mathcal{T})$ of a collection $\mathcal{T}$ of source trees and the list S of taxa contained in these trees. $Build_{PC}$ mainly differs from *Build* when the Aho graph contains one connected component on the set S of taxa currently considered (line 1). In this case, $Build_{PC}$ returns the star tree on S (i.e., a single polytomy on S, thus contradicting no input triplet), whereas *Build* simply concludes that the sources trees are incompatible. This star tree is then grafted as a subtree of the tree built by the previous recursive call. Thus, we can now output a supertree even when the source trees are incompatible. As an example, from the collection of Figure 2, $Build_{PC}$ infers the supertree T' displayed in Figure 3.

Proposition 3 *Given a collection $\mathcal{R}$ of triplets on a taxon set S, $Build_{PC}$ returns a tree T on S that satisfies the PC property for $\mathcal{R}$.*

The proof of this proposition can be found in Appendix 1.

A More Involved Algorithm to Infer a Supertree Satisfying PC.—$Build_{PC}$ sometimes produces poorly resolved trees due to multifurcations returned in cases where G contains a single connected component (i.e., when $\mathcal{R}$ contains conflicts covering the considered subset of taxa). In the most extreme (though unlikely) case, this situation occurs at the first step of the algorithm, which then outputs a star tree.

The most basic conflicts between triplets of $\mathcal{R}$ occur when two different triplets t and $\bar{t}$ appear in $\mathcal{R}$ for a same set of three taxa. Such a direct contradiction cannot be present in a tree that satisfies PC. Given $\mathcal{R}_{dc}$, the set of triplets such that $t, \bar{t} \in \mathcal{R}$ it seems relevant to consider the subset $\mathcal{R}' = \mathcal{R} - \mathcal{R}_{dc}$. We define a variant of $Build_{PC}$, called $PhySIC_{PC}$, that resorts to that subset whenever conflicts are detected. This enables the produced supertree T' to be generally much more resolved than the tree returned by $Build_{PC}$. For instance, Figure 7b shows the graph obtained for $\mathcal{R}|v(C_2)$, where $\mathcal{R}$ are triplets of the collection in Figure 2 and C_2 is the connected component shown in Figure 7a. This graph is connected due to the direct conflicts between $AB|C$ (resulting from T_1) and $BC|A$ (resulting from T_2). This situation leads $Build_{PC}$ to return a polytomy on A, B, C, D. In contrast, building the graph on the basis of $\mathcal{R}'$ results in two connected

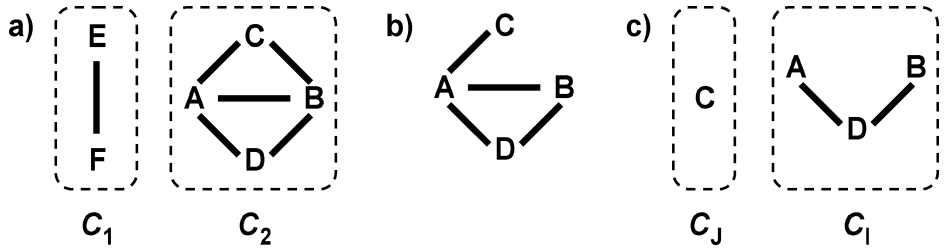

FIGURE 7. Examples of graphs. (a) The initial Aho graph G created from the triplets $rt(\mathcal{T})$ of the collection $\mathcal{T}$ displayed in Figure 2. The two connected components of G are $C_1 = \{E, F\}$ and $C_2 = \{A, B, C, D\}$. (b) The Aho graph obtained from $\mathcal{R}|v(C_2)$. This graph is connected, showing that the input trees conflict on the resolution of $\{A, B, C, D\}$ and hence are incompatible. (c) The Aho graph obtained from $\mathcal{R}|v(C_2)$ when removing the triplets $\mathcal{R}_{dc} = \{AB|C, AC|B\}$.

components, C_i and C_j, allowing $PhySIC_{PC}$ to propose a tree with two subtrees for taxa A, B, C, D.

The correctness of $Build_{PC}$ ensures that T' satisfies PC with respect to $\mathcal{R}'$ but without any guarantee that this also holds with respect to $\mathcal{R}$. To ensure the latter, and thus the correctness of $PhySIC_{PC}$, T' must not resolve any triplet of $\mathcal{R}_{dc}$. A way to ensure this is to collapse any branch of T' that resolves a triplet of $\mathcal{R}_{dc}$. The tree thus obtained is still always at least as resolved as the one proposed by $Build_{PC}$ and potentially contains supplementary branches. Indeed, direct contradictions at the root of a clade no longer prevent the proposition of clades on subsets of its taxa. For instance, on the collection of Figure 2, the tree initially computed by $PhySIC_{PC}$ is the tree called T in Figure 3. But as the branch leading to the clade (A, D, B) contradicts $AC|B \in \mathcal{R}_{dc}$, the branch above this clade is collapsed, and the final tree output by $PhySIC_{PC}$ is then the tree named T'' in Figure 3. This tree contains one clade more than the tree output by $Build_{PC}$ (the tree named T' in the figure).

These ideas are included in the $PhySIC_{PC}$ algorithm whose pseudo-code is detailed in Figure 9.

Theorem 1 *Given a triplet set $\mathcal{R} = rt(\mathcal{T})$ from a collection $\mathcal{T}$ of source trees on a taxon set S of n leaves, $PhySIC_{PC}$ returns in $O(n^4)$ time a tree T satisfying PC for $\mathcal{T}$.*

The proof of this result can be found in Appendix 1.

Ensuring that the Supertree Satisfies PI

The supertree T_{PC} output by $PhySIC_{PC}$ does not usually satisfy the PI property. The $PhySIC_{PI}$ algorithm transforms T_{PC} so that it also satisfies PI. To that aim $PhySIC_{PI}$ recursively searches the tree and checks that for each branch each triplet is induced by $\mathcal{R}(T_{PC}, \mathcal{T})$. Theorem 3.1.1 of Daniel (2004) provides a useful characterization to decide when a branch is justified, directly or indirectly, thanks to triplets present in $\mathcal{R}(T_{PC}, \mathcal{T})$. When considering the branch linking a node u to a child subtree S_i, the theorem considers a graph G_{ij} for any sibling subtree S_j of S_i. Any such graph G_{ij} is the Aho graph with vertices $L(S_i)$, and with edges due to triplets of $\mathcal{R}(T_{PC}, \mathcal{T})$ whose three leaves are in $L(S_i) \cup L(S_j)$. The theorem states that the branch from u to the root of S_i is justified if and only if G_{ij} is connected, for any sibling subtree S_j.

Example 5 *Consider for instance the simple example where $\mathcal{T}$ contains the trees ((A,B),X) and ((E,F),X). The Aho graph for $rt(\mathcal{T}) = \{AB|X, EF|X\}$ is made of three connected components: $C_1 = \{A, B\}$, $C_2 = \{E, F\}$, and $C_3 = \{X\}$; therefore, applying the $PhySIC_{PC}$ algorithm gives the tree $T_{PC} = ((A, B), (E, F), X)$. T_{PC} displays $AB|E$ even though this information is not induced by $\mathcal{T}$. Indeed, the branch defining*

Algorithm $Build_{PC}(S, \mathcal{R})$

if *S contains less than 3 taxa* **then** return the trivial tree on S
Let G denote the Aho graph for $\mathcal{R}$
if *G has only one connected component* **then**
1 return the star tree on $L(\mathcal{R})$
else
$\mathcal{C}_{PC} \leftarrow CC(G)$
foreach $C_i \in \mathcal{C}_{PC}$ **do**
2 **if** $(\mathcal{R}|v(C_i)) = \emptyset$ **then** $T_i \leftarrow$ star tree on $v(C_i)$
else $T_i \leftarrow Build_{PC}(v(C_i), \mathcal{R}|v(C_i))$
3 Return the tree made of a root node connected to $T_1, T_2, ..., T_{|\mathcal{C}_{PC}|}$

FIGURE 8. Details of the $Build_{PC}$ subroutine taking a set S of taxa and a set $\mathcal{R}$ of triplets on S as input.

```
Algorithm PhySIC_PC(S, R)
    if S contains less than 3 taxa then return the trivial tree on S
    Let G denote the Aho graph for R
    if G has several connected components then C_PC ← CC(G)
    else
        Let R_dc be the set of triplets t s.t. t, t̄ ∈ R
        R' ← R − R_dc
        Let G' be the Aho graph for R'
4       if G' is connected then C_PC ← v(G)
        else
            C_PC ← CC(G')
5           repeat
6               foreach AB|C ∈ R_dc do
7                   if A, B ∈ C_i and C ∈ C_j (with C_i, C_j ∈ C_PC and i ≠ j) then
8                       Build G'_i the Aho graph for R'|v(C_i)
                        if G'_i is connected then C_PC ← (C_PC − {C_i}) ∪ v(C_i)
9                       else C_PC ← (C_PC − {C_i}) ∪ CC(G'_i)
            until C_PC no longer changes
    foreach C_i ∈ C_PC do
        if (R|v(C_i)) = ∅ then T_i ← star tree on v(C_i)
        else T_i ← PhySIC_PC(v(C_i), R|v(C_i))
10  Return the tree made of a root node connected to T_1, T_2, ..., T_|C_PC|
```

FIGURE 9. Details of the $PhySIC_{PC}$ subroutine taking a set S of taxa and a set $\mathcal{R}$ of triplets on S as input.

the clade (A, B) is detected as not justified since the corresponding connected component, C_1, is not connected in the Aho graph when we consider only edges due to triplets with taxa in $C_1 \cup C_2$.

