Reconstructing Weighted Phylogenetic Trees and Phylogenetic Networks Using Answer Set Programming

by Duygu Çakmak

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabanci University

August, 2010

Reconstructing Weighted Phylogenetic Trees and Phylogenetic Networks Using Answer Set Programming

Approved by:

Asst. Prof. Dr. Esra Erdem (Dissertation Supervisor)	....
Assoc. Prof. Dr. Ugur Sezerman	....
Dr. Alfredo Gabaldon	....
Asst. Prof. Dr. Balkiz Ozturk	....
Assoc. Prof. Dr. Yücel Saygın	....

Date of Approval: ....

Reconstructing Weighted Phylogenetic Trees and Phylogenetic Networks Using Answer Set Programming

Duygu ÇAKMAK

CS, Master's Thesis, 2010

Thesis Supervisor: Esra Erdem

Abstract

Evolutionary relationships between species can be modeled as a tree (called a phylogeny) whose nodes represent the species, internal vertices represent their ancestors and edges represent genetic relationships. If there are borrowings between species, then a small number of edges that denote such borrowings can be added to phylogenies turning them into (phylogenetic) networks. However, there are too many such trees/networks for a given family of species but no phylogenetic system to automatically analyze them. This thesis fulfills this need in phylogenetics, by introducing novel computational methods and tools for computing weighted phylogenies/networks, using Answer Set Programming (ASP). The main idea is to define a weight function for phylogenies/networks that characterizes their plausibility, and to reconstruct phylogenies/networks whose weights are over a given threshold using ASP solvers.

We have studied computational problems related to reconstructing weighted phylogenies/networks based on the compatibility criterion, analyzed their computational complexity, and introduced two sorts of ASP-based methods (representation-based and search-based) for computing weighted phylogenies/networks. Utilizing these methods, we have introduced a novel divide-and-conquer algorithm for computing large weighted phylogenies, and implemented a phylogenetic system (Phylo-ASP) based on it. We have also implemented a phylogenetic system (PhyloNet-ASP) for reconstructing weighted networks. We have shown the applicability and the effectiveness of our methods by performing experiments on two real datasets: Indo European languages, and Quercus species in Turkey. Moreover, we have extended our methods to computing weighted solutions in ASP and modified an ASP solver accordingly, providing a useful tool (clasp-w) for various ASP applications.

Çözüm Kümesi Programlama kullanarak Ağırlıklı Filogenetik Ağaçlar ve Ağların Çıkarımı

Duygu ÇAKMAK

CS, Master Tezi, 2010

Thesis Supervisor: Esra Erdem

Özet

Türlerin tarihsel evrim ilişkileri filogenetik ağaç olarak modellenebilir. Bu ağacın yaprakları türleri, aradaki düğümleri ataları ve kenarları genetik ilişkileri temsil eder. Türler arasında ödünç alma olduğu durumda, filogenetik ağaçlara bu tür ilişkileri gösteren az sayıda kenar eklenerek, filogenetik ağlara dönuştürülebilirler. Ancak verilen bir tür ailesi için oldukça fazla olası ağaç ve ağ olabilir ve bu ağaçları otomatik olarak analiz edebilecek bir sistem mevcut değil. Bu tez, çözüm kümesi programlama (ASP) kullanarak ağırlıklı filojeni ve filogenetik ağ hesaplamak amacıyla yeni hesaplama yöntemleri ve yazılım sistemleri geliştirerek filogenetik çalışmalarındaki bu ihtiyacı karşılamaktadır. Ağırlıklı filojeni hesaplamasının arkasındaki genel fikir, bir filojeninin ve filogenetik ağin ne kadar makul olduğunu gösteren bir ağırlık fonksiyonu kullanarak belirli bir ağırlığın üzerindeki filojenileri ve filogenetik ağları, ASP çözücülerini kullanarak hesaplamak.

Bu tez kapsamında, uyumluluk kriterine göre ağırlıklı filojeni ve filogenetik ağ çıkarımı ile ilgili hesaplama problemlerini inceledik, bu problemlerin hesaplama karmaşıklığını analiz ettik. Ağırlıklı filojenileri ve filogenetik ağları hesaplamak için iki tip (gösterime dayalı ve aramaya dayalı)

ASP'ye dayalı hesaplama yöntemi geliştirdik. Bu yöntemlerden yararlanarak, büyük veriler üzerinde filojeni çıkarımı yapmak için böl-ve-yönet yöntemine dayanan yeni bir algoritma geliştirdik. Bu algoritmaya dayalı yazılım sistemleri geliştirdik: ağırlıkli filojeni çıkarımı ve analizi yapan PHYLO-ASP, ve ağırlıklı filogenetik ağ çıkarımı yapan PHYLONET-ASP. İki gerçek veri üzerinden (Hint Avrupa dilleri ve Türkiye'deki meşe ağaçları) yaptığımız testler ile yöntemlerimizin ve yazılım sistemlerimizin etkinliğini gösterdik. Bunların yanında, yöntemlerimizi ASP'de ağırlıklı çözümler bulacak şekilde genelleştirdik ve bir ASP çözücüyü (CLASP-W) bu yöntemlere uygun bir şekilde değiştirerek birçok ASP uygulaması için yararlı bir araç sağladık.

Acknowledgements

I wish to express my gratitude to,

- Esra Erdem, for her invaluable supervision, patience and understanding,
- Thesis jury committee for their participation,
- Ozan Erdem, Halit Erdogan, Seyma Mutlu, Firat Tahaoglu, Berk Taner, Tansel Uras, and Can Yildizli for their help and friendship during my masters,
- last, but not the least, to my family, for being there when I needed them to be.

This thesis has been supported by TUBITAK Grant 107E229.

Contents

List of Figures

List of Tables

1 Introduction

Phylogenetics is the study of evolutionary relations between species based on their shared traits. These relations can be modeled as a tree (phylogeny). A phylogeny (or a phylogenetic tree) is a tree whose leaves represents the species, internal vertices represent their ancestors and edges in between represents the relationships between them. In some cases, phylogenies are not fully adequate to describe the evolutionary relations between species because they do not represent borrowing. We can represent these borrowings by adding a small number of edges to a phylogenetic tree and in this way, we obtain phylogenetic networks. There have been various studies on phylogenetics and phylogenetic networks (check [12] for a survey). There are also some phylogenetic systems that can reconstruct phylogentic trees and phylogenetic networks such as PHYLIP[1]. However, there may be many many possible phylogenies (resp. phylogenetic networks) for a given set of taxonomic units, with the same number of incompatible characters. In such cases, experts analyze the phylogenies (resp. phylogenetic networks) manually and identify some more plausible than others. Instead of the identification of the phylogenies (resp. phylogenetic networks) manually, we have studied finding more desirable phylogenies (resp. phylogenetic networks) by defining weight measures to reflect their plausibility and computing weighted phylogenies (resp. phylogenetic networks). For instance, while reconstructing phylogenies, if each phylogeny is assigned a weight that characterizes the expected groupings with respect to some archeological evidence, then finding a phylogeny of higher weight over some threshold might be more desirable To

[1]http://evolution.gs.washington.edu/phylip.html

reconstruct weighted phylogenies and weighted networks, we have extended the results of [12] [35]. In [12], [35] and in this thesis, phylogeny reconstruction is studied with respect to the compatibility criterion [19]. According to the compatibility criterion, the goal is to reconstruct a phylogeny with the maximum number of "compatible" characters. Intuitively, a character is compatible if it evolves without backmutation (i.e., it does not evolve from one state to another and then back to the earlier state) or parallel evolution (i.e., if no state appears independently in different lines of descent). So this approach is suitable for the datasets without backmutation. Therefore, it is not suitable for genomic data.

We have used Answer Set Programming (ASP) to reconstruct weighted phylogenies and weighted phylogenetic networks. ASP is a declarative programming paradigm oriented towards difficult search problems. It is originated from answer set semantics and based on computing models. The idea behind answer set programming is to represent a computational problem in terms of theories such that the models of these theories correspond to the solutions of the problem. The models of these theories are called answer sets of the problem. The answer sets of a problem can be computed using answer set solvers, such as CLASP[2]. Choosing ASP for phylogeny and phylogenetic network reconstruction in this thesis has 2 main reasons: First, we need the definition of reachability of a vertex from an other vertex, for example, to ensure the connectedness of the vertices from the root in a tree. Also, in phylogenetic networks, there may be loops in the graph (due to bidirectional lateral edges); and we check the reachability of a vertex from another vertex

[2]http://www.cs.uni-potsdam.de/clasp/

for compatibility check. In ASP, we can define reachability easily by making use of recursive definitions.

The main contributions of this thesis can be summarized as follows:

- We have defined various optimization and decision problems for computing weighted phylogenies and phylogenetic networks and analyzed their computational complexity.

- We have introduced two sorts of computational methods to compute weighted phylogenies and phylogenetic networks: the first class of methods suggests modifying the ASP representation of the problem to compute weighted phylogenies using an existing ASP solver and the other class suggests modifying the search algorithm of the answer set solver to compute weighted phylogenies incrementally based on modifying the search algorithm of an answer set solver CLASP. In the representation-based method, weight measure is defined in ASP. In the search-based method, weight measure is defined externally in C++.

- Based on these methods, in order to compute weighted phylogenies for large datasets efficiently, we have introduced a novel divide-and-conquer approach for computing weighted phylogenies by inferring its smaller subtrees. This approach also makes use of domain-specific information provided by the experts.

- We have generalized the representation-based method and the search based-method, to compute weighted solutions in ASP so that they can be applicable to other domains.

- We have implemented the search-based method to compute weighted solutions in ASP, by modifying the search algorithm of the answer set solver CLASP (and called it CLASP-W).

- Based on the divide-and-conquer approach for computing weighted phylogenies , we have implemented a fully automated system (called PHYLO-ASP) to reconstruct and analyze phylogenies, utilizing CLASP-W. We have also implemented a system called PHYLONET-ASP for reconstructing weighted phylogenetic networks.

- We have shown the applicability of our methods by performing experiments on two real datasets (Indo European languages and Quercus species) using PHYLO-ASP and PHYLONET-ASP.

- To apply our method to real datasets, we have defined new weight measures for phylogenies and phylogenetic networks.

The significance of our contributions both from the point of view of ASP and from the point of view of phylogenetics can be summarized as follows:

- There is no phylogenetic system that can help experts to order phylogenies with respect to a weight measure that characterizes their plausibility considering also some domain-specific information.

- There is no answer set solver that can compute weighted solutions incrementally, where the weight function is defined externally in C++.

In the following, first we introduce ASP (Chapter 2) and then explain our methods for computing weighted phylogenies and phylogenetic networks in

ASP (Chapter 3 and Chapter 4). Next, we discuss related work (Chapter 5) and conclude with a discussion of future work (Chapter 6).

2 Answer Set Programming

Answer Set Programming(ASP) [59] [65] [56] is a declarative programming paradigm oriented towards solving difficult search problems [57]. It is originated from "answer set semantics" [46] and based on computing models. The idea behind ASP is to represent a computational problem as an ASP program whose models ("answer sets") correspond to the solutions of the problem. The answer sets for a program can be computed by ASP solvers such as CLASP.

In the following, we introduce the syntax of ASP programs and define the concept of an answer set for an ASP program. Then we give a list of some applications that use ASP. After that we describe the answer set solver CLASP and its algorithm to find answer sets. Finally we explain how to modify CLASP's algorithm to find "weighted answer sets".

2.1 ASP Programs under the Answer Set Semantics

The syntax of ASP programs under the answer set semantics is defined as follows.

We begin with a set of propositional symbols, called *atoms*. A *literal* is an expression of the form A or $\neg A$, where A is an atom. A *rule element* is an expression of the form L or *not* L, where L is a literal. A *rule* is an ordered pair

$$Head \leftarrow Body \tag{2.1}$$

where *Head* is a finite set of literals, and *Body* is a finite set of rule elements. If

$$Head = \{L_1, ..., L_k\}$$

and

$$Body = \{L_{k+1}, ..., L_m, not\, L_{m+1}, ..., not\, L_n\}$$

$(0 \leq k \leq m \leq n)$ then we will write (2.1) as

$$L_1; ...; L_k \leftarrow L_{k+1}, ..., L_m, not\, L_{m+1}, ..., not\, L_n. \tag{2.2}$$

If the body is empty, we will sometimes drop $\leftarrow$; a rule with the empty body and one literal in the head is called a *fact*. If the head is empty, we will sometimes denote it by $\bot$; a rule with the empty head is called a *constraint*. A *program* is a set of rules. A program is called *nondisjunctive* if, in every rule, $k \leq 1$. We denote the set of literals in the language of a program Π by $lit(\Pi)$.

We say that a consistent set X of literals is *closed under* Π if, for every rule (2.2) in Π,

$$\{L_1, ..., L_k\} \cap X \neq \emptyset \tag{2.3}$$

whenever

$$\{L_{k+1}, ..., L_m\} \subseteq X \tag{2.4}$$

and

$$\{L_{m+1}, ..., L_n\} \cap X = \emptyset \tag{2.5}$$

This definition of closure corresponds to the definition of closure introduced in [45], [46]. for programs without negation as failure.

Let Π be a program without negation as failure. Then we say that X is an answer set for Π iff X is a minimal set closed under Π. For instance, the answer sets for

$$p; q \tag{2.6}$$

are $\{p\}$ and $\{q\}$.

Now consider a program Π that may contain negation as failure. The *reduct* of Π relative to a consistent set X of literals, as defined in [45], [46] is obtained from Π.

- by deleting each rule (2.2) that does not satisfy (2.5) and
- by replacing each remaining rule (2.2) by

$$L_1; ...; L_k \leftarrow L_{k+1}, ..., L_m. \tag{2.7}$$

This program will be denoted by Π^X. For instance consider the program

$$\begin{array}{r} p; q \\ \neg r \leftarrow not\, p. \end{array} \tag{2.8}$$

The reduct of this program relative to $\{p\}$ is (2.6).

We say that X is an answer set for a program Π iff X is an answer set for Π^X. Consider, for instance, program (2.8) and its reduct (2.6) relative to $\{p\}$. Since {p} is an answer set for (2.6), this is an answer set for program (2.8) as well. It is easy to check if $\{q, \neg r\}$ is an answer set for program (2.8) too.

Answer set definition is extended to programs with “choice rules” in [66]. For example, a choice rule

$$\{p, q\} \leftarrow p. \tag{2.9}$$

intuitively means that if p is included in the answer set then choose arbitrarily which of the atoms p, q to include in the answer set.

In answer set programming, due to its nonmonotonicity, the set of logical consequences does not necessarily shrink monotonically with increasing information (due to the use of the negation-as-failure operator). As an example, consider the programs

$$p \leftarrow \mathit{not}\, q. \tag{2.10}$$

$$\begin{aligned} p &\leftarrow \mathit{not}\, q. \\ q &\leftarrow \mathit{not}\, p. \end{aligned} \tag{2.11}$$

$$\begin{aligned} p &\leftarrow \mathit{not}\, q. \\ q &\leftarrow \mathit{not}\, p. \\ r &\leftarrow p. \\ r &\leftarrow q. \end{aligned} \tag{2.12}$$

Intuitively, (2.10) expresses that p is in the answer set in the absence of q. The answer set for this program is $\{p\}$ and the set of consequences is $\{p\}$. In

(2.11), we add one more rule to (2.10); the answer sets for this program are $\{p\}$ and $\{q\}$ and the set of consequences is emptyset. In (2.12), we add two more rules to (2.11).The answer sets of this program are $\{p, r\}$ and $\{q, r\}$ and the set of consequences is $\{r\}$. Therefore, as we add new rules to the previous programs to obtain new programs, the consequences do not increase as we expect from a monotonic formalism.

2.2 Applications of ASP

There are various applications of ASP as shown in Table 1. Here are some examples:

- *Decision Support Systems:* An ASP system has been developed to help flight controllers of space shuttle to solve some planning and diagnostic tasks [67].

- *Planning:* Since ASP can be used to solve classical planning problems, there are some systems, such as DLVK [31], implemented to solve planning problems in ASP. In addition, planning problems based on Hierarchical Task Networks (HTN) are studied in ASP [25].

- *Semantic Web:* Semantic Web applications make use of ASP in order to provide advanced reasoning services [18] [32] [79].

Table 1: *Applications of ASP*

Applications	**Applications**
planning [24] [56] [77]	theory update/revision [52]
preferences [72] [11]	diagnosis [30] [4]
learning [70]	description logics and semantic web [18] [32] [79]
probabilistic reasoning [5]	data integration and question answering [1] [55]
multi-agent systems [77] [78] [82]	common sense knowledge bases
circuit design	wire routing [36] [26]
decision support systems [67]	bounded model checking [48]
game theory [83] [84]	logic puzzles [39]
phylogenetics [29] [14] [35] [33]	systems biology [80]
combinatorial auctions [6]	haplotype inference [34] [81]
systems biology [80] [41] [71] [40]	automatic music composition [10] [9]
verification of cryptographic protocols [23]	assisted living [61] [62]
context [28]	

2.3 Answer Set Solvers

There are several ASP solvers which are used to compute the answer sets of an ASP program, such as SMODELS[3], CMODELS[4], DLV[5] and CLASP[6]. Let us describe CLASP's algorithm to compute answer sets.

[3]http://www.tcs.hut.fi/Software/smodels/
[4]http://userweb.cs.utexas.edu/users/tag/cmodels.html
[5]http://www.dbai.tuwien.ac.at/proj/dlv/
[6]http://www.cs.uni-potsdam.de/clasp/

2.3.1 CLASP

CLASP is a conflict-driven answer set solver [44] [43]. It uses the concepts of constraint processing and satisfiability checking [42]. CLASP does a DPLL–like [22] [60] branch and bound search to find an answer set to the given problem: at each level, it does propagation followed by backtracking or selection of new literals according to the current conflicts. The overall working principle of CLASP is shown in Algorithm 1. Three main steps are called repeatedly in the algorithm until an answer set is computed: PROPAGATION, RESOLVE-CONFLICT and SELECT. In the PROPAGATION step, the literals that are needed to be included in the answer sets (due to the current assignment and conflicts) are decided. The RESOLVE-CONFLICT step seeks to resolve the conflicts encountered with the previous step. In the case of a conflict existence, CLASP learns the conflict and does backtracking to an appropriate level. In the SELECT step, a new literal (based on some heuristics) is selected to continue search.

CLASP's branch and bound search differs from DPLL in some aspects: First of all, DPLL is for solving SAT problems. However, solutions to SAT may not correspond to the answer sets of the problems [58]. For example, consider the following answer set program $\{p \leftarrow q, q \leftarrow p\}$ whose answer set is $\emptyset$. This program can be translated into SAT as $(\neg q \vee p) \wedge (\neg p \vee q)$ whose models are $\{p\}, \{p, q\}, \emptyset$. On the other hand, CLASP decomposes ASP formulations into local inferences which are obtained by Clark completion of a program [20] and then uses DPLL search over the local inferences.

Algorithm 1 CLASP

```
Require: An ASP program $\Pi$
Ensure: An answer set $A$ for $\Pi$
  $A \leftarrow \emptyset$ {current assignment of literals}
  $\bigtriangledown \leftarrow \emptyset$ {set of conflicts}
  while No Answer Set Found do
    {propagate according to the current assignment and conflicts; update the current assignment}
    PROPAGATION($\Pi, A, \bigtriangledown$)
    if There is a conflict in the current assignment then
      RESOLVE-CONFLICT($\Pi, A, \bigtriangledown$) {learn and update the conflict set and do backtracking}
    else
      if Current assignment does not yield an answer set then
        SELECT($\Pi, A, \bigtriangledown$) {select a literal to continue search}
      else
        return A
      end if
    end if
  end while
```

2.4 Computing Weighted Solutions

In ASP, some problems may have many solutions. Moreover, the correspondence between the answer sets and the solutions may not be one-to-one; there may be many answer sets that denote the same solution. In such cases, one way to compute more desirable solutions is to assign weights to solutions, and then pick the distinct solutions whose weights are over a given threshold. For example, in a planning problem, the weight of a plan can be defined in terms of the costs of actions, and then the distinct plans whose weights are less than a given value can be computed. In puzzle generation, the weight of a puzzle instance can be defined by means of some difficulty measure, and then difficult puzzles whose weights are over a given value can be generated.

While computing such weighted solutions, there can be two types of methods: the representation-based methods and the search-based methods [14].

In the representation-based methods, ASP representation of the problem can be modified to compute weighted solutions. In some cases, some aggregates (e.g., sum,count) can be used to compute the weight of a solution [73, 38, 76]; while in some others, a weight formulation can be added explicitly to the ASP representation.

In the search-based methods, instead of modifying the ASP representation of the problem, the weight function can be defined externally and the search algorithm of the answer set solver can be modified to compute weighted solutions as in [14].

We have modified the search algorithm of the answer set solver CLASP to compute weighted solutions with the search-based method. We call the modified version CLASP-W. The modified algorithm can be seen in Algorithm 2. The procedure WEIGHT-ANALYZE is the weight measure of a given problem and needs to be implemented according to that given problem.

The WEIGHT-ANALYZE function is called at each step of the search; therefore, it should be capable of identifying the partial solution formed by the currently selected literals, and measuring the weight of that partial solution. Since a partial solution may extend to many complete solutions, the WEIGHT-ANALYZE function computes instead an upper bound (resp. a lower bound) for the weight of a solution that extends the current partial solution. Computing an exact upper bound (resp. a lower bound) might be hard and inefficient; therefore, one may be interested in implementing a heuristic function that computes an approximate upper bound (resp. lower bound) for a solution. To guarantee to find a complete solution, the heuristic function shall be admissible. In other words, the upper bound (resp. lower bound) computed by the heuristic function shall be greater (resp. less) than or equal to the exact upper bound (resp. lower bound). If this is not the case, then we have a risk of missing a solution. Once we define the WEIGHT-ANALYZE function to estimate the weight of a solution, we check whether the estimated weight is less (resp. greater) than the given weight threshold w. If the upper bound (resp. the lower bound) computed by the heuristic function is already less (resp. greater) than the given weight threshold w, then there is no solution that can be characterized by the current assignment of literals and that has a weight greater (smaller) than w. Therefore; we set the current assignment of literals as conflict in that case. After setting an assignment as conflict, CLASP-W learns that assignment and does backtracking and never selects those assignment in the further stages of the search.

Algorithm 2 CLASP-W

```
Require: An ASP program $\Pi$ and a nonnegative integer $w$
Ensure: An answer set for $\Pi$, that describes an at least (resp. at most)
  $w$-weighted solution
  $A \leftarrow \emptyset$ {current assignment of literals}
  $\bigtriangledown \leftarrow \emptyset$ {set of conflicts}
  while $A$ does not represent an answer set do
  {propagate according to the current assignment and conflicts;update the
    current assignment}
    NOGOOD-PROPAGATION$(\Pi, A, \bigtriangledown)$
  {compute an upper (resp. lower) bound for the weight of a solution that
    contains $A$}
    $weight \leftarrow$ WEIGHT-ANALYZE$(A)$
  {if the upper bound $weight$ is less than the desired weight value $w$}
  {then no need to continue search to find an at least $w$-weighted solution}
    if There is a conflict in unit-propagation OR $weight < w$ then
      RESOLVE-CONFLICT $(\Pi, A, \bigtriangledown)$ {learn and update the conflict set and
      do backtracking}
    end if
    if Current assignment does not yield an answer set then
      SELECT$(\Pi, A, \bigtriangledown)$ {select a literal to continue search}
    else
      return A
    end if
  end while
  return false
```

3 Reconstructing Weighted Phylogenetic Trees using ASP

Cladistics (or phylogenetic systematics) developed by Will Henning [49, 50, 51] is the study of evolutionary relations between species (or "taxonomic" "unit") based on their shared traits. These relations can be modeled as a tree (phylogeny). A phylogeny (or a phylogenetic tree) is a tree whose leaves represent the species; internal vertices their ancestors; and edges in between, the relationships between them. There are two main approaches to cladistics: Character-based and distance-based. Our approach is character-based cladistics as in [12, 69].

In character-based cladistics, shared traits are "(qualitative) characters". A character is a trait in which taxonomic units can instantiate a variety of ways. If a character is instantiated by a set of taxonomic units in the same way, then these taxonomic units are assigned the same "state" of the character.

There are two main criteria in character-based cladistics: Maximum parsimony and maximum compatibility. In maximum parsimony [27], the aim is to minimize character state changes along the edges. In maximum compatibility [19], the aim is to maximize the number of "compatible" characters. Intuitively, a character is compatible if it evolves without backmutation[7] or parallel evolution.[8] We consider the latter criteria while reconstructing phylogenies.

[7]If a character evolves from one state to another and then back to the earlier state, then backmutation occurs in the evolution of that character.

[8] If a state appears independently in the different lines of descent, then parallel evolution occurs.

While reconstructing phylogenies, there may be many possible phylogenies for a given set of taxonomic units, with the same number of incompatible characters. In such cases, experts analyze the phylogenies manually and identify some more plausible than others. Instead of identifying the phylogenies manually, we aim to find more plausible phylogenies automatically. In order to do that, first we define some weight measures for the phylogenies to reflect their plausibility; then we introduce computational methods to compute weighted phylogenies over a certain weight threshold.

3.1 Preliminaries

Before we describe the problems related to weighted phylogenetic tree reconstruction, we need to introduce some definitions as in [12].

A *directed graph (digraph)* is an ordered pair $\langle V, E \rangle$, where V is a set and E is a binary relation on V. In a digraph $\langle V, E \rangle$, the elements of V are called *vertices*, and the elements of E are called the *edges* of the digraph. The *out-degree* of a vertex v is the number of edges (v, u) $(u \in V)$ and the *in-degree* of v is the number if edges (u, v) $(u \in V)$. A digraph $\langle V', E' \rangle$ is a subgraph of a digraph $\langle V, E \rangle$ if $V' \subset V$ and $E' \subset E$.

In a digraph $\langle V, E \rangle$, a path from vertex u to a vertex u' is a sequence $v_0, v_1, .., v_k$ of vertices such that $u = v_0$ and $u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $1 \leq i \leq k$. If there is a path from a vertex u to a vertex v, then we say that v is *reachable from* u. If V' is a subset of V, a path from u to v whose vertices belong to V' is a *path* from u to v in V'. If there exist a path from u to v in V', v is *reachable from* u *in* V'.

A *rooted tree* is a digraph with a vertex of in-degree 0, called the *root*, such that every vertex different from the root has in-degree 1 and is reachable from the root. In a rooted tree, a vertex of out-degree 0 is called a *leaf*.

A *phylogenetic tree* (or *phylogeny*) for a set of taxa is a finite rooted binary tree $\langle V,\ E\rangle$ along with two finite sets I and S and a function f from L x I to S, where L is the set of leaves of the tree. The set L represents the given taxonomic units, whereas the set V describes their ancestral units and the set E describes the genetic relationships between them. The elements of I are usually positive integers ("indices") that represent, intuitively, qualitative characters, and elements of S are possible states of these characters. The function f "labels" every leaf v by mapping every index i to the state $f(v,i)$ of the corresponding character in that taxonomic unit.

For a phylogeny (V,E,L,I,S,f), a state $s \in S$ is *essential* with respect to a character $j \in I$ if there exist two different leaves l_1 and l_2 in L such that $f(l_1,j) = f(l_2,j) = s$. A character $i \in I$ is *informative* if it has at least 2 essential states.

A character $i \in I$ is *compatible* with a phylogeny (V,E,L,I,S,f) if there exist a function $g : V$ x $i \rightarrow S$ such that

- For every leaf v of the phylogeny, $g(v,i) = f(v,i)$
- For every $s \in S$ if the set

  $V_{is} = \{x \in V : g(x,i) = s\}$

  is nonempty, then the digraph $\langle V,\ E\rangle$ has a subgraph with the set V_{is} of vertices that is a rooted tree.