This theorem is the basis of a *decision* algorithm called *Identifies* that states whether a given set of triplets identifies a given tree (Daniel, 2004). It is possible to design a simple variant of this algorithm that always returns a tree (not just a *yes* or *no* answer): when a branch between a node p and the root of a subtree S_i is not justified, the idea is to replace S_i by a star tree on the taxa of the corresponding clade. This crude variant removes the unjustified branches, but also potentially many other branches; i.e., those inside S_i, those leading to sibling subtrees S_j of S_i, and those inside S_j subtrees. $PhySIC_{PI}$ is a more refined variant that only collapses the unjustified branches. See the pseudo-code in Figure 10 for details. In this code, $PhySIC_{PI}$ is given a tree T in which unjustified branches are to be collapsed and a collection $\mathcal{T}$ of source trees or, equivalently, the corresponding set of triplets (as written in the pseudo-code). $PhySIC_{PI}$ repeatedly calls the $Check_{PI}$ subroutine to detect unjustified branches that are then removed until none remains (note that in the pseudo-code of $Check_{PI}$, $S(T)$ denotes (complete) subtrees connected to the root of T; i.e., the subtrees corresponding to the largest clades under the root of T).

From the collection of Figure 2, $PhySIC_{PC}$ infers the supertree T'' displayed in Figure 3 and none of the three internal branches of T'' are collapsed by $Check_{PI}$. For instance, consider the step where $Check_{PI}$ checks the subtree ((A,D),B,C) of T'', whose child subtrees are (A,D) plus the two trivial subtrees on B and C. The sole branch that has to be checked in ((A,D),B,C) is the one defining the clade (A,D). Here, $Check_{PI}$ builds two Aho graphs with vertices $\{A, D\}$: one with edges due to triplets on $\{A, D\} \cup \{B\}$ and one with edges due to triplets on $\{A, D\} \cup \{C\}$. Both graphs are connected thanks to triplets of the source tree T_2; therefore, $Check_{PI}$ does not collapse any branch at this step.

Theorem 2 *Given a collection $\mathcal{T}$ of trees and a tree satisfying PC for $\mathcal{T}$, $PhySIC_{PI}$ returns in $O(n^4)$ time a tree T on $L(\mathcal{T})$ that satisfies both PC and PI for $\mathcal{T}$.*

The proof of this Theorem can be found in the appendix.

Algorithm $PhySIC_{PI}(T, \mathcal{T})$

$T_{PI} \leftarrow T$
repeat
  $\mathcal{R}_{PI} \leftarrow \mathcal{R}(T_{PI}, \mathcal{T})$
  $T_{PI} \leftarrow Check_{PI}\ (T_{PI}, \mathcal{R}_{PI})$
until T_{PI} *no longer changes*
Return T_{PI}

Algorithm *PhySIC* $(\mathcal{T})$

Let S be the taxa appearing in $\mathcal{T}$
$\mathcal{R} \leftarrow rt(\mathcal{T})$
$T_{PC} \leftarrow PhySIC_{PC}(S, \mathcal{R})$
Return $PhySIC_{PI}(T_{PC}, \mathcal{R})$

Algorithm $Check_{PI}(T, \mathcal{R})$

if T *is made of a single leaf* **then** return T
Let G be the Aho graph for $\mathcal{R}$
if $|CC(G)| = 1$ **then** return "error, $\mathcal{R}$ is incompatible"
repeat
  foreach $T_i \in S(T)$ **do**
    Let G_i be the Aho graph for $\mathcal{R}|L(T_i)$
    foreach $T_j \in S(T)$ *s.t.* $T_i \neq T_j$ **do**
      Build G_{ij} from G_i and $\mathcal{R}|(L(T_i) \cup L(T_j))$
      if G_{ij} *is not connected* **then**
        Collapse the branch between the root of T and T_i
until *no branch of* T *is collapsed*
foreach $T_i \in S(T)$ **do**
  $T'_i \leftarrow Check_{PI}\ (T_i, \mathcal{R}|L(T_i))$
Return the tree made of a root node connected to $T'_1, T'_2, ..., T'_{|S(T)|}$

FIGURE 10. Details of the *PhySIC* algorithm and $PhySIC_{PI}$ and $Check_{PI}$ subroutines.

The *PhySIC* algorithm (see pseudo-code) builds a supertree for a collection of k source trees $\mathcal{T}$ by first computing the set $rt(\mathcal{T})$ and then successively calling $PhySIC_{PC}$ and $PhySIC_{PI}$. Since $rt(\mathcal{T})$ is computed in $O(kn^3)$, *PhySIC* runs in $O(kn^3 + n^4)$ time.

Theorem 3 *Given a collection $\mathcal{T}$ of k source trees on n leaves, PhySIC returns in $O(kn^3 + n^4)$ time a tree satisfying both PC and PI.*

Lastly, we note that similar procedures can be designed to modify the supertree proposed by any existing supertree method. If a method proposes a supertree T that does not satisfy PC and PI, it is possible in polynomial time to transform T into a tree T' that satisfies these properties. Indeed, the algorithm indicated previously to check PC indicates the triplets from which the incompatibility arises. Then the branches of T inducing these triplets can be collapsed to obtain a tree T' satisfying PC. Now, $PhySIC_{PI}$ can be applied to T' to ensure that it also satisfies PI (without invalidating PC).

BIOLOGICAL CASE STUDIES ON PRIMATES

To illustrate the impact of the PC and PI properties on supertree inference, and to compare the behavior of veto methods like *PhySIC* to that of voting methods like MRP and MMC, we present two case studies centered on Primates. This mammalian order is one of the first taxonomic groups for which a large-scale supertree approach has been conducted (Purvis, 1995a). The first example is designed to show the desirable properties of *PhySIC* compared to other supertree methods on a smaller, understandable taxonomic scale. The second example addressing the question of the primate supertree at the genus level shows how *PhySIC* performs on a larger taxonomic scale—approaching what supertree studies tend to be performed on—and shows that varying degrees of resolution are achieved in the supertree depending upon the nodes retained from the input trees.

First Example: Illustration of Supertree Desirable Properties

Source Trees.—We focused on a subsample of Primates IRBP (interphotoreceptor retinoid binding protein) and ADRA2B (α2B-adrenergic receptor) gene sequences, respectively, from Poux and Douzery (2004) and Poux et al. (2006), with a rodent (*Mus*) and lagomorph (*Oryctolagus*) outgroup. For ADRA2B, the hominoid representative was *Pan*, with the sequence downloaded from the chimp ENSEMBL project. The ADRA2B and IRBP source trees were inferred by maximum likelihood (ML) analysis of the corresponding alignments, using PHYML (Guindon and Gascuel, 2003), version 2.4.4, under a GTR+Γ_4+INV model of DNA evolution. The node support was estimated after 1000 bootstrap replicates using the same software and expressed as bootstrap percentages (BP). Denser taxonomic and phylogenetic information for Strepsirrhines (i.e., *Lemurs* and *Galagos*) was sought from