A character is *incompatible* with a phylogeny if it is not compatible with that phylogeny. For example in Figure 1, the character "Hand" is compatible

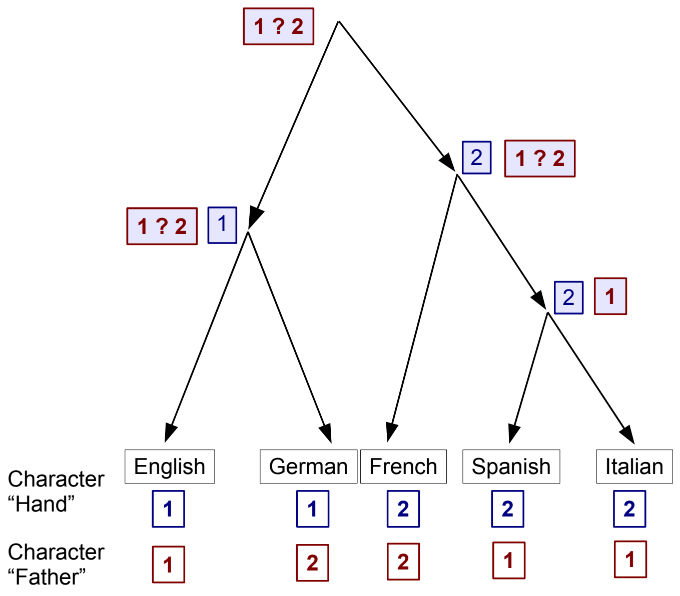


Figure 1: Compatible/Incompatible Character: The blue boxes denote the labels of the character Hand, and the red boxes denote the labeling of the character Father

with respect to the given phylogeny, since every unit with the same state is connected to each other with a tree. On the other hand, the character "Father" is incompatible, since there is no possible labeling of internal vertices that connects all the units which have the same labels.

3.2 Weighted Phylogenies

In phylogeny reconstruction, there may be many possible phylogenies with the same number of incompatible characters and some phylogenies may be more desirable than the others, from the experts' point of view. In such cases, one way to pick more desirable phylogenies without human intervention is to assign weights to phylogenies, and then pick the distinct phylogenies whose weights are over a given threshold.

Therefore, we have formulated several weight measures in order to compute weighted phylogenies with different data sets. There are two types of weight measures: domain-independent and domain-dependent. Domain independent weight measures do not require domain-specific information about the dataset, and therefore can be applied to any dataset. On the other hand, domain-dependent weight measures require domain-specific information. For example, experts usually provide information about how to group species. A group of species is called as a subgroup from now on. Although not as well-known as subgroup information, sometimes we may have further domain-specific information as to how the subgroups can be classified. A group of subgroup is called as a class from now on.

Domain Independent Weight Functions

Weight Measure 1 (W1) We define a weight measure in such a way that while minimizing the number of incompatible characters, we try to maximize the total weight of these characters.

Consider a phylogeny $P = (V, E, I, S, f)$. Let IC denote the characters in I that are informative and compatible with this phylogeny. The weight of a phylogeny P is the sum of the weights of all informative characters that are compatible with the tree:

$$weight_1(P) = \sum_{i \in IC} w(i) \tag{3.1}$$

The weight $w(i)$ of a character i is a nonnegative integer given as domain information.

Weight Measure 2 (W2) We define a weight measure in such a way that the phylogenies with the informative characters which have more essential states have more weight. The motivation behind this weight measure is that the characters with many essential states give more information as to how the species are related to each other.

Consider a phylogeny $P = (V, E, I, S, f)$ with leaves L. Let IC denote the characters in I that are informative and compatible with this phylogeny. The weight of a phylogeny P is the sum of the weights of all informative characters that are compatible with the tree:

$$weight_2(P) = \sum_{i \in IC} w(i) \tag{3.2}$$

The weight $w(i)$ of an informative character is defined as the number of leaves that are mapped to an essential state for that character:

$$w(i) = |\{l \in L : f(l, i) = s, i \text{ is informative, } s \text{ is essential}\}| \tag{3.3}$$

Domain Dependent Weight Functions

Weight Measure 3 (W3) Suppose that we are given some domain-specific information as to how the taxonomic units are grouped as "sub-groups" and "classes". Then we define a weight measure in such a way that the leaves that belong to the same class are grouped as close to each other as possible.

Consider a phylogeny $P = (V, E, I, S, f)$ with leaves L. The weight of phylogeny P is the sum of the weights of all vertices except its root r:

$$weight_3(P) = \sum_{v \in V/\{r\}} \varphi(v) \tag{3.4}$$

The weight $\varphi(v)$ of a vertex v is defined as follows:

1. We label the leaves with their class information.

2. We propagate the labels of the leaves up to the root and we label each internal vertex with the labels of its children.

3. We assign a weight to each vertex by comparing its labels with those of its sibling. To be able to compare the labeling of the vertices, we define the *contribution* $\varsigma(c, v)$ of a vertex v with respect to a label c as follows. Let *sibling*(v) denote the sibling of the vertex v, and Let *label*(v) denote the labels of the vertex v.

$$\varsigma(c, v) = \begin{cases} 0 & \text{if } c \notin label(sibling(v)), \\ 0 & \text{if } |label(v)| = \text{the total \# of classes}, \\ \frac{1}{|label(v)|} & \text{otherwise} \end{cases} \tag{3.5}$$

The weight $\varphi(v)$ of a vertex v is then the minimum of the following two values: the maximum value *maxContr*(v) of the contributions $\varsigma(c, v)$ over its labels c, and the maximum value *maxContr*$(sibling(v))$ of the

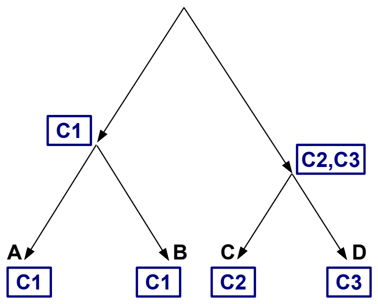


Figure 2: *A phylogenetic tree with class labels*

contribution $\varsigma(c', sibling(v))$ over its sibling's labels c'. That is,

$$\varphi(v) = min(maxContr(v), maxContr(sibling(v))). \qquad (3.6)$$

Let us give a small example to show this process. Consider the phylogenetic tree in Figure 2. The leaves are labeled with respect to the following class information: the leaves A and B are expected to be grouped in the same class, so they are labeled by C1; there is no information as to how C and D are expected to be grouped, so we label them by C2 and C3 respectively. Then we propagate these labels to their ancestors. We compute the weights of the vertices as follows: $\varphi(A) = 1$, $\varphi(B) = 1$, the other vertices have 0 weight. Then the weight of the phylogeny is 2.

Weight Measure 4 (W4) This weight measure is motivated by the definition of compatibility. We define it in such a way that, for each character, the leaves with the same character states are grouped as close to each other as possible.

Consider a phylogeny $P = (V, E, I, S, f)$ with leaves L, and suppose that the vertices of the phylogeny are labeled by a function $g : V \times I \rightarrow S$. Let IC denote the characters in I that are informative and compatible with this phylogeny. The weight of phylogeny P is the sum of the weights of all informative characters that are compatible with the tree:

$$weight_4(P) = \sum_{i \in IC} w(i) \tag{3.7}$$

The weight $w(i)$ of a character i is defined as the number of leaves having a sibling $sibling(l)$ with the same character state:

$$w(i) = |\{l : l \in L, f(l,i) = g(sibling(l), i)\}|. \tag{3.8}$$

Specific to the dataset, to get more plausible phylogenies, we can incorporate further domain-specific information. For instance, for Indo-European languages, historical linguist Don Ringe indicates that groupings of some languages are least likely to occur. If the to-be-reconstructed phylogenies have such odd groupings, we can reduce some amount from the total weight of the phylogeny, provided that the weight of a phylogeny is not negative.

3.3 Problem Definitions

We are interested in the following sorts of computational problems for computing weighted phylogenetic trees:

MAXIMUM COMPATIBILITY PROBLEM (MCP) Given three sets L, I, S and a function f, from LxI to S, finding a phylogeny (V, E, L, I, S, f) with the maximum number of compatible characters is called *the Max-*

imum Compatibility Problem (MCP).

n-COMPATIBILITY PROBLEM (n-CP) Given three sets L, I, S and a function f, and a non-negative integer n, decide the existence of a phylogeny (V, E, L, I, S, f) with at most n incompatible characters.

A phylogeny (V, E, L, I, S, f) is *perfect* if all characters in I are compatible with the phylogeny. Determining whether a phylogeny (V, E, L, I, S, f) is perfect is called *the Perfect Phylogeny Problem* (PPP). PPP is NP-hard [8, 64].

Proposition 1. *n-CP is NP-complete, if every character has binary states.*

Proof. **n-CP is in NP:** By verifying whether a given phylogeny has at most n incompatible characters in polynomial time, we will prove that n-CP is in NP. Intuitively, we have to do $|I|$ compatibility checks for each character. For each compatibility check, consider the algorithm in Algorithm 3.

The complexity of FINDLABELING is $O(|V|^2)$. The complexity of CHECKCONNECTEDNESS is $O(|V|)$. So, the complexity of the algorithm CHARCOMPATIBILITY is $O(|V|^2+|V|)$. Therefore, the overall algorithm has $O(|I|(|V|^2+|V|)) \approx O(|I||V|^2)$ complexity.

n-CP is NP-hard: By reducing the CLIQUE problem[9], which is NP-complete [53], to the n-CP, we can prove that the latter is NP-hard as in [85]. The main idea behind the reduction is that any pair of compatible character collection in n-CP should correspond to a set of vertices in the

[9]A graph $G = (V, E)$ and a positive integer $J < |V|$ is given. The problem is determine whether G contain a *clique* of at least size J, that is, a subset $V' \subset V$ such that $|V'| > J$ and every two vertices in V' are joined by an edge in E.

graph that forms a clique. We can reduce CLIQUE to n-CP in polynomial time as follows:

The number of vertices in a CLIQUE problem corresponds to the number of characters in n-CP. Three times the sum of the number of vertices in a clique correspond to the number of leaves in n-CP. The cardinality of clique is equal to n. We build a matrix $X = [X_{i,j}], 1 \leq i \leq |I|, 1 \leq j \leq |L|$ such that, X has a character column for each vertex in V, and three taxon-rows for each unordered pair of vertices in V. For each edge $(u, v) \notin E$, we set the row entries in column u for that edge to 011, and the row entries in column v to 110. All other entries in X are 0.

Two characters, C_1 and C_2, are *incompatible* if and only if all of three elements (1,0), (0,1), (1,1) are in $\{\bigcup_{1 \leq j \leq |L|, l_j \in L} (f(l_j, C_1), f(l_j, C_2))\}$. In other words, with respect to our reduced instance, the pair of characters that corresponds to vertices not joined by an edge in the graph are incompatible.

Since n-CP is both in NP and NP-hard, n-CP is NP-complete.

$\square$

Let w be a *weight* function that maps every phylogeny to a nonnegative integer. Then we define *the Maximum Weighted Compatibility Problem(MWCP)* as follows:

> MAXIMUM WEIGHTED COMPATIBILITY PROBLEM(MWCP) Given three sets L, I, S, a function f from $L \times I$ to S, a function *weight*, find a phylogeny (V, E, L, I, S, f) with the maximum weight.

Note that MWCP generalizes MCP: For instance, if we take $w(i) = 1$ for every $i \in I$, then the MWCP is a MCP.
MWCP can be converted into the following decision problems:

> w-WEIGHTED COMPATIBILITY PROBLEM(w-WCP)
>
> Given three sets L, I, S, a function f from $L \times I$ to S, a function *weight*, and a non-negative integer w, decide the existence of a phylogeny (V, E, L, I, S, f) whose weight is at least w.

Similarly, w-WCP generalizes kCP.

> w-WEIGHTED n-COMPATIBILITY PROBLEM(wn-WCP)
>
> Given three sets L, I, S, a function f from $L \times I$ to S, a function *weight*, and two non-negative integers n and w, decide the existence of a phylogeny (V, E, L, I, S, f) with at most n incompatible characters and whose weight is at least w.

Proposition 2. *wn-WCP is NP complete.*

Proof. **n-CP is in NP:** Membership follows from the fact that we can not only guess a candidate tree (V, E), but also check in polynomial time whether $w(V, E, L, I, S, f) \geq w$ and whether the phylogeny has at most c incompatible characters (Theorem 17 in [64]).

***n*-CP is NP-hard:** If we take $weight(S) = 1$ for every S, then wn-WCP is a n-CP. Hence it is at least as hard as n-CP. We have shown previously that n-CP is NP-complete. Therefore, wn-WCP is NP-hard.

Since wn-CP is both in NP and NP-hard, wn-CP is NP-complete.

□

Algorithm 3 CharCompatibility

```
INPUT: $(V, E, L, I, S, f)$ , $i \in I$
OUTPUT: COMPATIBLE / INCOMPATIBLE.
if FindLabeling $(i, V, E, L, S, f)$ == NO_LABELING then
  return INCOMPATIBLE
else
  $\langle g, count0, count1 \rangle$ := FindLabeling$(i, V, E, L, S, f)$
  if CheckConnectedness$(V, E, g, count0, count1, i)$ then
    return COMPATIBLE
  else
    return INCOMPATIBLE
  end if
end if
```

3.4 ASP Formulation

We describe the phylogeny reconstruction problem and weight measures in ASP as follows.