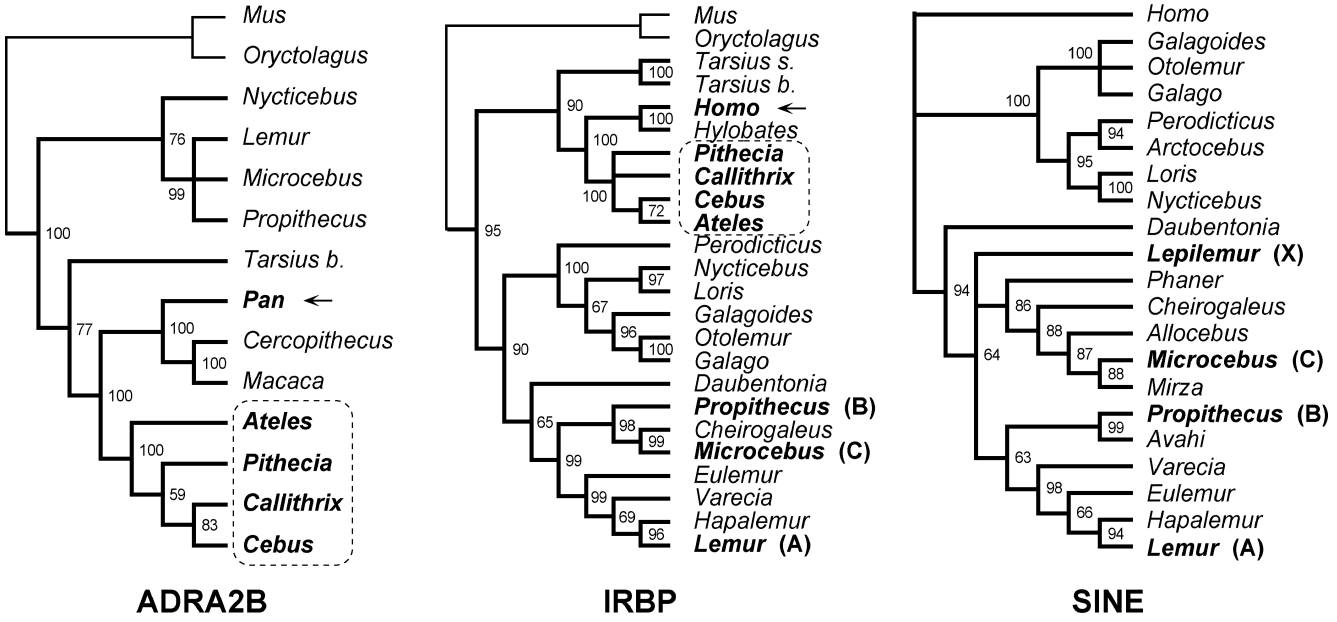

FIGURE 11. Three source trees for the case study on primates. Majority rule consensus source trees derived from the bootstrap analysis of ADRA2B and IRBP sequences (in maximum likelihood) and SINE characters (in maximum parsimony). Bootstrap percentages are indicated on nodes, and only nodes defined by more than 50% are considered. Thin branches lead to the outgroup. Taxa in bold are handled differently by the different supertree algorithms and illustrate three different situations (arrows, letters, box: see main text). The two *Tarsius* species are *T. bancanus* (*b*) and *T. syrichta* (*s*).

a study of presence-absence of short interspersed nuclear elements (SINE) integrations in primate genomes (Roos et al., 2004: fig. 2). Sixty-one monolocus SINE characters detected by these authors were subjected to a maximum parsimony analysis using PAUP* (Swofford, 2002), version 4b10, with 1000 bootstrap replicates using a heuristic search, with 10 random additions of taxa, and TBR branch swapping. We only retained the best supported nodes of source trees; i.e., those showing at least 50% bootstrap (cf. also Daubin et al., 2002).

Comparison of Supertrees Inferred from PhySIC, *MMC, and MRP.*—Starting from the three source trees (Figure 11), supertrees were built using the MMC, MRP, and *PhySIC* methods. For MRP, the matrix representation of the three source topologies resulted in 47 characters. Parsimony analysis was conducted under PAUP*, with a heuristic search with 1000 random addition sequences, and TBR branch swapping, resulting in 864 equally parsimonious trees, a strict consensus of which provided the MRP supertree. The MMC supertree was obtained using the program distributed by Rod Page. Figure 12 shows the supertrees respectively reconstructed by MMC, MRP, and *PhySIC* with its $PhySIC_{PC}$ intermediate step.

The supertrees produced all contain some soft polytomies, each of them representing uncertainty about the resolution of a node's child subtrees or lineages. A soft polytomy can have two distinct interpretations, differing in the set of admissible fully resolved phylogenies it encompasses. Consider the case of the polytomous node P in the MMC tree of Figure 12. This node has three child subtrees $S_1 = (Homo, Hylobates)$, $S_2 = (Pan,(Cercopithecus, Macaca))$, and S_3, the Platyrrhinii clade. The most widespread meaning of a soft polytomy accepts any fully resolved tree on subtrees S_1, S_2, S_3 that keeps their monophyly: $((S_1, S_2), S_3)$, $((S_1, S_3), S_2)$, or $((S_2, S_3), S_1)$. Strict consensus, majority-rule consensus, and hence MRP, interpret polytomies in this way (Margush and McMorris, 1981). Polytomies proposed by $PhySIC_{PC}$ are also to be interpreted in this way. A second interpretation of soft polytomies was introduced by the Adams consensus (Adams, 1972) and is also intended by MC (Semple and Steel, 2000) and MMC (Page, 2002). This interpretation accepts as possible phylogeny any fully resolved tree that maintains the structure of each subtree respectively, no matter whether or not S_1, S_2, and/or S_3 are kept monophyletic (i.e., their leaves can be interleaved). Thus, the polytomy P of the MMC tree in Figure 12 can indeed give rise to fully resolved trees grouping *Pan* and *Homo* without *Hylobates*, as long as *Pan* is kept outside the clades containing *Cercopithecus* and *Macaca* (which is the structure imposed by S_2). Under this interpretation, a soft polytomy represents a much wider range of fully resolved phylogenies than with the first interpretation, and is harder to interprete in a phylogenetic context. (In particular, this means that simulation studies on supertree methods that use the Robinson and Foulds distance to evaluate the performance of MC or MMC are misleading: on the previous example, the MMC method would have been considered to propose the incorrect clades *Homo* + *Hylobates*, and *Pan* + *Hylobates*.)

The contribution of $PhySIC_{PC}$ to the supertree inference may be illustrated by the situation among platyrrhines. Here, ADRA2B indicates that *Ateles* is the

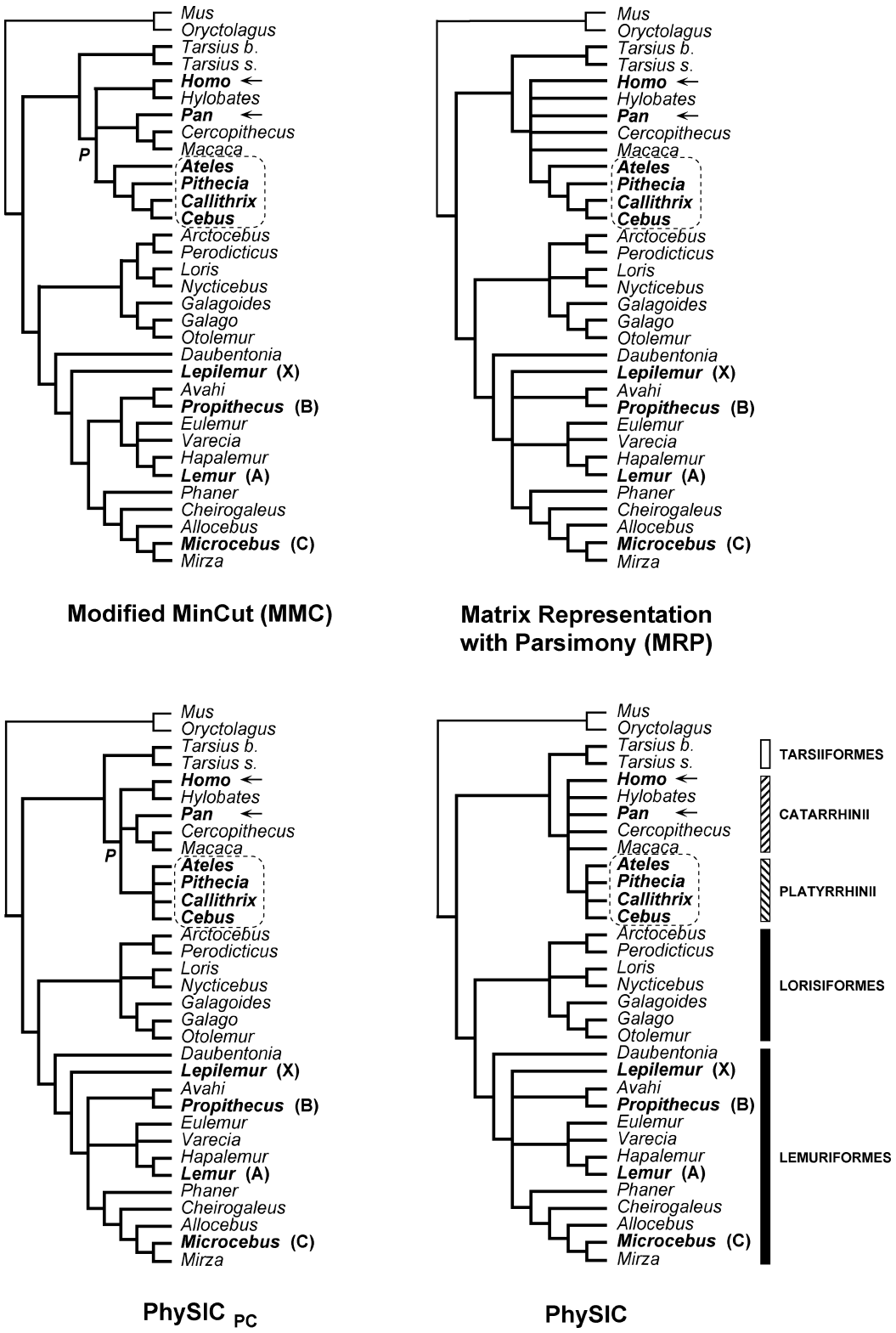

FIGURE 12. Comparison of supertrees inferred by three methods: MMC, MRP, and *PhySIC* (the $PhySIC_{PC}$ intermediate step of the latter is also displayed). Thin branches lead to the outgroup. Taxa in bold are handled differently by the different supertree algorithms and illustrate three different situations. (i) Arrows indicate the surprising positioning of *Homo* and *Pan* under the MMC and $PhySIC_{PC}$ algorithms; (ii) A-B-C-X letters correspond to taxa arbitrarily grouped by MMC and $PhySIC_{PC}$ (*cf.* Figure 2); (iii) boxes contain platyrrhine taxa for which MMC and MRP contradict the IRBP source topology. The *P* label on MMC and $PhySIC_{PC}$ supertrees refers to the polytomy involving catarrhine and platyrrhine clades. The taxonomic frame for Primates is given on the *PhySIC* supertree. Hatched rectangles represent Anthropoidea (Catarrhini + Platyrrhini). White and black rectangles respectively represent Haplorrhini (Tarsiiformes + Anthropoidea) and Strepsirrhini (Lorisiformes + Lemuriformes).

sister-group of *Pithecia*, *Callithrix*, and *Cebus*, whereas IRBP indicates that *Ateles* and *Cebus* are the closest relatives (Figure 11: boxed areas). This conflict is detected by $PhySIC_{PC}$. As a result, the $PhySIC_{PC}$ and *PhySIC* supertrees display all four platyrrhines within a multifurcation. By contrast, MRP and MMC give priority to the *Callithrix* + *Cebus* grouping present in the ADRA2B source tree, and thus contradicts the *Ateles* + *Cebus* grouping present in the IRBP source tree. Resolution of this conflict between the source trees reflects the voting approach followed by MRP and MMC. For instance, consider the case of MRP: the ADRA2B source topology comprises two nodes within platyrrhines, against one node for the IRBP topology. Therefore, MRP favors the node *Callithrix* + *Cebus* involved in a topological conflict but belonging to a larger and more resolved clade (Bininda-Emonds and Bryant, 1998). The behavior of MRP and MMC on platyrrhines is problematic. Indeed, it favors one source topology while contradicting another, just on the basis of their respective levels of resolution, and despite the fact that both contain the same number of taxa for the platyrrhine subtree. MRP has already been criticized on this point (e.g., Goloboff, 2005). Note that source trees also conflict on the position of *Propithecus* with respect to *Microcebus* and *Lemur*. However, in this case, MRP behaves as $PhySIC_{PC}$ and *PhySIC*; i.e., displays a polytomy on groups containing these three taxa (Figure 12: letters *A*-*B*-*C*). By contrast, MMC groups together the *Propithecus* and *Lemur* clades, following the SINE information, but contradicting the IRBP information.

The complementary contribution of $PhySIC_{PI}$ to the supertree inference may be illustrated by the situation among Catarrhines + Platyrrhines. Although not contradicting the source trees, the $PhySIC_{PC}$ supertree contains two topological errors. First, man and chimp do not group together relative to the gibbon, as would be expected from a plethora of data (Goodman et al., 2005). *Homo* is instead associated with *Hylobates*, whereas *Pan* branches with the two cercopithecoids, *Cercopithecus* and *Macaca*. This situation results from the taxon sampling of the source topologies. More precisely, man and chimp are not simultaneously present in any source tree; i.e., the former clusters with the gibbon (IRBP) and the latter with cercopithecoids (ADRA2B). These two source clades are reproduced in $PhySIC_{PC}$ and MMC supertrees.

In the case of $PhySIC_{PC}$, these two clades are involved in a polytomy with the platyrrhines. This polytomy means that (*Homo*, *Hylobates*) is a sister clade of the clade containing *Pan*. However, although these clades are correct when considered separately, they should not be sister groups in the supertree. $PhySIC_{PI}$ detects this situation of arbitrary resolution and collapses the corresponding branches, thus the final *PhySIC* supertree allows for a group (*Pan*, *Homo*). MMC displays the same polytomy as $PhySIC_{PC}$ but with a different meaning: the interleaving interpretation of this soft polytomy means that MMC does not reject the expected resolution, namely grouping (*Pan*, *Homo*) as a sister clade of *Hylobates*. In conclusion, both MMC and *PhySIC* allows for the expected group (*Pan*, *Homo*), but note that the *PhySIC* supertree is more accurate, as its polytomy does not allow the catarrhine taxa *Homo*, *Hylobates*, or *Pan* to branch within the platyrrhines. Here, MRP does not introduce arbitrary resolutions and proposes a polytomy involving the five catarrhine taxa.

Another problem of MMC and $PhySIC_{PC}$ supertrees is that *Lepilemur* is the sister-group of all Lemuriformes but *Daubentonia*, whereas this topological information is not present in the only source tree (SINEs) for which *Lepilemur* is scored. This result is explained by the fact that the restriction of IRBP and ADRA2B source topologies to taxa lettered *A*-*B*-*C*-*X* leads to the situation described on Figure 6. Thanks to the PI property, the *PhySIC* algorithm again corrects this problem and displays a polytomy involving the major clades of lemuriformes, together with *Lepilemur* (Figure 12). The same polytomy is also proposed by MRP. Overall, this first case study illustrates that the two properties introduced in the present work help to identify and manage the potential arbitrary and conflicting resolutions arising in supertrees when combining independent source topologies.

Second Example: A PhySIC Supertree of Primate Genera

Primary Data and Source Tree Inference.—We used 24 data sets to reconstruct the primate phylogeny: two mitochondrial DNA (mtDNA), 19 nuclear DNA, and three transposable elements data sets. All sequences used in this study were retrieved from EMBL-Genbank databases. The sampling of genes and other molecular markers is detailed in Table 1. The corresponding data are available under TreeBASE accession numbers S1879 and M3455. This combined data set encompasses 95% of all primate extant genera (Wilson and Reeder, 2005); i.e., 66 genera. Two subfossil genera from ancient DNA analyses were also included (Karanth et al., 2005). All genes were aligned with Clustal X (Thompson et al., 1997) with subsequent manual refinement. We used *Mus* and *Rattus* as outgroups in all analyses for which sequence data was available. Each gene was analyzed with the ML criterion under the best fitting model (Table 1). Separate ML phylogenetic reconstructions and bootstrap analyses were performed with PHYML (Guindon and Gascuel, 2003) as described in previous section. Maximum parsimony phylogenetic reconstruction and bootstrap analysis on the three transposable element data sets were also conducted using PAUP* as described in previous section. Clades of the source trees with BP values above a specified threshold were retained. To evaluate the influence of this parameter, five *PhySIC* supertrees were inferred by respectively considering BP $\geq$ 50%, 60%, 70%, 80%, and 90%. Each run of *PhySIC* took less than 4 s on an Intel MacBook.

The Major Clades of Primate Genera.—The most resolved supertree reconstructed by *PhySIC* is obtained when source trees were restricted to nodes supported by more than 70% bootstrap (Figure 13; BP$\geq$ 70%). This topology conforms to current ideas on primate phylogeny and is close to the informal supertree of Primates at the genus level proposed by Goodman et al. (2005). In

TABLE 1. Molecular markers used to infer the primate source trees. The best-fitting model and the number of primate genera per marker are indicated. (mtDNA: mitochondrial DNA)