Algorithm 4 FindLabeling

```
INPUT: $i$, $V$, $E$, $L$, $S$, $f$
OUTPUT: $\langle g$, $count0$, $count1\rangle$ or NO_LABELING
for all $l \in L$ do
  $g(l) := f(l)$
end for
while there is $n \in V \setminus L$ such that $g(n)$ is not defined do
  // In the following, $\{n_1, n_2\}$ denote the children of $n$ and $n_s$ denotes the
  sibling of $n$.
  for all $n \in V \setminus L$ such that $g(n_1)$, $g(n_2)$ and $g(n_s)$ are defined do
    if CheckSiblings($g(n_1)$, $g(n_2)$, $g(n_s)$, $n_s$) == CONFLICT then
      $g(n) := g(n_s)$
      IncrementCounts($count0$, $count1$, $g(n)$)
    else if CheckSiblings($g(n_1)$, $g(n_2)$, $g(n_s)$, $n_s$) == NO_LABELING
    then
      return NO_LABELING
    else
      $g(n)$ := CheckSiblings($g(n_1)$, $g(n_2)$, $g(n_s)$, $n_s$)
      IncrementCounts($count0$, $count1$, $g(n)$)
    end if
  end for
  for all $n \in V \setminus L$ such that $g(n_1)$, $g(n_2)$ are defined, $g(n_s)$ is not defined
  do
    if CheckSiblings($g(n_1)$, $g(n_2)$, NOT_DEFINED, $n_s$) $\neq$ CON-
    FLICT && CheckSiblings($g(n_1)$, $g(n_2)$, NOT_DEFINED, $n_s$) $\neq$
    NO_LABELING then
      $g(n)$ := CheckSiblings($g(n_1)$, $g(n_2)$, NOT_DEFINED, $n_s$)
      IncrementCounts($count0$, $count1$, $g(n)$)
    else
      return NO_LABELING
    end if
  end for
end while
```

Algorithm 5 CheckSiblings

```
INPUT: $x_1$, $x_2$, $x_3$, $n_s$
OUTPUT: $x_1$ or CONFLICT or NO_LABELING
if $x_1 == x_2$ then
  return $x_1$
else if $n_s \neq$ NOT_DEFINED then
  if $g(n_s) \neq$ CONFLICT then
    return CONFLICT
  else
    return NO_LABELING
  end if
else
  return NO_LABELING
end if
```

Algorithm 6 IncrementCounts

```
INPUT: $count0$, $count1$, $state$
OUTPUT: $count0$, $count1$
if $state == count1$ then
  $count1{+}{+}$
else
  $count0{+}{+}$
end if
```

Algorithm 7 FindRoot

```
INPUT: $V$, $E$
OUTPUT: $v \in V$
return a node that has no incoming edge in $E$.
```

Algorithm 8 CountConnectedNodes

```
INPUT: $V$, $E$, $rootNode$, $nodeCount$
OUTPUT: $nodeCount$
for all children $n$ of $rootNode$ do
  CountConnectedNodes($V$, $E$, $n$, $nodeCount + 1$)
end for
return $nodeCount$
```

Algorithm 9 CheckConnectedness

```
INPUT: $V$, $E$, $g$, $count0$, $count1$, $i$
OUTPUT: CONNECTED / NOT_CONNECTED.
$V_0 := \{ v \in V | \, g(v) = 0\}$
$E_0 := \{\{x, y\} \in E | \, x, \, y \in V_0\}$
$V_1 := \{v \in V | \, g(v) = 1\}$
$E_1 := \{\{x, y\} \in E | \, x, \, y \in V_1\}$
$treeRoot$ := FindRoot($V$, $E$)
$root0$ := FindRoot ($V_0$, $E_0$)
$root1$ := FindRoot ($V_1$, $E_1$)
$nodeCount0$ := CountConnectedNodes($V_0$, $E_0$, $root0$, 1)
$nodeCount1$ := CountConnectedNodes($V_1$, $E_1$, $root1$, 1)
if $count0$ == $nodeCount0$ && $count1$ == $nodeCount1$ then
  return CONNECTED
else
  return NOT_CONNECTED
end if
```

3.4.1 Phylogeny Reconstruction

ASP formulation of phylogeny reconstruction is done in two parts as in [12]: In the first part, rooted binary trees whose leaves represent the given taxa are generated and in the second part, the rooted binary trees with more than n incompatible characters are eliminated.

In the first part, we make use of the reachability of a vertex from another vertex to ensure the connectedness of the vertices from the root of the phylogeny. That we can define reachability easily by making use of recursive definitions in ASP has played an important role in our choice (and [12]'s choice) of ASP to represent phylogeny reconstruction.

3.4.2 Weight Functions

There are several weight functions we have formulated in ASP, which are described in Subsection 3.2:

W1 We describe the weight of a phylogeny as an ASP program in two parts. Suppose that the schematic variables `PW`, `W` denote phylogeny weights, `C` denotes a character and `CW` denotes the user defined weight of an informative character.

In the first part, we define the weight of a phylogeny as the sum of the weights of characters compatible with it:

```
weightOfThePhylogeny(PW) :- addWeightsOfCharacters(PW,C), maxCharacter(C).

addWeightsOfCharacters(PW,0) :- compatible(0), weightOfCharacter(0,PW).
addWeightsOfCharacters(0,0) :- not compatible(0).

addWeightsOfCharacters(PW+CW,C) :- compatible(C), informative_character(C),
                   weightOfCharacter(C,CW), addWeightsOfCharacters(PW,C-1).
addWeightsOfCharacters(PW,C) :- not compatible(C), addWeightsOfCharacters(PW,C-1).
addWeightsOfCharacters(PW,C) :- not informative_character(C),
                   addWeightsOfCharacters(PW,C-1).
```

In the second part, we describe the weight constraint to ensure that the weight of the phylogeny is greater than or equal to w:

```
:- weightOfThePhylogeny(PW),PW<w.
```

W2 We describe the weight of a phylogeny as an ASP program in three parts. Suppose that the schematic variable `PW` denotes the phylogeny weight, `IC` denotes an informative character, `CW` denotes a character weight, and `C` denotes a character.

In the first part, we describe the weight `CW` of an informative character `IC` as follows:

```
weightOfChar(IC,CW) :- CW{leaf(V):f(V,IC,S):essential_state(IC,S)}CW.
```

In the second part, we define the sum of the weights of characters compatible with the phylogeny:

```
totalWeightOfChars(PW) :- addCharWeights(PW,C), maxChar(C).

addCharWeights(PW,1) :- compatible(1), weightOfChar(1,PW).
addCharWeights(0,1) :- not compatible(1).

addCharWeights(PW+CW,C) :- compatible(C),
  weightOfChar(C,CW), addCharWeights(PW,C-1).
addCharWeights(PW,C) :- not compatible(C), addCharWeights(PW,C-1).
```

In the third part, we describe the weight constraint to ensure that the weight of the phylogeny is greater than or equal to w:

```
:- weightOfPhylogeny(W), W<w.
```

W4 We describe the weight of a phylogeny as an ASP program in three parts. Suppose that the schematic variable `PW` denotes the phylogeny weight, `CW` denotes a character weight, and `C` denotes a character.

In the first part, we describe a leaf L as `valuedLeaf(L,C)` with respect to an informative character, if the sibling of L has the same character state with L, and we define the weight `CW` of an informative character with respect to `valuedLeaf(L,C)` as follows:

```
weightOfCharacter(C,CW) :- addWeightsOfCharacters(CW,C,k).


addWeightsOfCharacters(1,C,0) :- valuedLeaf(0,C).
addWeightsOfCharacters(0,C,0) :- not valuedLeaf(0,C).

addWeightsOfCharacters(CW+1,C,L+1) :- valuedLeaf(L+1,C),
                      addWeightsOfCharacters(CW,C,L), leaf(L), L<k.
addWeightsOfCharacters(CW,C,L+1) :- not valuedLeaf(L+1,C),
                      addWeightsOfCharacters(CW,C,L), leaf(L), L<k.

valuedLeaf(L,C) :- sibling(L,Y), f(L,C,S), g(Y,C,S), vertex(Y), leaf(L),
                 ic(C), state(S), L!=Y.
```

In the second part, we define the weight of the phylogeny as the sum of the weights of informative characters compatible with it:

```
weightOfThePhylogeny(PW) :- totalWeightOfCharacters(PW,c).


totalWeightOfCharacters(CW,0) :- weightOfCharacter(0,CW).
totalWeightOfCharacters(CW+PW,C+1) :- totalWeightOfCharacters(PW,C),
                                weightOfCharacter(C+1,CW), ic(C+1).
totalWeightOfCharacters(PW,C+1) :- totalWeightOfCharacters(PW,C),
                                weightOfCharacter(C+1,CW), not ic(C+1).
```

3.5 Computational Methods: Representation-Based vs. Search-Based

We have studied two different methods for reconstructing weighted phylogenies: Representation-based method and search-based method.

3.5.1 Representation-Based Method

In the representation-based method, we modify the representation of the problem to compute weighted phylogenies. In order to do that, we formulate the phylogeny reconstruction as an ASP program P as described in Subsection 3.4.1. Then we formulate the weight function as an ASP program W as described in the Subsection 3.4.2. Finally, we compute weighted phylogenies by computing the solutions of the ASP program $P \cup W$.

3.5.2 Search-Based Method

In the search-based method, in order to compute weighted phylogenies, instead of modifying the representation of the problem, we implement the weight measure externally as a C++ program and we modify the search algorithm of the answer set solver CLASP. The modified version of CLASP is called CLASP-W(Subsection 2.4).

In order to compute phylogenies with the search-based method, we have defined a heuristic function to estimate an upper bound for each weight function in Subsection 3.4.2:

Upper Bound for W1 Let A be a partially constructed phylogeny of P. Let I be the set of characters for P. Let NI_A be the set of uninformative characters for A. Let NC_A be the set of incompatible characters for A.

Then, we can define the heuristic function with respect to A and a set I of characters as follows:

$$UB_1(A, I) = \sum_{i \in I} w(i) - \sum_{i \in NI_A} w(i) - \sum_{i \in NC_A} w(i).$$

With this heuristic function (implemented as a C++ program) and the phylogeny reconstruction program of [12], CLASP-W can compute all correct solutions (i.e., phylogenies whose weight is at least w). In other words, this heuristic function ensures that the following holds for every phylogeny P computed in the end:

$$w \leq \mathit{weight}_1(P) \leq UB_1(A, I).$$

This result follows from $\mathit{weight}_1(P) \leq UB_1(A, I)$ (admissibility), and $w \leq UB_1(A, I)$ iff $w \leq \mathit{weight}_1(P)$ (correctness).

Proposition 3. *UB_1 is admissible.*

Proof. Let A be a partially constructed phylogeny of P. Let I be the set of characters for P. Let NI_A be the set of uninformative characters for A. Let NC_A be the set of incompatible characters for A. Let NI_P be the set of uninformative characters for P. Let NC_P be the set of incompatible characters for P. Let C_P be the set of compatible characters for P. Then we want to show that,

$$\sum_{i \in IC} w(i) \leq \sum_{i \in I} w(i) - \sum_{i \in NC_A} w(i) - \sum_{i \in NI_A} w(i). \tag{3.9}$$

Since by definition, $\sum_{i \in I} w(i) = \sum_{i \in NC_P} w(i) + \sum_{i \in C_P} w(i) + \sum_{i \in NI_P} w(i)$, we can rewrite 3.9 as:

$$\sum_{i \in IC} w(i) \leq \sum_{i \in C_P} w(i) + \sum_{i \in NC_P} w(i) - \sum_{i \in NC_A} w(i) + \sum_{i \in NI_P} w(i) - \sum_{i \in NI_A} w(i)$$

Since $IC \subseteq C_P$, then $\sum_{i \in IC_P} w(i) - \sum_{i \in C_P} w(i) \geq 0$. Since $NC_A \subseteq NC_P$, then $\sum_{i \in NC_P} w(i) - \sum_{i \in NC_A} w(i) \geq 0$. Since $NI_A \subseteq NI_P$, then $\sum_{i \in NI_P} w(i) - \sum_{i \in NI_A} w(i) \geq 0$. Therefore, $UB_1(A, I) \geq \mathit{weight}_1(P)$.

□

Upper Bound for W2 Let A be a partially constructed phylogeny of P. Let I be the set of characters for P. Let NI_A be the set of uninformative characters for A. Let NC_A be the set of incompatible characters for A.

Then, we can define the heuristic function with respect to A and a set I of characters as follows:

$$UB_2(A, I) = \sum_{i \in I} w(i) - \sum_{i \in NI_A} w(i) - \sum_{i \in NC_A} w(i).$$

This result follows from $weight_2(P) \leq UB_2(A, I)$ (admissibility(Proposition 4)), and $w \leq UB_2(A, I)$ iff $w \leq weight_2(P)$ (correctness).

Proposition 4. *UB_2 is admissible.*

Proof. Let A be a partially constructed phylogeny of P. Let I be the set of characters for P. Let NI_A be the set of uninformative characters for A. Let NC_A be the set of incompatible characters for A. Let NI_P be the set of uninformative characters for P. Let NC_P be the set of incompatible characters for P. Let C_P be the set of compatible characters for P. We want to show that

$$\sum_{i \in IC} w(i) \leq \sum_{i \in I} w(i) - \sum_{i \in NC_A} w(i) - \sum_{i \in NI_A} w(i). \tag{3.10}$$

Since by definition, $\sum_{i \in I} w(i) = \sum_{i \in NC_P} w(i) + \sum_{i \in C_P} w(i) + \sum_{i \in NI_P} w(i)$, we can rewrite 3.10 as:

$$\sum_{i \in IC} w(i) \leq \sum_{i \in C_P} w(i) + \sum_{i \in NC_P} w(i) - \sum_{i \in NC_A} w(i) + \sum_{i \in NI_P} w(i) - \sum_{i \in NI_A} w(i)$$

Since $IC \subseteq C_P$, then $\sum_{i \in C_P} w(i) - \sum_{i \in IC_P} w(i) \geq 0$. Since $NC_A \subseteq NC_P$, then $\sum_{i \in NC_P} w(i) - \sum_{i \in NC_A} w(i) \geq 0$. Since $NI_A \subseteq NI_P$, then $\sum_{i \in NI_P} w(i) -$

$\sum_{i \in NI_A} w(i) \geq 0$. Therefore, $UB_2(A, I) \geq weight_2(P)$.

$\square$

Upper Bound for W3 Let A be a partially constructed phylogeny of P. Let $sibling(v)$ denote the sibling $v \in V$ and $label(v)$ denote the label of $v \in V$. We define the heuristic function as follows with respect to the set of vertices V of P:

$$UB_{\varphi}(A, V) = \sum_{v \in V} \varphi'(v) \tag{3.11}$$

where $\varphi'(v)$ is defined as follows:

$$\varphi'(v) = \begin{cases} 1 & \text{if } label(v) = \emptyset \text{ or } v \notin V_P, \\ 1 & \text{if } sibling(v) \text{ is not yet defined,} \\ & \text{in } A \text{ or } label(sibling(v)) = \emptyset \text{ ,} \\ minC(v) & \text{otherwise} \end{cases} \tag{3.12}$$

and $minC(v)$ is defined as follows:

$$minC(v) = \min(maxContr(v), maxContr(sibling(v))). \tag{3.13}$$

This result follows from $weight_3(P) \leq UB_{\varphi}(A, V)$ (admissibility), and $w \leq UB_{\varphi}(A, V)$ iff $w \leq weight_3(P)$ (correctness).

Proposition 5. *UB_{φ} is admissible.*

To prove Proposition 5, we need the following lemmas, definitions and notation.

Let P be a phylogeny (V, E, L, I, S, f). We say that a phylogeny $X = (V_X, E_X, L_X, I_X, S_X, f_X)$ is *contained in* P (denoted $X \subseteq P$) if $V_X \subseteq V$, $E_X \subseteq E$, $L_X \subseteq L$, $I_X \subseteq I$, $S_X \subseteq S$, $f|_{L_X} = f_X$.

Let $X = (V_X, E_X, L_X, I_X, S_X, f_X)$ and $Y = (V_Y, E_Y, L_Y, I_Y, S_Y, f_Y)$ be two partial phylogenies contained in P. Let us denote by $label_X(v)$ (resp. $label_Y(v)$) for a vertex $v \in V$, the set of the labels of v in X (resp. Y). We say that X is *label-contained* in Y (denoted $X \subseteq_l Y$) if

- $X \subseteq Y$,
- for every $v \in V$, $label_X(v) \subseteq label_Y(v)$,
- $|E_{P_2} \setminus E_{P_1}| \leq 1$.

In the following, for each function h defined over partial phylogenies above, let us denote by h_Z the function h defined for a partial phylogeny Z.

Then, for these lemmas, let $P_1 = (V_{P_1}, E_{P_1}, L_{P_1}, I_{P_1}, S_{P_1}, f_{P_1})$ and
$P_2 = (V_{P_2}, E_{P_2}, L_{P_2}, I_{P_2}, S_{P_2}, f_{P_2})$ be two partial phylogenies of P, where $P_1 \subseteq_{l_1} P_2$.

Lemma 1. *For every vertex $v \in V$, if $label_{P_1}(v) = \emptyset$ or v is not in V_{P_1}, then $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.*

Proof. Take any $v \in V$. Assume that $label_{P_1}(v) = \emptyset$ or v is not in V_{P_1}. Under this assumption, we want to show $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$ for v. Because of the assumption, from the definition of φ'_{P_1}, $\varphi'_{P_1}(v) = 1$. Since 1 is the maximum value of φ'_{P_1} and φ'_{P_2}, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.

□

Lemma 2. *For every vertex $v \in V$, if $sibling_{P_1}(v) \notin V_{P_1}$ or $label_{P_1}(sibling_{P_1}(v)) = \emptyset$, then $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.*

Proof. Take any $v \in V$. Assume that $sibling_{P_1}(v) \in V_{P_1}$ or $label_{P_1}(sibling_{P_1}(v)) = \emptyset$. Under this assumption, we want to show $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$ for v. Because of the assumption, from the definition of φ'_{P_1}, $\varphi'_{P_1}(v) = 1$. Since 1 is the maximum value of φ'_{P_1} and φ'_{P_2}, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.

□

Lemma 3. *For a partial phylogeny P_1 of P and for every vertex $v \in V$, if the following conditions hold:*

(i) $label_{P_1}(v) \neq \emptyset$,

(ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

(iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) = \emptyset$,

then $\varphi'_{P_1}(v) = 0$.

Proof. Take any $v \in V$. Assume that (i), (ii) and (iii) hold for v. Under these assumptions, we want to show, $\varphi'_{P_1}(v) = 0$. Due to (i) and (ii),

$$\varphi'_{P_1}(v) = minC_{P_1}(v) = min(maxContr_{P_1}(v), maxContr_{P_1}(sibling_{P_1}(v))).$$

Due to (iii), since v and $sibling_{P_1}(v)$ do not share a label in P_1, $\forall l \in label_{P_1}(v)$, $\varsigma_{P_1}(l, v) = 0$ and $\forall l \in label_{P_1}(sibling_{P_1}(v))$, $\varsigma(l, sibling_{P_1}(v)) = 0$. That is $maxContr_{P_1}(v) = 0$ and $maxContr_{P_1}(sibling_{P_1}(v)) = 0$. Therefore, $\varphi'_{P_1}(v) = 0$. □

Lemma 4. *For every vertex $v \in V$, if the following conditions hold:*

(i) $label_{P_1}(v) \neq \emptyset$,

(ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

(iii) $label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v)) = \emptyset$.

then $\varphi'_{P_1}(v) = \varphi'_{P_2}(v)$.

Proof. Take any $v \in V$. Assume that (i), (ii), and (iii) hold for v. Under this assumption, we want to show $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$ for v.

Since (i), (ii), (iii) and $P_1 \subseteq_l P_2$, then by Lemma 3, $\varphi'_{P_1}(v) = 0$. Since (i), (ii), (iii) and $P_1 \subseteq_l P_2$, then by Lemma 3, $\varphi'_{P_2}(v) = 0$. Therefore, $\varphi'_{P_1}(v) = \varphi'_{P_2}(v)$.

□

Lemma 5. *For every vertex $v \in V$, if*

(i) $label_{P_1}(v) \neq \emptyset$*,*

(ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$*,*

(iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$*.*

$\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.

Proof. Take any $v \in V$. Assume that (i), (ii) and (iii) hold for v. Under this assumption, we want to show $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$ for v.

Consider two cases:

Case 1: $|label_{P1}(v)| =$ the total number of classes

– Due to (i) and (ii) and the propagation of labels described in the definition of *label*,

$$\begin{aligned} \varphi'_{P_2}(v) &= minC_{P_2}(v) \\ &= min(maxContr_{P_2}(v), maxContr_{P_2}(sibling_{P_2}(v))). \end{aligned}$$

Due to propagation of labels described in the definition of *label*, $|label_{P2}(v)| =$ the total number of classes and since $|label_{P2}(v)| =$ the total number of classes, due to the definition of ς, $\varphi'_{P_2}(v) = maxContr_{P_2} = 0$. Since 0 is the minimum value of φ'_{P_1} and φ'_{P_2}, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.

Case 2: $|label_{P1}(v)| <$ the total number of classes

Due to (i) and (ii),

$$\begin{aligned}\varphi'_{P_1}(v) &= minC_{P_1}(v)\\ &= min(maxContr_{P_1}(v), maxContr_{P_1}(sibling_{P_1}(v))).\\ &= min(\tfrac{1}{|label_{P_1}(v)|}, \tfrac{1}{|label_{P_1}(sibling_{P_1}(v))|})\end{aligned}$$

Due to (i) and (ii) and the propagation of labels described in the definition of *label*,

$$\begin{aligned}\varphi'_{P_2}(v) &= minC_{P_2}(v)\\ &= min(maxContr_{P_2}(v), maxContr_{P_2}(sibling_{P_2}(v))).\\ &= min(\tfrac{1}{|label_{P_2}(v)|}, \tfrac{1}{|label_{P_2}(sibling_{P_2}(v))|})\end{aligned}$$

Due to $P_1 \subseteq_l P_2$, since $|label_{P_1}(v)| \leq |label_{P_2}(v)|$ and $|label_{P_1}(sibling_{P_1}(v))| \leq |label_{P_2}(sibling_{P_2}(v))|$, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$.

□

Lemma 6. *If the following conditions hold for every vertex $v \in V$:*

(i) $label_{P_1}(v) \neq \emptyset$,

(ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

(iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) = \emptyset$,

(iv) $label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v)) \neq \emptyset$.

(v) $E_{P_2} = E_{P_1}$

then there exists a label $Z \in label_{P_2}(sibling_{P_2}(v))$ such that,

(a) $Z \in (label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v))$,

(b) $Z \notin label_{P_1}(v)$,

(c) $Z \in label_{P_1}(sibling_{P_1}(v))$,

(d) for some leaf child v_c *of* v, $Z \in label_{P_2}(v_c)$.

Proof. Take any $v \in V$. Assume that (i), (ii), (iii),(iv) and (v) hold for v. Due to (iv), (a) holds. Due to (iii) and $P_1 \subseteq_l P_2$, (b) holds. Due to $P_1 \subseteq_l P_2$, (c) holds. Due to (iv) and propagation of labels described in the definition of *label*, (d) holds. $\square$

Lemma 7. *If the following conditions hold for every vertex* $v \in V$*:*

(i) $label_{P_1}(v) \neq \emptyset$,

(ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

(iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) = \emptyset$,

(iv) $label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v)) \neq \emptyset$,

(v) $E_{P_2} \neq E_{P_1}$.

then

(a) there exists an edge $(v, v_c) \in E_{P_2}$ *but not in* E_{P_1} *and,*

(b) there exists a label $Z \in label_{P_2}(sibling_{P_2}(v))$ *such that,*

(b1) $Z \in (label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v))$

(b2) $Z \in label_{P_1}(sibling_{P_1}(v))$,

(b3) $Z \notin label_{P_1}(v)$

(b4) for some child v_c *of* v, $Z \in label_{P_2}(v_c)$.

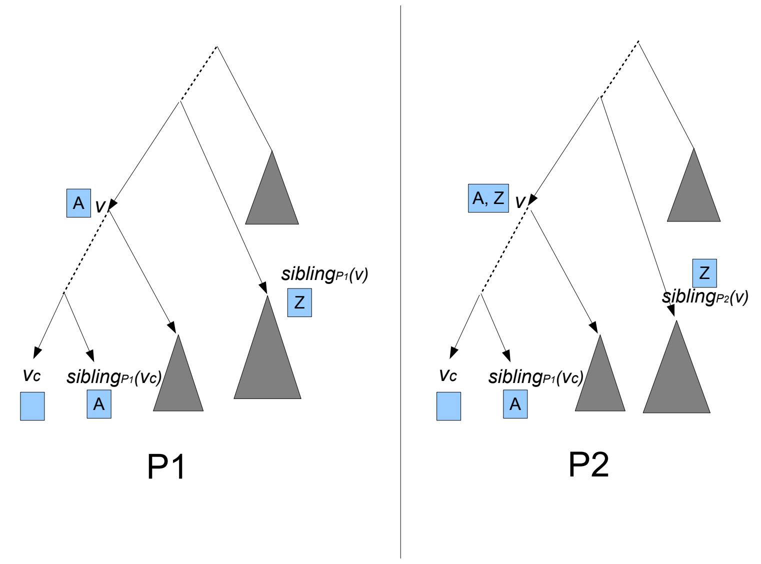


Figure 3: Case 1: The boxes next to the vertices denote their labels.

Proof. Take any $v \in V$. Assume that (i), (ii), (iii),(iv) and (v) hold for v. Due to (v), (a) holds. Due to (iv), (b1) holds. Due to $P_1 \subseteq_l P_2$, (b2) holds. Due to $P_1 \subseteq_l P_2$, (iii) and (iv), (b3) holds. Due to (iv) and the propagation of labels described in the definition of *label*, (b4) holds.

□

Lemma 8. *If the following conditions hold for every vertex $v \in V$:*

(i) $label_{P_1}(v) \neq \emptyset$,

(ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

(iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) = \emptyset$,

(iv) $label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v)) \neq \emptyset$.

then

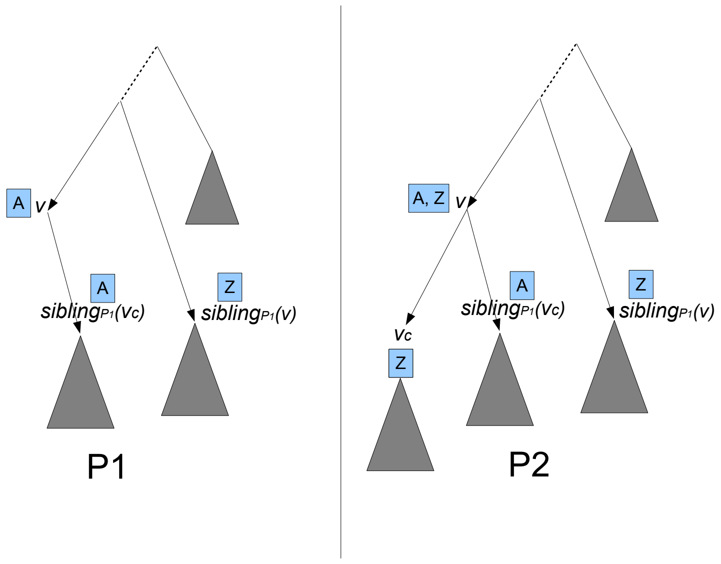


Figure 4: Case 2: The boxes next to the vertices denote their labels.

(a) $\varphi'_{P_2}(v) \geq \varphi'_{P_1}(v)$,

(b) There exists a child v_c *of* v, $\varphi'_{P_2}(v_c) \leq \varphi'_{P_1}(v_c)$,

(c) $(\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c)) - (\varphi'_{P_2}(v) - \varphi'_{P_1}(v)) \geq 0$.

Proof. Take any $v \in V$. Assume that (i), (ii), (iii) and (iv) hold for v.

(a) $\varphi'_{P_2}(v) \geq \varphi'_{P_1}(v)$.

Due to Lemma 3, $\varphi'_{P_1}(v) = 0$. Since 0 is the minimum value of φ'_{P_1} and φ'_{P_2}, $\varphi'_{P_2}(v) \geq \varphi'_{P_1}(v)$.