Markers	Model	Genera
α-2B Adrenergic receptor	HKY + Γ	16
Albumin gene introns 3 and 4	K80 + I	20
ATPase 7A	HKY + Γ	11
Breast and ovarian cancer susceptibility 1	HKY + Γ	12
Cytochrome *b* [mtDNA]	GTR + Γ+I	68
α-1,2 fucosyltransferase	HKY + Γ	21
β globin	GTR + I + Γ	22
γ globin exons 1 and 2	HKY + Γ	26
ϵ globin	HKY+Γ	33
Glucose-6-phosphate dehydrogenase introns 4 and 5	HKY + Γ	18
Glucose-6-phosphate dehydrogenase introns 7 and 8	HKY + Γ	12
Interphotoreceptor retinoid-binding protein exon 1	HKY + Γ	34
Interphotoreceptor retinoid-binding protein intron 1	K80 + Γ	24
Lysozyme gene	HKY + Γ	19
Membrane cofactor protein gene	HKY + Γ	13
β_2-microglobulin precursor exons 1 and 2	HKY + Γ	18
NADH dehydrogenase subunit 5 [mtDNA]	GTR + Γ+I	24
Phospholipase C β4 gene	GTR	13
Testis-specific protein gene	HKY + Γ	21
von Willebrand gene introns 11 and 12	HKY + Γ	37
9.3 kb chromosome Xq13.3 fragment	HKY + Γ	13
SINE (Roos et al., 2004)	—	20
ALU (Singer et al., 2003)	—	8
SINE (Xing et al., 2005)	—	15

addition, we here extend their taxon sampling with the three extant genera *Euoticus*, *Piliocolobus*, and *Simias*, and the two subfossil genera *Megaladapis* and *Paleopropithecus*. Our supertree displays the fundamental dichotomy among Primates between Strepsirrhini and Haplorrhini. Strepsirrhines then split into Lorisiformes (*Lorises* and *Galagos*) and Lemuriformes (lemurs and *Daubentonia*, the aye-aye). Haplorrhines also split into Tarsiers and Anthropoids. The latter clade subsequently divides into monophyletic New World primates (Platyrrhini) and Old World primates (Catarrhini). Platyrrhini display a trifurcation involving the three families Atelidae (the *Ateles* + *Alouatta* clade), Pitheciidae (the *Pithecia* + *Callicebus* clade), and Cebidae (the *Cebus* + *Saimiri* + *Aotus* + *Saguinus* clade). Catarrhines split into Hominoidea (gibbons and apes) and Cercopithecoidea (colobines and cercopithecines).

Identifying and Labeling the Causes of Supertree Polytomies.—Because veto methods are used for evaluating the topological congruence of source trees, and for measuring their degree of leaf overlap, the *PhySIC* program outputs labels on each polytomous node. A label "C" (standing for *Contradiction*) indicates that the polytomy results from contradictions among the source trees on phylogenetic relationships of corresponding taxa: proposing a resolution for the polytomy would contradict at least one source tree; i.e., would not respect the PC property. A label "I" (standing for *Induction*) indicates a lack of cross-information in the source trees: any dichotomous resolution of the clade would be at least partially arbitrary and thus would not respect the PI property. Note that a given label applies only to the node to which it is assigned but not to other nodes in its subtrees. For instance, in the primate genera supertree (Figure 13), the platyrrhine trifurcation (Atelidae, Pitheciidae, Cebidae) with a C label indicates that there is topological contradiction among the source trees about the sister-group relationships of these three families. However, the C label does not put the monophyly of Atelidae, Pitheciidae, and Cebidae into question. Note also that a same polytomy can be characterized by both C and I labels. This means that the inability of the supertree to propose a dichotomous resolution is partly due to a lack of taxonomic overlap, and partly due to contradictions. For example, Figure 13 shows that the clade *Cercopithecus*, *Erythrocebus*, *Chlorocebus*, and *Miopithecus* is tagged by both C and I, reflecting two problems. On the one hand, source topologies disagree about the placement of *Erythrocebus*: this genus is either related to *Cercopithecus* (as suggested by IRBP exon 1) or to *Chlorocebus* (*cf*. the TSPY and chromosome Xq13.3 markers, and the Alu characters of Xing et al. [2005]). On the other hand, the input trees analyzed here do not provide the information required to know whether *Miopithecus* is the sister group of *Cercopithecus*, or is that of *Erythrocebus* + *Chlorocebus*, or is the most basal genus in the clade.

Impact of the Robustness of Source Trees on Veto Supertree Resolution.—The number of clades retained from the original source trees depends on the bootstrap threshold imposed to select them for supertree inference. Choosing a low threshold thus increases the number of retained source clades and hence lowers the number of polytomies due to a lack of cross-information among source trees but increases the number of polytomies due to conflicts among source trees. Increasing the threshold has the opposite effect. The primate supertree of Figure 13 was obtained with BP $\geq$ 70%. Lowering the threshold to BP $\geq$ 50% or BP $\geq$ 60%, *PhySIC* yields a completely multifurcating supertree, due to weakly supported clades that conflict among source trees. When the bootstrap stringency is increased from the BP $\geq$ 70% to BP $\geq$ 80% threshold, a similar level of resolution in the genus level phylogeny is obtained with the exception of two additional polytomies: the first involves Indriidae (*Indri* + *Avahi* + *Propithecus* + *Paleopropithecus*) relative to other lemuriformes, and the second involves *Allenopithecus* relative to the *Cercopithecus* clade (white stars in Figure 13 refer to disappearing branches). Interestingly, increasing the threshold removes a topological conflict among *Lophocebus*, *Papio* and *Theropithecus*: with the PC property being satisfied, then the *PhySIC* supertree groups together the latter two genera. At the BP $\geq$ 90% threshold, 7 additional polytomies with respect to the BP $\geq$ 70% topology appear (Figure 13: black stars refer to node collapsing). This reflects the fact that less source nodes (i.e., the nodes of source trees) are available for supertree inference. The PI property is thus less often satisfied in the $PhySIC_{PC}$ supertree, leading to a greater number of irresolutions in the *PhySIC* supertree.

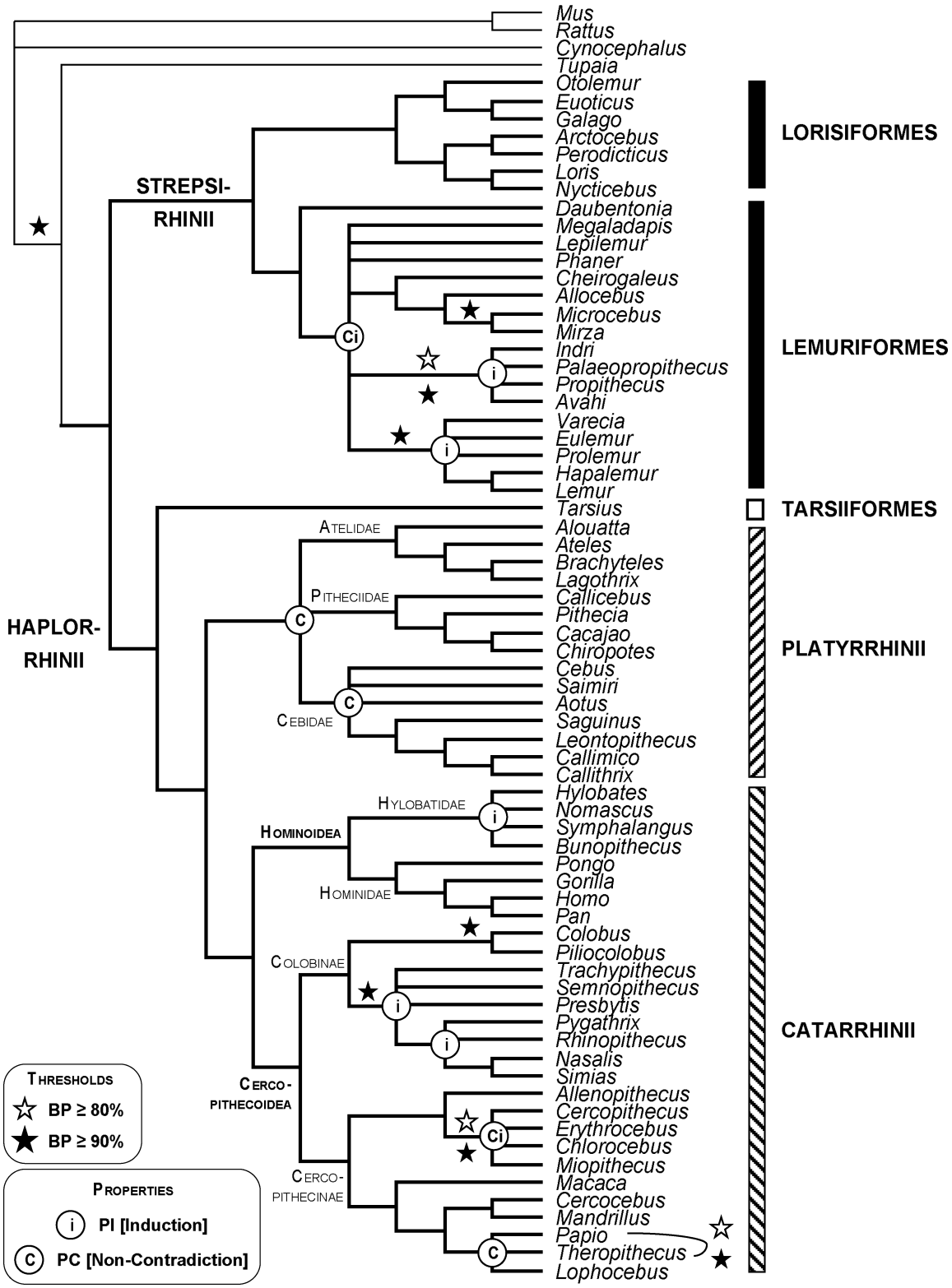

FIGURE 13. Primate *PhySIC* supertree including 95% of all extant genera and containing no contradiction or arbitrary resolution with respect to the source trees, as defined by the PI and PC properties. The genus-level primate supertree has been reconstructed from 24 molecular source trees restricted to nodes supported by more than 70% bootstrap. When the bootstrap threshold is increased to 80%—respectively, 90%—the supertree topology changes: disappearing branches as well as one appearing clade (*Papio* + *Theropithecus*) are indicated by white—respectively, black—stars. Polytomies are labeled by tags pointing out the properties (PI, PC, or both) that would not be satisfied if the corresponding clade was more resolved. The taxonomic frame and clade names for Primates are given. Hatched rectangles represent Anthropoidea (Platyrrhini + Catarrhini). White and black rectangles respectively represent Haplorrhini (Tarsiiformes + Anthropoidea) and Strepsirrhini (Lorisiformes + Lemuriformes).

Overall, two reasons can lead the *PhySIC* method to propose a poorly resolved supertree. First, it is possible that the source trees contain too little cross-information for the method to decide how the taxa of the respective source trees branch relatively to each other. In this case, all methods, including voting methods, will produce unresolved supertrees. Obtaining more resolved supertrees can then only be achieved by adding new source trees containing new clades on the key taxa. The second reason why the *PhySIC* supertree can lack resolution is the presence of topological conflicts among source trees. Like other veto methods, *PhySIC* is very sensitive to incongruences in the source trees. Thus, to obtain a well-resolved tree, a preliminary process whereby unreliable clades are collapsed in the source trees is usually necessary before applying the method. This collapsing can be done on the basis of the support values provided on the clades by most phylogenetic inference methods (e.g., bootstrap values, Bayesian posterior probabilities, Bremer support). We showed that a well-resolved supertree of Primates can be obtained with such an approach from a nontrivial number of gene trees. Note that on some data sets, contradicting clades showing high support values can occur, e.g., due to lateral gene transfers. In such cases, veto methods will still produce unresolved supertrees (as long as they are not allowed to exclude rogue taxa). This can be seen as a drawback or as a way to pinpoint such events. In such cases, outlier source trees can be identified (Shimodaira and Hasegawa, 1999; Lerat et al., 2003) and then curated or removed from the collection of source trees, leading to a more resolved supertree.

CONCLUSION

Veto supertree methods are of interest for combining source topologies containing reliable clades. Their study also brings insight for the characterization of what we expect from voting methods. Indeed, when source trees are not conflicting, there is no fundamental difference between the two approaches. In such cases, veto and voting approaches should lead to *reasonable* supertrees. What *reasonable* means can be characterized by several formal properties. In the present work, we showed pitfalls of some previously proposed supertree properties, and also proposed new properties. In the general case of conflicting source trees, we believe there is still room for improvement, e.g., detecting arbitrary clades of a supertree even when it partially conflicts with some source trees, as usually happens in the voting context. With the new theoretical material at hand we believe that this is a reasonable goal.

ACKNOWLEDGEMENTS

We thank J. Cotton, R. Page, O. Bininda-Emonds, and an anonymous reviewer for many invaluable remarks on a first version of this manuscript. This work has been supported by the "ACI Informatique-Mathématique-Physique en Biologie Moléculaire [ACI IMP-Bio]," by the "Action incitative BIOSTIC-LR," by IFR119 "Biodiversité Continentale Méditerranéenne et Tropicale" (Montpellier), and by the Research Networks Program in BIOINFORMATICS of the High Council for Scientific and Technological Cooperation between France and Israel. This publication is the contribution no. 2007-053 of the Institut des Sciences de l'Évolution de Montpellier (UMR 5554–CNRS). We thank V. Lefort for creating the web site of PhySIC.

REFERENCES

Adams, E. 1972. Consensus techniques and the comparison of taxonomic trees. Syst. Zool. 21:390–397.

Aho, A. V., Y. Sagiv, T. G. Szymanski, and J. D. Ullman. 1981. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. SIAM J. Comp. 10:405–421.

Bandelt, H.-J., and A. W. M. Dress. 1986. Reconstructing the shape of a tree from observed dissimilarity data. Adv. Appl. Math. 7:309–343.

Baum, B. R. 1992. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. Taxon 41:3–10.

Baum, B. R. and M. A. Ragan. 2004. The MRP method. Pages 17–34 *in* Phylogenetic supertrees: Combining information to reveal the Tree of Life (O. Bininda-Emonds, ed.). Kluwer, The Netherlands.

Berry, V., and O. Gascuel. 2000. Inferring evolutionary trees with strong combinatorial evidence. Theor. Comput. Sci. 240:217–298.

Berry, V., S. Guillemot, F. Nicolas, and C. Paul. 2005. On the approximation of computing evolutionary trees. In Proceedings of the 11th International Computing and Combinatorics Conference (COCOON'05) (L. Wang, ed.). LNCS. Springer. 3595:115–125.

Berry, V., and F. Nicolas. 2004. Maximum agreement and compatible supertrees. Pages 205–219 *in* Proceedings of CPM (S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, eds.), volume 3109 of LNCS.

Berry, V., and C. Semple. 2006. Fast computation of supertrees for compatible phylogenies with nested taxa. Syst. Biol. 55:270–288.

Bininda-Emonds, O. R. P. 2003. Novel versus unsupported clades: Assessing the qualitative support for clades in MRP supertrees. Syst. Biol. 52:839–848.

Bininda-Emonds, O. R. P. 2004a. The evolution of supertrees. Trends Ecol. Evol. 19:315–322.

Bininda-Emonds, O. R. P. 2004b. Phylogenetic supertrees: Combining information to reveal the tree of life, volume 4 of Computational biology series. Kluwer, The Netherlands.

Bininda-Emonds, O. R. P., and H. N. Bryant. 1998. Properties of matrix representation with parsimony analyses. Syst. Biol. 47:497–508.

Bryant, D. 1997. Building trees, hunting for trees and comparing trees: Theory and method in phylogenetic analysis. PhD thesis, Department of Mathematics, University of Canterbury, UK.

Bryant, D., and V. Berry. 2001. A structured family of clustering and tree construction methods. Adv. Appl. Math. 27:705–732.

Bryant, D., C. Semple, and M. Steel. 2004. Supertree methods for ancestral divergence dates and other applications. Pages 129–150 *in* Phylogenetic supertrees: Combining information to reveal the Tree of Life (O. Bininda-Emonds, ed.), volume 4 of Computational biology series. Kluwer, The Netherlands.

Chen, D., L. Diao, O. Eulenstein, D. Fernández-Baca, and M. J. Sanderson. 2003. Flipping: A supertree construction method. Pages 135–160 *in* Bioconsensus, volume 61 of Series in discrete mathematics and theoretic computer science, DIMACS. American Mathematics Society, Providence, Rhode Island.

Cotton, J. A., C. S. C. Slater, and M. Wilkinson. 2006. Discriminating supported and unsupported relationships in supertrees using triplets. Syst. Biol. 55:345–350.

Criscuolo, A., V. Berry, E. J. P. Douzery, and O. Gascuel. 2006. SDM: A fast distance-based approach for (super)tree building in phylogenomics. Syst. Biol. 55:740–755.

Daniel, P. 2004. Supertree methods: Some new approaches. Master's thesis, University of Canterbury, UK.

Daniel, P., and C. Semple. 2004. Supertree algorithms for nested taxa. Pages 151–171 *in* Phylogenetic supertrees: Combining information to reveal the Tree of Life (O. Bininda-Emonds, ed.), volume 4 of Computational biology series. Kluwer, The Netherlands.

Daubin, V., M. Gouy, and G. Perrière. 2002. A phylogenomic approach to bacterial phylogeny: Evidence of a core of genes sharing a common history. Genome Res. 12:1080–1090.

Davies, T. J., T. G. Barraclough, M. W. Chase, P. S. Soltis, D. E. Soltis, and V. Savolainen. 2004. Darwin's abominable mystery: Insights from a supertree of the angiosperms. Proc. Natl. Acad. Sci. USA 101:1904–1909.