(b) There exists a child v_c of v, $\varphi'_{P_2}(v_c) \leq \varphi'_{P_1}(v_c)$.

Consider two cases:

Case 1: $[E_{P_2} = E_{P_1}]$ Due to Lemma 6, there exist a label $Z \notin label_{P_1}(v)$ and $Z \in label_{P_2}(v)$ and there is a leaf-child v_c of v such that $Z \in label(v_c)$

due to propagation of labels described in the definition of *label*. Since $Z \notin label_{P_1}(v)$, there is no child v_d of v such that $Z \in label_{P_1}(v_d)$; therefore, $Z \notin label_{P_1}(v_c)$. Since v_c is a leaf, then $label_{P_1}(v_c) = \emptyset$. Then by Lemma 1, $\varphi P_1(v_c) = 1$. Since 1 is the maximum value of φP_1 and φP_2, $\varphi'_{P_2}(v_c) \leq \varphi'_{P_1}(v_c)$.

Case 2: $[E_{P_2} \neq E_{P_1}]$ Due to Lemma 7, since edge $(v, v_c) \notin E_{P_1}$, $v_c \notin V_{P_1}$. Then by Lemma 1, $\varphi_{P_1}(v_c) = 1$. Since 1 is the maximum value of φ'_{P_1} and φ'_{P_2}, $\varphi'_{P_2}(v_c) \leq \varphi'_{P_1}(v_c)$.

(c) $(\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c)) - (\varphi'_{P_2}(v) - \varphi'_{P_1}(v)) \geq 0$.

Consider two cases:

Case 1: $[E_{P_2} = E_{P_1}]$ Since (i), (ii) and (iii) hold, then by Lemma 3, $\varphi'_{P_1}(v) = 0$. Let us consider the case when $\forall v \in V$, $(\varphi'_{P_2}(v) - \varphi'_{P_1}(v))$ is maximum. Since (i) and (ii) hold, then $label_{P_2}(v) \neq \emptyset$, $label_{P_2}(sibling_{P_2}(v)) \neq \emptyset$ and

$$\begin{aligned}\varphi'_{P_2}(v) &= minC_{P_1}(v)\\ &= min(maxContr_{P_1}(v), maxContr_{P_1}(sibling_{P_1}(v))).\end{aligned}$$

Since (i), (iii) and (iv) hold, we know that one of v or $sibling_{P_2}(v)$ has at least 2 labels in P_2 and the other one has at least 1 label in P_1. (Note that $Z \in label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v))$.) Since by Lemma 6, $Z \notin label_{P_1}(v)$, $Z \in label_{P_2}(v_c)$, Z is also in $label_{P_2}(v)$; then we know that v has at least 2 labels and $sibling_{P_2}(v)$ has at least 1 label in P_2. Therefore,

$$\begin{aligned}\varphi'_{P_2}(v) &= min(\tfrac{1}{|label_{P_2}(v)|}, \tfrac{1}{|label_{P_2}(sibling_{P_2}(v))|})\\ &= min(\tfrac{1}{2}, 1) = \tfrac{1}{2}.\end{aligned}$$

(Observe that if the number of labels of v or $sibling(v)$ is greater than 2, $\varphi'_{P_2}(v)$ is smaller). Since the maximum value of φ'_{P_2} is $\frac{1}{2}$ and the value of φ'_{P_1} is 0, $\varphi'_{P_2}(v) - \varphi'_{P_1}(v) \leq \frac{1}{2}$.

Let us consider the case when $\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c)$ is minimum. Since $Z \in label_{P_2}(v)$, there should be a leaf $l \in V_{P_2}$ such that $Z \in label(l)$. Let $v_c = l$. Since $Z \notin label_{P_1}(v)$, There is no child v_d of v such that $Z \in label_{P_1}(v_d)$; therefore, $Z \notin label_{P_1}(v_c)$. Since v_c is a leaf, then either $v_c \notin V$ or $label_{P_1}(v_c) = \emptyset$. Then by Lemma 1, $\varphi P_1(v_c) = 1$. Since $Z \notin label_{P_1}(v)$, and $\exists C \in label_{P_1}(v)$; $C \in label_{P_1}(sibling(v_c))$(Because C should be propagated from its child $sibling(v_c)$ to v.). Since $Z \neq C$, and the condition (iii), by Lemma 3, $\varphi_{P_1}(v_c) = 0$. Since the value of $\varphi_{P_1}(v_c)$ is 1 and the value of $\varphi_{P_1}(v_c)$ is 0, $\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c) = 1$. Since $\varphi'_{P_2}(v) - \varphi'_{P_1}(v) \leq \frac{1}{2}$ and $\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c) = 1$, $(\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c)) - (\varphi'_{P_2}(v) - \varphi'_{P_1}(v)) > 0$.

Case 2: $[E_{P_2} \neq E_{P_1}]$ Since (i), (ii) and (iii) hold, then by Lemma3, $\varphi'_{P_1}(v) = 0$. Let us consider the case when $(\varphi'_{P_2}(v) - \varphi'_{P_1}(v))$ is maximum. Due to (i),(ii), and $P_1 \subseteq_l P_2$, $label_{P_2}(v) \neq \emptyset$, $label_{P_2}(sibling_{P_2}(v)) \neq \emptyset$ and

$$\begin{aligned} \varphi'_{P_2}(v) &= minC_{P_1}(v) \\ &= min(maxContr_{P_1}(v), maxContr_{P_1}(sibling_{P_1}(v))). \end{aligned}$$

Due to (i) and $P_1 \subseteq_l P_2$, v has at least one label A in P_2. Due to Lemma 7, v has another label Z in P_2. Due to Lemma 7 and $P_1 \subseteq_l P_2$, Z is also a label of $sibling_{P_2}(v)$. We know that v has at least 2 labels and $sibling_{P_2}(v)$ has at least 1 label in P_2. Therefore,

$$\varphi'_{P_2}(v) = min(\frac{1}{|label_{P_2}(v)|}, \frac{1}{|label_{P_2}(sibling_{P_2}(v))|}) = min(\frac{1}{2}, 1) = \frac{1}{2}.$$

Since the maximum value of φ'_{P_2} is $\frac{1}{2}$ and the value of $\varphi'_{P_1}(v)$ is 0, $\varphi'_{P_2}(v) - \varphi'_{P_1}(v) \leq \frac{1}{2}$.

Let us now consider the case when $\forall v \in V$, $\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c)$ is minimum. Due to Lemma 7, since edge $(v, v_c) \notin E_{P_1}$, $v_c \notin V_{P_1}$. Then by Lemma 1, $\varphi'_{P_1}(v_c) = 1$.

Due to (i), v has at least one label A in V_1. Due to Lemma 7, there exist a label Z in $label_{P_2}(v)$, that is not in $label_{P_1}(v)$; thus $Z \neq A$. Due to Lemma 7, $v_c \notin V_{P_1}$. On the other hand, due to the definition of *label*, there exist a child v_s of v in P_1, such that $A \in label_{P_1}(v_s)$. Since $(v, v_c) \in P_2$, v_s is the sibling of v_c in P_2. So, $sibling_{P_1}(v_c)$ and $sibling_{P_2}(v_c)$ has at least 1 label which is A. So far we know that $label_{P_2}(v_c)$ has at least one label Z, and $label_{P_2}(v_c)$ has at least one label A. The function φ'_{P_2} gets the maximum value for v_c for instance under the following condition: $label_{P_2}(sibling_{P_2}(v_c)) = \{A\}$ and $label_{P_2}(v_c) = \{Z, A\}$. If v_c and $sibling_{P_2}(v_c)$ have more than 2 labels, due to the definition of ς, φ'_{P_2} decreases. Then

$$\begin{aligned}
\varphi'_{P_2}(v_c) = minC_{P_2}(v) & \\
&= min(maxContr_{P_2}(v_c), maxContr_{P_2}(sibling_{P_2}(v_c))) \\
&= min(\tfrac{1}{|label_{P_2}(v_c)|}, \tfrac{1}{|label_{P_2}(sibling_{P_2}(v_c))|}) \\
&= min(\tfrac{1}{2}, 1) = \tfrac{1}{2}.
\end{aligned}$$

Since the minimum value of $\varphi' P_1$ is 1 and the maximum value of $\varphi' P_1$ is $\frac{1}{2}$, $\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c) \geq \frac{1}{2}$.

Since $\varphi'_{P_2}(v) - \varphi'_{P_1}(v) \leq \frac{1}{2}$ and $\varphi P_1(v_c)$ is $\frac{1}{2}$, $\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c) \geq \frac{1}{2}$,

$$(\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c)) - (\varphi'_{P_2}(v) - \varphi'_{P_1}(v)) \geq 0.$$

□

Lemma 9. $UB_\varphi(P_2, V) \leq UB_\varphi(P_1, V)$.

Proof. Consider two cases:

- Case 1:(See Figure 3) Assume that one of the following holds:

  (i) $label_{P_1}(v) = \emptyset$,

  (ii) $label_{P_1}(sibling_{P_1}(v)) = \emptyset$,

  (iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

  (iv) $label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v)) = \emptyset$.

  If (i), by Lemma 1, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$. By definition of UB, $UB_\varphi(P_2, V) \leq UB_\varphi(P_1, V)$.

  If (ii), by Lemma 2, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$. By definition of UB, $UB_\varphi(P_2, V) \leq UB_\varphi(P_1, V)$.

  If neither (i) nor (ii) holds, and (iii) holds, by Lemma 5, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$. By definition of UB_φ, $UB_\varphi(P_2, V) \leq UB_\varphi(P_1, V)$.

  If neither (i) nor (ii) holds, and (iv) holds, by Lemma 4, $\varphi'_{P_1}(v) \geq \varphi'_{P_2}(v)$. By definition of UB_φ, $UB_\varphi(P_2, V) \leq UB_\varphi(P_1, V)$.

- Case 2:(See Figure 4) Assume that all of the following hold:

  (i) $label_{P_1}(v) \neq \emptyset$,

  (ii) $label_{P_1}(sibling_{P_1}(v)) \neq \emptyset$,

(iii) $label_{P_1}(v) \cap label_{P_1}(sibling_{P_1}(v)) = \emptyset$,

(iv) $label_{P_2}(v) \cap label_{P_2}(sibling_{P_2}(v)) \neq \emptyset$.

In other words, although $\varphi'_{P_2}(v) \geq \varphi'_{P_1}(v)$ by Lemma 8(a), there is a child v_c of v such that $\varphi'_{P_2}(v_c) \leq \varphi'_{P_1}(v_c)$ by Lemma 8(b). Moreover, the difference between $(\varphi'_{P_1}(v_c) - \varphi'_{P_2}(v_c))$ is greater than the difference between $(\varphi'_{P_2}(v) - \varphi'_{P_1}(v))$ by Lemma 8(c). Therefore, $\sum_{v \in V} \varphi P2(v) \leq \sum_{v \in V} \varphi P1(v)$; and $UB_\varphi(P_2, V) \leq UB_\varphi(P_1, V)$.

□

Lemma 10. *$weight(P) = UB_\varphi(P, V)$.*

Proof. Since P is a complete phylogeny, all of its vertices and labels of each vertex is complete. Then, for every $v \in V$, by definition of φ', $\varphi'_P(v) = minC_P(v) = \varphi_P(v)$. Therefore, by the definitions of *weight* and UB_φ, $weight(P) = UB_\varphi(P, V)$. □

Proof of Proposition 5. For any partial phylogeny P_0 of P, there exists a sequence $P_0, P_1, ..., P_k = P$ of partial phylogenies such that $P_0 \subseteq_{l_1} P_1 \subseteq_{l_1} ... \subseteq_{l_1} P_k = P$. Since for every i, $P_i{+}1 \subseteq_l P_i$, by Lemma 9, for all partial phylogenies P_i, where $(0 \leq i \leq k)$, $UB_\varphi(P_i, V) \leq UB_\varphi(P_{i+1}, V)$. By Lemma 10, $weight(P_k) = UB_\varphi(P_k, V)$. Then $UB_\varphi(P_0, V) \geq weight(P, V)$. Therefore, UB_φ is admissible.

□

Upper Bound for W4 Let A be the partially constructed phylogeny of P. Let IC denote the set of compatible and informative characters for P, and I_A denote the set of informative and compatible characters for A. Let $maxW$ denote the maximum weight that a character can have (i.e. $maxW = |\{l : l \in L\}|$). Then, we can define the heuristic function with respect to A and a set I of characters as follows:

$$UB_4(A,I) = \sum_{i \in I_A} w(i) + \sum_{i \in IC \setminus I_A} maxW \tag{3.14}$$

With this heuristic function (implemented as a C++ program) and the phylogeny reconstruction program of [12], CLASP-W can compute all correct solutions (i.e., phylogenies whose weight is at least w). In other words, this heuristic function ensures that the following holds for every phylogeny P computed in the end:

$$w \leq weight_4(P) \leq UB_4(A,I). \tag{3.15}$$

This result follows from $weight_4(P) \leq UB_4(A,I)$ (admissibility), and $w \leq UB_P(A,I)$ iff $w \leq weight_4(P)$ (correctness).

Proposition 6. *UB_4 is admissible.*

We need the following lemma to prove Proposition 6.

Lemma 11. *$w(i) \leq maxW$ for every $i \in I$.*

Proof.

$$w(i) = |\{l : l \in L, f(l,i) = g(sibling(l),i)\}|.maxW = |\{l : l \in L\}|.$$

Therefore, $w(i) < maxW$ for every $i \in I$. □

Proof of Proposition 6. We want to show that, $weight_4(P) \leq UB_4(A,I)$.

$$weight_4(P) = \sum_{i \in IC} w(i) = \sum_{i \in I_A} w(i) + \sum_{i \in IC \setminus I_A} w(i)$$

$$UB_4(A,I) = \sum_{i \in I_A} w(i) + \sum_{i \in IC \setminus I_A} maxW$$

Since by Lemma 3.5.2, $w(i) \leq maxW$ for every i,

$$\sum_{i \in I_A} w(i) + \sum_{i \in IC \setminus I_A} w(i) \leq \sum_{i \in I_A} w(i) + \sum_{i \in IC \setminus I_A} maxW.$$

Therefore, $weight_4(P) \leq UB_4(A, I)$.