Dekker, M. C. 1986. Reconstruction methods for derivation trees. Master's thesis, University of Amsterdam, The Netherlands.
Delsuc, F., H. Brinkmann, and H. Philippe. 2005. Phylogenomics and the reconstruction of the tree of life. Nat. Rev. Genet. 6:361–375.
Dutheil, J., S. Gaillard, E. Bazin, S. Glemin, V. Ranwez, N. Galtier, and K. Belkhir. 2006. Bio++: A set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics. BMC Bioinformatics 7:188.
Goloboff, P. A., 2005. Minority-rule supertrees? MRP, compatibility, and MinFlip may display the least frequent groups. Cladistics 21:282–294.
Goloboff, P. A., and D. Pol. 2002. Semi-strict supertrees. Cladistics 18:514–525.
Goodman, M., L. I. Grossman, and D. E. Wildman. 2005. Moving primate genomics beyond the chimpanzee genome. Trends Genet. 21:511–517.
Gordon, A. G. 1986. Consensus supertrees: The synthesis of rooted trees containing overlapping sets of labelled leaves. J. Classif. 3:335–348.
Grunewald, S., M. A. Steel, and M. S. Swenson. 2007. Closure operations in phylogenetics. Math Biosci. 208:521–537.
Guillemot, S., and V. Berry. 2007. Finding a largest subset of rooted triples identifying a tree is an NP-hard task. Technical report, LIRMM, University of Montpellier 2.
Guindon, S., and O. Gascuel. 2003. A simple, fast and accurate method to estimate large phylogenies by maximum-likelihood. Syst. Biol. 52:696–704.
Huson, D. H., S. M. Nettles, and T. J. Warnow. 1999. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. J. Comput. Biol. 6:369–386.
Karanth, K. P., T. Delefosse, B. Rakotosamimanana, T. J. Parsons, and A. D. Yoder. 2005. Ancient DNA from giant extinct lemurs confirms single origin of Malagasy primates. Proc. Natl. Acad. Sci. USA 102:5090–5095.
Lapointe, F.-J., and G. Cucumel. 1997. The average consensus procedure: Combination of weighted trees containing identical or overlapping sets of taxa. Syst. Biol. 46:306–312.
Lerat, E., V. Daubin, and N. A. Moran. 2003. From gene trees to organismal phylogeny in prokaryotes: The case of the gamma-proteobacteria. PLoS Biol. 1:101–109.
Margush, T., and F. McMorris. 1981. Consensus n-trees. Bull. Math. Biol. 43:239–244.
Neumann, D. A. 1983. Faithful consensus methods for n-trees. Math. Biosci. 63:271–287.
Page, R. D. M. 2002. Modified mincut supertrees. Pages 537–552 *in* Proceedings of the 2nd International Workshop on Algorithms in Bioinformatics (WABI'02) (R. Guigó and D. Gusfield, eds.). Springer, Rome, Italy.
Pisani, D., and M. Wilkinson. 2002. Matrix representation with parsimony, taxonomic congruence, and total evidence. Syst. Biol. 51:151–155.
Poux, C., P. Chevret, D. Huchon, W. W. de Jong, and E. J. P. Douzery. 2006. Arrival and diversification of caviomorph rodents and platyrrhine primates in South America. Syst. Biol. 55:228–244.
Poux, C., and E. J. P. Douzery. 2004. Primate phylogeny, evolutionary rate variations, and divergence times: A contribution from the nuclear gene IRBP. Am. J. Phys. Anthropol. 124:1–16.
Purvis, A. 1995a. A composite estimate of primate phylogeny. Philos. Trans. R. Soc. Lond. B Biol. Sci. 348:405–421.
Purvis, A. 1995b. A modification to Baum and Ragan's method for combining phylogenetic trees. Syst. Biol. 44:251–255.
Ragan, M. A. 1992. Matrix representation in reconstructing phylogenetic relationships among the eukaryots. Biosystems 28:47–55.
Roos, C., J. Schmitz, and H. Zischler. 2004. Primate jumping genes elucidate strepsirrhine phylogeny. Proc. Natl. Acad. Sci. USA 101:10650–10654.
Ross, H. A., and A. G. Rodrigo. 2004. An assessment of matrix representation with compatibility in supertree construction. *in* Phylogenetic supertrees: Combining information to reveal the tree of life (O. R. P. Bininda-Emonds, ed.), volume 4 of Computational biology series. Kluwer, The Netherlands.
Sanderson, M. J., A. Purvis, and C. Henze. 1998. Phylogenetic supertrees: Assembling the trees of life. Trends Ecol. Evol. 13:105–109.
Semple, C., and M. Steel. 2000. A supertree method for rooted trees. Discrete Appl. Math. 105:147–158.
Semple, C. and M. A. Steel. 2003. Phylogenetics, volume 24 of Oxford lecture series in mathematics and its applications. Oxford University Press, Oxford, UK.
Shimodaira, H., and M. Hasegawa. 1999. Multiple comparisons of log-likelihoods with applications to phylogenetic inference. Mol. Biol. Evol. 16:1114–1116.
Singer, S. S., J. Schmitz, C. Schwiegk, and H. Zischler. 2003. Molecular cladistic markers in New World monkey phylogeny (Platyrrhini, Primates). Mol. Phylogenet. Evol. 26:490–501.
Steel, M. A. 1992. The complexity of reconstructing trees from qualitative characters and subtree. J. Classif. 9:91–116.
Steel, M. A., A. W. M. Dress, and S. Böcker. 2000. Simple but fundamental limitations on supertree and consensus tree methods. Syst. Biol. 49:363–368.
Swofford, D. L. 2002. PAUP*: Phylogenetic analysis using parsimony (*and other methods). Version 4b10. Sinauer Associates, Sunderland, Massachusetts.
Thompson, J. D., T. J. Gibson, F. Plewniak, F. Jeanmougin, and D. G. Higgins. 1997. The clustalX windows interface: Flexible strategies for multiple sequence alignment aided by quality analysis tools. Nucleic Acids Res. 24:4876–4882.
Thorley, J. L., and M. Wilkinson. 2003. A view of supertrees methods. Pages 185–194 *in* Bioconsensus (M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, and F. S. Roberts, eds.), volume 61 of Discrete mathematics and theoretical computer science, DIMACS American Mathematics Society, Providence, Rhode, Island.
Wilkinson, M. 1994. Common cladistic information and its consensus representation: Reduced Adams and reduced cladistic consensus trees and profiles. Syst. Biol. 43:343–368.
Wilkinson, M., D. Pisani, J. A. Cotton, and I. Corfe. 2005. Measuring support and finding unsupported relationships in supertrees. Syst. Biol. 54:823–831.
Wilkinson, M., J. Thorley, D. Pisani, F.-J. Lapointe, and O. McInerney. 2004. Some desiderata for liberal supertree. Pages 227–246 *in* Phylogenetic supertrees: Combining information to reveal the tree of life (O. R. P. Bininda-Emonds, ed.), volume 4 of Computational biology series. Kluwer, The Netherlands.
Wilson, D. E., and D. M. Reeder. 2005. Mammal species of the world. Johns Hopkins University Press, Baltimore, Maryland.
Xing, J., H. Wang, K. Han, D. A. Ray, C. H. Huang, L. G. Chemnick, C.-B. Stewart, T. R. Disotell, O. A. Ryder, and M. A. Batzer. 2005. A mobile element based phylogeny of Old World monkeys. Mol. Phylogenet. Evol. 37:872–880.

First submitted 21 Agust 2006; reviews returned 23 November 2006; final acceptance 8 June 2007
Associate Editor: Olaf Bininda-Emonds

APPENDIX 1

Proof of Proposition 1

The proof is based on the fact that a set $\mathcal{R}$ identifies a tree T if and only if $rt(T) = cl(\mathcal{R})$ (Grunewald et al., 2006, lemma 2.1).

- $\Rightarrow$ By definition, $\mathcal{R}(T, \mathcal{T})$ only contains triplets on three-taxon sets for which there is a resolution in T. Moreover, PC ensures that any such triplet cannot be resolved in $\mathcal{R}(T, \mathcal{T})$ differently than that in T. It follows that $\mathcal{R}(T, \mathcal{T}) \subseteq rt(T)$. $\mathcal{R}(T, \mathcal{T})$ is therefore compatible and $cl(\mathcal{R}(T, \mathcal{T})) \subseteq cl(rt(T)) = rt(T)$. Meanwhile, having proved that $\mathcal{R}(T, \mathcal{T})$ is compatible, it is clear from PI that $cl(\mathcal{R}(T, \mathcal{T})) \supseteq rt(T)$.
- $\Leftarrow$ Having $\mathcal{R}(T, \mathcal{T})$ identifying T ensures that $\mathcal{R}(T, \mathcal{T})$ is compatible. On one hand, $cl(\mathcal{R}(T, \mathcal{T})) = rt(T)$ implies that for all $t \in rt(T)$, $t \in cl(\mathcal{R}(T, \mathcal{T}))$; i.e., $\mathcal{R}(T, \mathcal{T}) \vdash t$ and therefore PI holds. On the other hand, given $t \in rt(T)$, if $\mathcal{R}(T, \mathcal{T}) \vdash \bar{t}$ then, by definition of the closure, $\bar{t} \in cl(\mathcal{R}(T, \mathcal{T})) = rt(T)$ so that both t and $\bar{t}$ are in $rt(T)$, which is not possible. This proves that $\mathcal{R}(T, \mathcal{T}) \nvdash \bar{t}$ for any $t \in rt(T)$; i.e., PC holds.