$\square$

3.6 Phylo-ASP

Phylo-ASP is a tool for analyzing and reconstructing phylogenies with a character-based approach based on the compatibility criterion. It is designed and implemented to solve all the problems given in Subsection 3.3, using the methods described in 3.5, with a divide-and-conquer approach. There is no such phylogenetic system which can help experts to order phylogenies with respect to a weight measure that characterize their plausibiliy with respect to some domain-specific information.

There are four modules in Phylo-ASP : Phylo-Analyze-ASP, Phylo-Reconstruct-ASP, PhyloCompare-ASP and Phylo-ReconstructN-ASP. PhyloCompare-ASP and Phylo-ReconstructN-ASP are studied in [15] and [29]. The other two modules are introduced in this thesis.

3.6.1 Phylo-Analyze-ASP

Phylo-Analyze-ASP is for analyzing input (leaf-labeling function) and output (phylogenies). It analyzes the leaf-labeling function and finds uninformative and non-unique characters. Additionally, it finds the incompatible characters with a given phylogeny. The input and the output of this module is as follows:

Input: Phylo-Analyze-ASP takes two kinds of input:

- Character states of taxonomic units for every character as a matrix
- A phylogenetic tree in Newick format (Optional)

Output: Phylo-Analyze-ASP outputs the uninformative and the non-unique characters. If a phylogenetic tree in Newick format is given as an input, then the program additionally outputs the incompatible characters for that tree.

3.6.2 Phylo-Reconstruct-ASP

Phylo-Reconstruct-ASP is for reconstructing (weighted) phylogenies for a given input. It can solve different decision and optimization problems, which are stated in Subsection 3.3.

Moreover, it can process domain specific information such as how to group taxonomic units, or sibling information and considers this information while reconstructing phylogenies. In addition, if domain specific information about how to group taxonomic units exists, then Phylo-Reconstruct-ASP reconstructs phylogenies with a divide-and-conquer approach. This approach is very important, because computing a phylogeny with a large number of taxanomic units and characters is very time consuming and yet sometimes it is not possible to reconstruct phylogenies for large datasets with existing methods.

In the divide-and-conquer approach, the phylogeny reconstruction problem is divided into smaller phylogeny reconstruction problems. For example, consider a set of species, with a large number of characters (Figure 5a). Experts usually provide domain-specific information about how to group these species based on biological or morphological data (Figure 5b). With this grouping information, first we build a phylogeny for all groups (Figure 5c). Then with the labels of the groups extracted from this phylogeny, we build phylogenies for each group (Figure 5d).

Finally, we combine the main phylogeny for all groups and a phylogeny for each group, and obtain a complete phylogeny (Figure 5e).

Note that, another alternative to divide-and-conquer approach is to reconstruct phylogenies for each group first, and then with the labeling information of the roots of these phylogenies, to reconstruct a main phylogeny. However, with this alternative, since labeling of the groups are picked before the computation of a main phylogeny, the main phylogeny may not be optimal (i.e., the main phylogeny may have a minimum number of incompatible characters with a different possible labeling). For example, consider the phylogeny in Figure 6 and suppose that the species A and B are grouped together, and C, D, and E are grouped together. With this approach, a phylogeny for the species A and B, and a phylogeny for the species C, D, and E are computed in the first step. The root of the former can be labeled with 0 or 1 with the same priority with respect to the compatibility criterion, so that one of them will be picked and propagated to the mail phylogeny, as the labeling of the group. If it is labeled with 0, then this character will be incompatible in the main phylogeny. Therefore, the minimum number of incompatible characters for the complete phylogeny will be 1, even though it is possible to label the internal vertices in a way to make it 0.

The input and the output of this module is as follows:

Input: PHYLO-RECONSTRUCT-ASP takes six kinds of input:

- Character states of taxonomic units for every character
- Strict grouping information, which specifies how to group species exactly (Optional)
- Preferred grouping information, which specifies how to group species preferably (Optional)

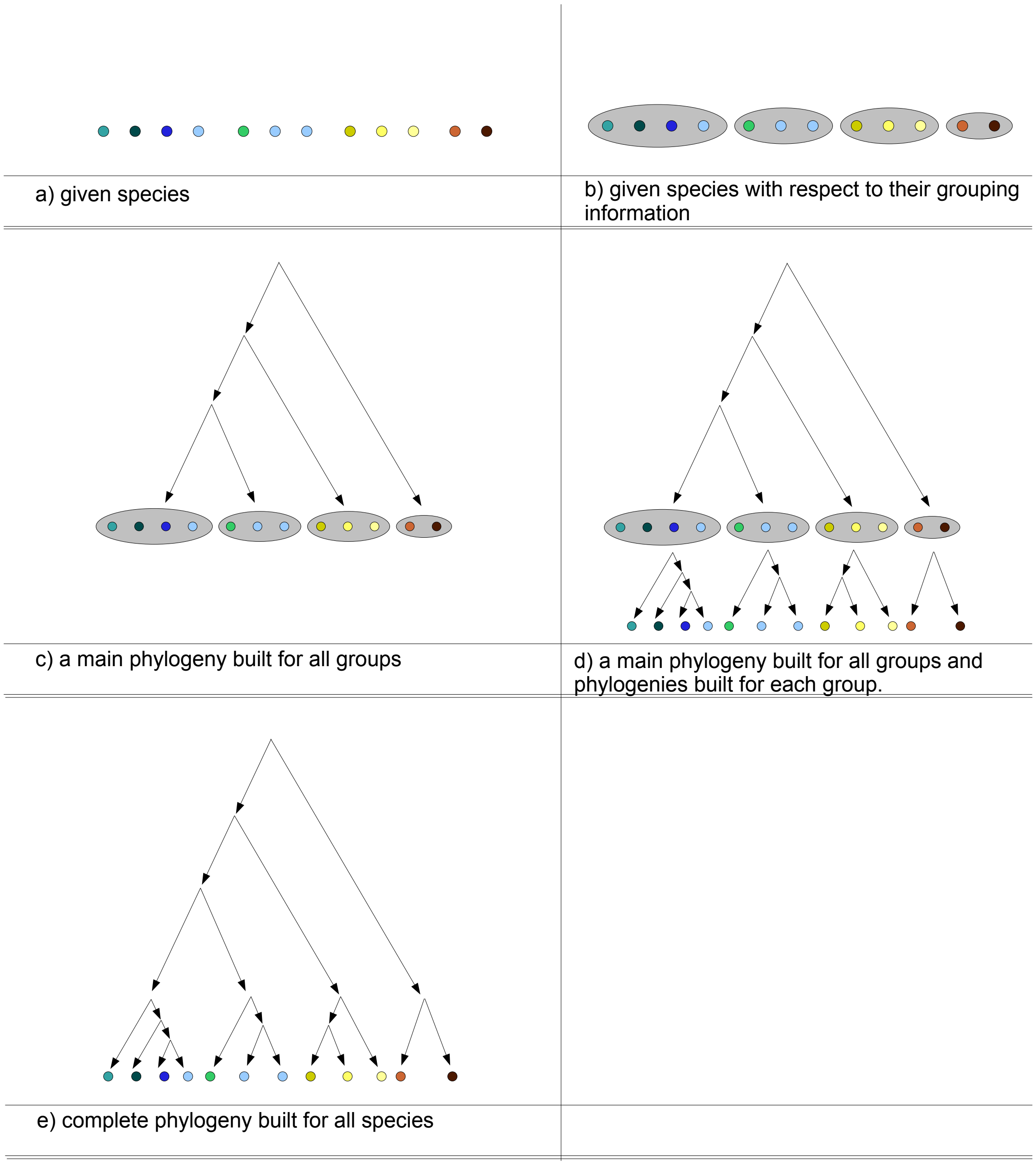

a) given species
b) given species with respect to their grouping information
c) a main phylogeny built for all groups
d) a main phylogeny built for all groups and phylogenies built for each group.
e) complete phylogeny built for all species

Figure 5: The divide-and-conquer technique used in PhyloReconstruct-ASP

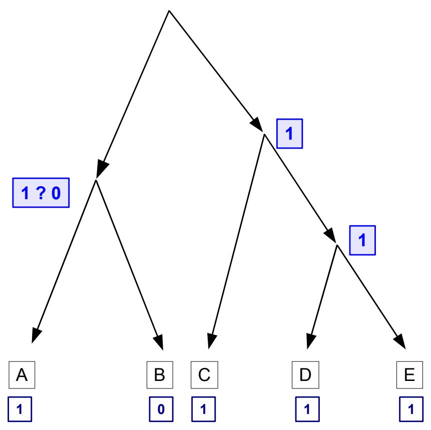


Figure 6: A phylogeny

- Weight measures (Optional)
- A nonnegative integer n, which is the maximum number of incompatible characters that a phylogeny can have (Optional)
- A nonnegative integer w, which is the minimum weight that a phylogeny can have (Optional)

Output: PHYLO-RECONSTRUCT-ASP outputs the computed phylogenetic trees in Newick format as well as the uninformative, non-unique and incompatible characters with the tree.

The input, other than character states, are optional. Depending on the given input, PHYLO-RECONSTRUCT-ASP can solve different kinds of problems:

- If n is given as an input, PHYLO-RECONSTRUCT-ASP solves n-CP problem: It reconstructs all (or a desired number of) phylogenies with at most n incompatible characters with respect to the character states.

- If w is given as an input, Phylo-Reconstruct-ASP solves k-WCP problem: It reconstructs all (or a desired number of) phylogenies with at least w weight with respect to the character states and the desired weight measure.
- If both n and w are given as an input, Phylo-Reconstruct-ASP solves wn-WCP problem: It reconstructs all (or a desired number of) phylogenies with at most n incompatible characters and at least w weight with respect to the character states and the desired weight measure.
- If both n and w are not given as an input, Phylo-Reconstruct-ASP solves either MCP or MWCP, depending on the existence of the weight measure as input.

Overall Structure: The overall structure of Phylo-ASP can be seen in Figure 7:

1 **Extra Preprocessing:** In this step, the aim is to reduce the dataset by finding some incompatible characters before the phylogeny reconstruction.

2 **Preprocessing:** In this step, the aim is again to reduce the dataset by finding uninformative and nonunique characters before the phylogeny reconstruction.

3 **Reconstruction:** In this step,

  3.1 Phylo-Reconstruct-ASP first computes phylogenetic trees for all groups with respect to given grouping information.

  3.2 Then the labels of the groups are extracted from these phylogenies (constructed in Step 3.1).

  3.3 After that, with the labels of the groups extracted, Phylo-Reconstruct-ASP computes phylogenetic trees for each group.

4 **Combination:** In this step, PHYLO-RECONSTRUCT-ASP combines the phylogenies computed in Step 3.1 with the phylogenies computed in Step 3.3, outputs the combined phylogenies.

Extra Preprocessing: In order to reduce the dataset, we identify the characters that are incompatible for any possible phylogeny for that dataset. We have two different methods to identify such characters:

E1 We find some incompatible characters by enumerating all phylogenies for each group. We say that a character is incompatible for any phylogeny, if it is incompatible for all phylogenies computed for a group. Such characters are guaranteed to be incompatible in any phylogeny (Proposition 7).

E2 We find some incompatible characters for a phylogeny (V, E, L, I, S, f) by analyzing the input matrix. For each set G of taxonomic units of P, we define $S_{Gi} = \{s \in S | s \in f(v,i), v \in G\}$. Then a character $i \in I$ is incompatible if the following holds:

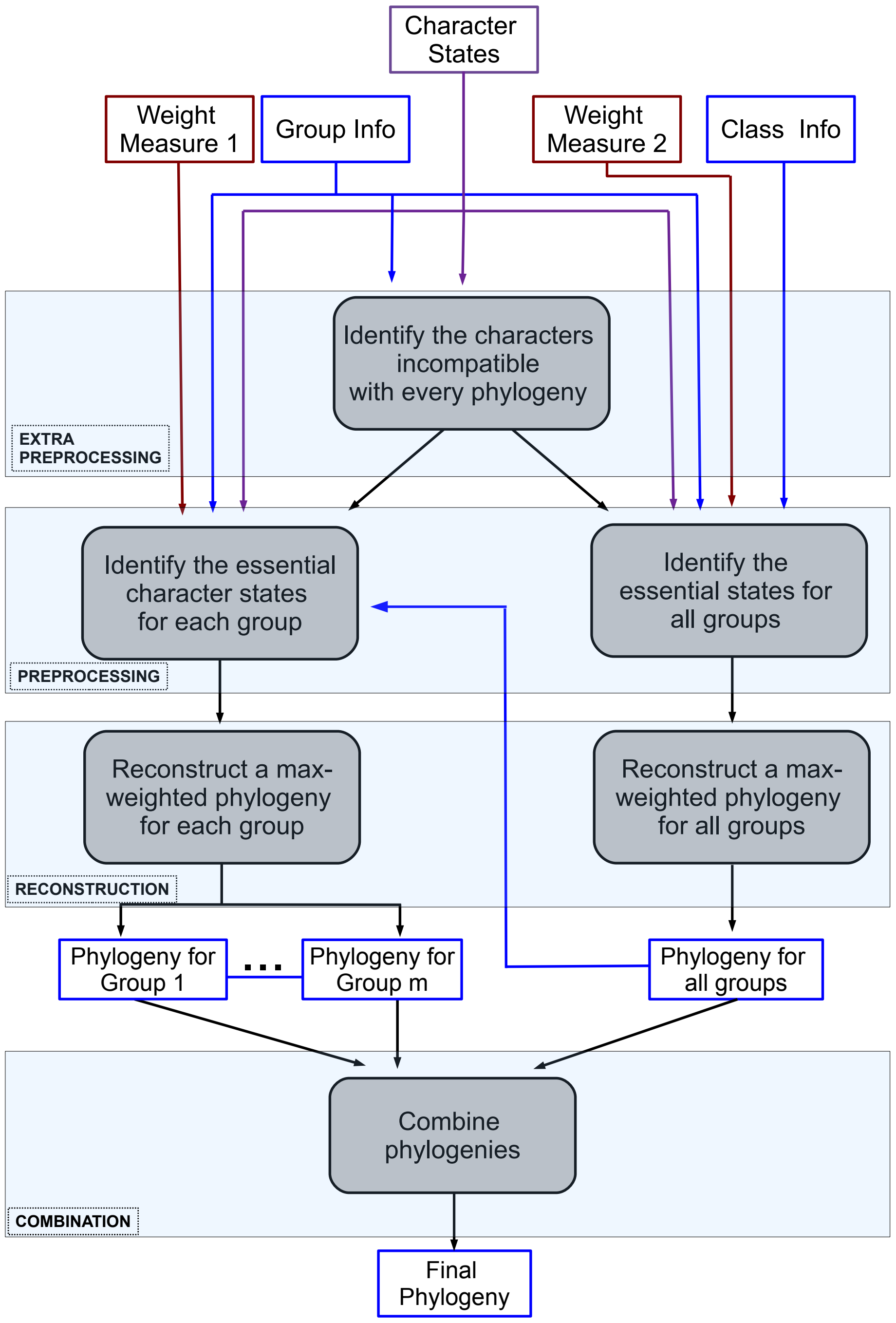
For a set G of taxonomic units and a set G' of taxonomic units,

i) $|S_{Gi} \cap S_{G'i}| \geq 2$.