Proof of Proposition 3

Let $AB|C \in rt(T)$ be a triplet of the output supertree T and consider the recursive step where the tree returned in line 3 hosts A, B in the same subtree T_i, whereas C is in another subtree, say T_j. This means that A, B are vertices in a connected component C_i of the current Aho graph G, while C is in another component C_j of $CC(G)$. If $\mathcal{R}$ contained $AC|B$ or $BC|A$, then C_i and C_j would not have been distinct connected components (in such cases, graph G would have contained the edge (A, C) or (B, C)). Thus, for any triplet t of $rt(T)$, we know that no triplet $\bar{t}$ is in $\mathcal{R}$; i.e., T does not directly contradict $\mathcal{R}$. This ensures that $Build_{PC}$ returns a tree satisfying PC (lemma 1).

Proof of Theorem 1

Correctness of the Algorithm.—The triplets present in the tree T_{PC} returned by $PhySIC_{PC}$ depend on the internal branches of this tree. Any internal branch of T_{PC} is created at line 10 during some recursive step, thus linking a subtree T_i to the root of the tree returned by this step. Thanks to Lemma 1, we just have to prove that every branch created at this line does not generate a triplet that directly contradicts $\mathcal{R}$.

At this line, either (i) $\mathcal{C}_{PC}$ corresponds exactly to the connected components of G (we have $|CC(G)| > 1$) and then, from Proposition 3, the triplets created are not in direct contradiction with $\mathcal{R}$; or (ii) $\mathcal{C}_{PC}$ corresponds to a partition of the vertices of G based on $\mathcal{R}'$. Two cases are then possible. In the first case (line 4), G' is connected and the current call returns a multifurcation that generates no triplet and hence does not invalidate PC. In the second case, the `Repeat` loop ensures that for each set of three taxa $A, B, C \in v(G)$, $\mathcal{C}_{PC}$ does not contain two elements C_i and C_j with $A, B \in C_i$ and $C \in C_j$ such that either $AC|B$ or $BC|A$ belongs to $\mathcal{R} = \mathcal{R}' \cup \mathcal{R}_{dc}$. Indeed, $AC|B$ (and $BC|A$) cannot be in $\mathcal{R}_{dc}$ since C_i would have been removed from $\mathcal{C}_{PC}$ (line 9). Moreover, if $AC|B$ (respectively $BC|A$) was in $\mathcal{R}'$, then A and C (respectively B and C) would have been in the same component of $\mathcal{C}_{PC}$ contradicting $C_i \neq C_j$. This proves that the tree T_{PC} built in line 10, and whose subtrees bijectively correspond to the elements of $\mathcal{C}_{PC}$, is such that no triplet of $rt(T_{PC})$ directly contradicts $\mathcal{R}$.

Time Complexity of the Algorithm.—The most time-consuming operations in $PhySIC_{PC}$ are the computation of $\mathcal{R}'|v(C_i)$ and G'_i (line 8), and that of the connected components of this graph (line 9). Obtaining $\mathcal{R}'|v(C_i)$ and constructing G'_i requires considering each triplet of $\mathcal{R}$ at most once and thus has a time complexity of $O(n^3)$. Determining the $CC(G'_i)$s costs $O(n^2)$ (which is the maximum number of edges for a graph with n vertices). During the whole set of recursive calls to $PhySIC_{PC}$, $\mathcal{C}_{PC}$ is modified at most $O(n)$ times (proportional to the number of clades of a tree with n leaves). Lines 8 and 9 are executed as many times as $\mathcal{C}_{PC}$ is modified; i.e., $O(n)$ times. Thus, for the whole set of recursive calls to $PhySIC_{PC}$, the computation time required by these critical lines is $O(n^4)$, which is also the complexity of the entire procedure.

Proof of Theorem 2

Correctness of the Algorithm.—we first prove that $PhySIC_{PI}$ always returns a tree, denoted by T_{PI}, and then that T_{PI} satisfies both PC and PI. By hypothesis, $PhySIC_{PI}$ is called with a tree T_{PC} that satisfies PC. Because $PhySIC_{PI}$ modifies this tree by only collapsing some of its branches (possibly none), the tree considered in any execution $Check_{PI}$ never directly contradicts $\mathcal{T}$. This ensures that the $Check_{PI}$ subroutine never exits in line 12; i.e., $PhySIC_{PI}$ always returns a tree. Moreover, being a contraction of T_{PC}, this tree satisfies PC.

$Check_{PI}$ differs from the *Identifies* algorithm in that `Return "tree not identified"` in the latter is replaced by line 14 in the former. Given a tree T and a set $\mathcal{R}$ of triplets, *Identifies*$(T, \mathcal{R})$ returns `yes` if $\mathcal{R}$ identifies T (otherwise it returns `no`) (Daniel, 2004, theorem 3.1.1). This ensures that when a call to $Check_{PI}(T, \mathcal{RPI})$ issued by $PhySIC_{PI}$ in line 11 collapses no branch of T_{PI}, then the set $\mathcal{R}$ identifies this tree. Since the tree T_{PI} returned by $PhySIC_{PI}$ is such that it is not modified by the last run of $Check_{PI}$, then T_{PI} is identified by $\mathcal{R}(T_{PI}, \mathcal{T}) = \mathcal{R}_{PI}$. In other words, T_{PI} satisfies both PI and PC for $\mathcal{T}$ (Prop. 1).

Time Complexity of the Algorithm.—As for $PhySIC_{PC}$, the most time-consuming operations done by $PhySIC_{PI}$ are the construction of the Aho graph G_{ij} and the computation of its connected components in the $Check_{PI}$ subroutine. The G_i graphs that may be used in $Check_{PI}$ can be precomputed in the $PhySIC_{PI}$ part of the pseudo-code (i.e., before calling $Check_{PI}$), knowing $rt(\mathcal{T})$ and the current tree T_{PI} to be examined in $Check_{PI}$. This preprocess clearly requires $O(n^4)$ time, since there are $O(n)$ such graphs (one for each clade of T), each of which is obtained by examining the $O(n^3)$ triplets of $\mathcal{R}(T_{PI}, \mathcal{T})$. Each G_{ij} graph can be obtained from a copy of the corresponding G_i graph, completed by the edges due to triplets $AB|C$ having $A, B \in C_i$ and $C \in C_j$. All the G_{ij} graphs required during the recursive calls to $Check_{PI}$ resulting from an execution of line 11 in $PhySIC_{PI}$ can also be precomputed in the $PhySIC_{PI}$ pseudo-code part. This can be done just before line 11, provided that $Check_{PI}$ is modified to end as soon as an edge is collapsed (line 14)—it is clear that this slight modification does not modify the correctness of the algorithm. Indeed, the only G_{ij}s that are then required by $Check_{PI}$ are those corresponding to two sibling clades C_i and C_j of the current T_{PI} tree. Computing all of these G_{ij}s before line 11 of $PhySIC_{PI}$ is done in $O(n^3)$ since each triplet $AB|C$ of $rt(T_{PI})$ adds an edge between A and B in the one and only graph G_{ij}, such that C_i and C_j are sibling clades in T_{PI} and $A, B \in C_i$ and $C \in C_j$.

Note also that the only information used by $Check_{PI}$ on graph G_i and G_{ij} is the number of their connected components. The total number of edges present in the G_{ij} graphs is in $O(n^3)$: precomputation of the number of connected components for this set of graphs is thus globally $O(n^3)$ time. As this has to be done at each pass of the `Repeat` loop, and as this loop is done at most $O(n)$ times (each pass results in the collapsing of one of the $O(n)$ clades of T), this part of the computation is globally (on the whole for $PhySIC_{PI}$) in $O(n^4)$ time. Determination of the number of connected components of each G_i is done only once just before the `Repeat` loop. For each of these $O(n)$ graphs, this requires examining $O(n^3)$ triplets. Thus, this preprocess also costs $O(n^4)$ time. The preprocesses done for G_i and G_{ij} graphs thus require $O(n^4)$ time and reduce the running time of $Check_{PI}$. The modification of $Check_{PI}$, consisting of returning to $PhySIC_{PI}$ as soon as an edge is collapsed, also simplifies the algorithm (e.g., the `Repeat` loop is no longer required).

Thanks to the preprocessing, the only time-consuming operation in $Check_{PI}$ for the current tree T_{PI} is the examination of the $O(n^2)$ pairs of sibling clades C_i and C_j of this tree. Operations performed for each of this pair of clades is in $O(1)$ (the number of connected components of useful graphs G, G_i and G_{ij} have been preprocessed). Because a new tree T_{PI} can only be obtained by collapsing one of the $O(n)$ edges (line 14) of T_{PI}, this can at most occur $O(n)$ times. Therefore, all executions of $Check_{PI}$ issued by a run of $PhySIC_{PI}$ are in $O(n^3)$ time. Thus, the whole complexity of the $PhySIC_{PI}$ algorithm is no more than the cost of the preprocessing; i.e., $O(n^4)$ time.

.