ii) For at least two states $s, s' \in S_{Gi} \cap S_{G'i}$:

* $\exists\, v \in G$ s.t. $s \in f(v,i)$ and $\nexists t \neq s$ s.t. $t \in f(v,i)$
* $\exists\, v \in G'$ s.t. $s \in f(v,i)$ and $\nexists t \neq s$ s.t. $t \in f(v,i)$
* $\exists\, v \in G$ s.t. $s' \in f(v,i)$ and $\nexists t \neq s'$ s.t. $t \in f(v,i)$
* $\exists\, v \in G'$ s.t. $s' \in f(v,i)$ and $\nexists t \neq s'$ s.t. $t \in f(v,i)$

Note that, when f is a function, any character satisfying (i) is guaranteed to be incompatible for any phylogeny (Proposition 3.6.2). When f is a relation, any



Character States
Weight Measure 1
Group Info
Weight Measure 2
Class Info
Identify the characters incompatible with every phylogeny
EXTRA PREPROCESSING
Identify the essential character states for each group
Identify the essential states for all groups
PREPROCESSING
Reconstruct a max-weighted phylogeny for each group
Reconstruct a max-weighted phylogeny for all groups
RECONSTRUCTION
Phylogeny for Group 1
...
Phylogeny for Group m
Phylogeny for all groups
Combine phylogenies
COMBINATION
Final Phylogeny

Figure 7: The Overall System Architecture of PhyloReconstruct-ASP

character satisfying (i) and (ii) is guaranteed to be incompatible for any phylogeny. (Proposition 3.6.2)

Proposition 7. *Let $G \subseteq L$ be a set of taxonomic units. Let P be a phylogeny (V, E, L, I, S, f), that contains a phylogeny for G. Let i be a character in I. If i is incompatible with respect to every possible phylogeny for G, then i is incompatible with respect to P.*

Proof. Let $G \subseteq L$ be a set of taxonomic units. Let P be a phylogeny (V, E, L, I, S, f), that contains a phylogeny P_G for G. Let i be a character in I.

Take any $i \in I$. Assume that i is incompatible with respect to every possible phylogeny for G. We want to show i is incompatible with respect to P.

Since i is incompatible with a phylogeny for G, then there exists no function $g_G : G \times i \rightarrow S$ that satisfies the following conditions:

[G1] for every leaf $v \in G$, $g(v, i) = f(v, i)$

[G2] for every $s \in S$, if the set $V_{is}^G = \{x \in G : g_p(x, i) = s\}$ is non empty, then the digraph $\langle G, E\rangle$ has a subgraph with the set V_{is}^G of vertices that is a rooted tree.

Suppose i is compatible with P.Since i is compatible with P, there exists a function $g_P : V \times i \rightarrow S$ that satisfies the following conditions:

[P1] for every leaf $v \in V$, $g(v, i) = f(v, i)$

[P2] for every $s \in S$, if the set $V_{is}^P = \{x \in V : g_P(x, i) = s\}$ is non empty, then the digraph $\langle V, E\rangle$ has a subgraph with the set V_{is}^P of vertices that is a rooted tree.

From the definition of g_P above, we can extract a function $g_G : G \times i \rightarrow S$ such that the following holds:

- Since $P_G \subseteq P$, then $g_G(G, i) = f(v, i) = g_P(v, i)$. Therefore, g_G satisfies [G1].
- If the set $V^G_{is} = \{x \in G : g_p(x, i) = s\}$ is non empty, then V^P_{is} is non empty. Then there exists a subgraph of P, with the set V^P_{is} of vertices, that is a rooted tree. Since $V^G_{is} \subseteq V^P_{is}$, then P_G also has such a tree. Therefore, g_G satisfies [G2].

Since g_G satisfies [G1] and [G2], i is compatible with G. This contradicts with i is compatible with a phylogeny for P.

□

Proposition 8. *Let $G \in L$ and $G' \in L$ be a set of taxonomic units. Let P be a phylogeny (V, E, L, I, S, f) that contains a phylogeny for G and a phylogeny for G'. Let i be a character in I and let f be a function $V \times I \to S$. For G and G', if $|S_{Gi} \cap S_{G'i}| \geq 2$, then a character $i \in I$ is incompatible.*

Proof. Let $G \in V$ and $G' \in V$ be a set of taxonomic units. Let P be a phylogeny (V, E, L, I, S, f) that contains a phylogeny P_G for G and a phylogeny P'_G for G'. Let i be a character in I and let f be a function $V \times I \to$.

For any character $i \in I$, assume $|S_{Gi} \cap S_{G'i}| \geq 2$. We want to show that i is incompatible for P.

Take any i in I. Due to $|S_{Gi} \cap S_{G'i}| \geq 2$, there should be at least 2 different states s, t such that, $s, t \in S_{Gi}$ and $s, t \in S_{G'i}$. Note that since f_{P_G} is defined from $V_{P_G} \times I_{P_G}$ to S_{P_G}, if there exists two different states in S_{Gi} then there should be at least 2 different leaves, $l_{1G}, l_{2G} \in G$. Similarly, since $f_{P'_G}$ is defined from $V_{P'_G} \times I_{P'_G}$ to $S_{P'_G}$, if there exists two different states in $S_{G'i}$ then there should be at least 2 different leaves, $l_{1G'}, l_{2G'} \in G$. Suppose that for character i, the leaves $l_{1G} \in G$ and $l_{1G'} \in G'$ are labeled with s and $l_{2G} \in G$ and $l_{2G'} \in G'$ are labeled with t in P. Due to the definition of compatibility, if i is compatible, then

(a) for the state s, there is a subgraph SG_s of P, with the set $V_is = \{v \in V : g(v, i) = s\}$ of vertices that is a rooted tree.

(b) for the state t, there is a subgraph SG_t of P, with the set $V_it = \{v \in V : g(v, i) = t\}$ of vertices that is a rooted tree.

Note that, if i is compatible, then

- due to (a), the root r_G of the phylogeny for G and the root r'_G of the phylogeny for G' should be in SG_s, in order to connect l_{1G} and $l_{1G'}$.
- due to (b), the root r_G of the phylogeny for G and the root r'_G of the phylogeny for G' should be in SG_t, in order to connect l_{2G} and $l_{2G'}$.

Since all the vertices in SG_s should be labeled with s, and all the vertices in SG_t should be labeled with t and since $s \neq t$, [a] and [b] can not hold. Therefore, i is incompatible. $\square$

Proposition 9. *Let $G \in V$ and $G' \in V$ be a set of taxonomic units. Let P be a phylogeny (V, E, L, I, S, f) that contains a phylogeny for G and a phylogeny for G'. Let i be a character in I and let f be a relation $V \times I \rightarrow S$. Then a character $i \in I$ is incompatible if the following holds:*

For G and G',

i) $|S_{Gi} \cap S_{G'i}| \geq 2$.

ii) For at least two states $s, s' \in S_{Gi} \cap S_{G'i}$:

- $\exists\ v \in G$ *s.t.* $s \in f(v, i)$ *and* $\nexists t \neq s$ *s.t.* $t \in f(v, i)$
- $\exists\ v \in G'$ *s.t.* $s \in f(v, i)$ *and* $\nexists t \neq s$ *s.t.* $t \in f(v, i)$
- $\exists\ v \in G$ *s.t.* $s' \in f(v, i)$ *and* $\nexists t \neq s'$ *s.t.* $t \in f(v, i)$

– $\exists\ v \in G'$ *s.t.* $s' \in f(v,i)$ *and* $\nexists t \neq s'$ *s.t.* $t \in f(v,i)$

Proof. Let $G \in P$ and $G' \in P$ be a set of taxonomic units. Let P be a phylogeny (V, E, L, I, S, f) that contains a phylogeny for G and a phylogeny for G'. Let i be a character in I and let f be a relation $V \times I \to S$.

For any character $i \in I$, assume (i) and (ii). We want to show that i is incompatible for P.

Due to condition (ii), for the character i to be incompatible, for at least two states s and t, there should exist two vertices in G, such that one of vertices should map only to one state s, and the other vertex should map only to another state, t. Similarly, there should exist two vertices in G', such that one of them should map only to one state s, and the other one should map only to another state, t. Then, the proof of this claim is very similar to Proposition 3.6.2: Take any i in I. Due to $|S_{Gi} \cap S_{G'i}| \geq 2$, there should be at least 2 different states s, t such that, $s, t \in S_{Gi}$ and $s, t \in S_{G'i}$. Note that since f_{P_G} is defined from $V_{P_G} \times I_{P_G}$ to S_{P_G}, if there exists two different states in S_{Gi} then there should be at least 2 different leaves, $l_{1G}, l_{2G} \in G$. Similarly, since $f_{P'_G}$ is defined from $V_{P'_G} \times I_{P'_G}$ to $S_{P'_G}$, if there exists two different states in $S_{G'i}$ then there should be at least 2 different leaves, $l_{1G'}, l_{2G'} \in G$. Suppose that for character i, the leaves $l_{1G} \in G$ and $l_{1G'} \in G'$ are labeled with s and $l_{2G} \in G$ and $l_{2G'} \in G'$ are labeled with t in P. Due to the definition of compatibility, if i is compatible, then

(a) for the state s, there is a subgraph SG_s of P, with the set $V_is = \{v \in V : g(v,i) = s\}$ of vertices that is a rooted tree.

(b) for the state t, there is a subgraph SG_t of P, with the set $V_it = \{v \in V : g(v,i) = t\}$ of vertices that is a rooted tree.

Note that, if i is compatible, then

- due to (a), the root r_G of the phylogeny for G and the root r'_G of the phylogeny for G' should be in SG_s, in order to connect l_{1G} and $l_{1G'}$.
- due to (b), the root r_G of the phylogeny for G and the root r'_G of the phylogeny for G' should be in SG_t, in order to connect l_{2G} and $l_{2G'}$.

Since all the vertices in SG_s should be labeled with s, and all the vertices in SG_t should be labeled with t and since $s \neq t$, [a] and [b] can not hold. Therefore, i is incompatible. $\square$

Preprocessing: In order to reduce the dataset, we identify characters that are uninformative and nonunique for any possible phylogeny for that dataset as in [12].

Phylogeny Reconstruction: In phylogeny reconstruction, PHYLO-RECONSTRUCT-ASP first computes main phylogenetic trees for all groups with respect to given grouping information (Algorithm 11):

- First, we minimize the number n of incompatible characters.
- Then, we maximize the weight w of the phylogeny with respect to n.
- Then we compute all phylogenies with n and w values.

Then the labels of the groups are extracted from these phylogenies and with the labels of the groups extracted, PHYLO-RECONSTRUCT-ASP computes phylogenetic trees for each group:

- First, we minimize the number n_g of incompatible characters.
- Then, we maximize the weight w_g of the phylogeny with respect to n_g.
- Then we compute all phylogenies with n_g and w_g values.

If the input is a a leaf-labeling relation rather than a leaf-labeling function, PHYLO-RECONSTRUCT-ASP first minimizes the number of essential states, in order to reduce the number of incompatible characters of the complete phylogeny and reconstruct phylogenies as explained before, with the minimum number of essential states.

3.7 Experimental Results

We have applied our methods and tested our system with two different datasets: Indo-European languages and *Quercus* species. For both datasets, in addition to solving an optimization problem to find phylogenies with the minimum number of incompatible characters and maximum weight, we are interested in solving a decision problem to find phylogenies with a small number of incompatible characters and a large weight.

We are interested in decision problems because some computed phylogenies can be identified as plausible by the experts even though they do not have minimum number of incompatible characters or maximum weight. For example, in [12], for Indo-European languages, some phylogenies computed with 17 or 18 incompatible characters are identified as plausible, even though the minimum number of incompatible characters for the dataset is 16. In addition, in [14], the phylogenies with the weights over 45 are identified as plausible, even though the maximum weight of the computed phylogenies is 65.

Based on this motivation, we can decide for the thresholds as follows: first we compute a phylogeny with the minimum number of compatible characters (since our approach to phylogenetics is based on the compatibility criterion) by increasing the number of incompatible characters one by one starting from 0 until a phylogeny is computed.

Algorithm 10 ReconstructMaxWeightedPhylogenies

```
Require: a leaf-labeling function $f : L \times I \to S$ or a leaf-labeling relation $f : L \times I \times S$, Grouping information GroupInfo, a weight measure WeightMeasure that maps a phylogeny to a number.
Ensure: A set $P_M$ of main phylogenies, a set $P_C$ of complete phylogenies, a set $P_G$ of phylogenies for each group, the minimum number $n_M$ (resp. $n_C$)of incompatible characters of the phylogenies in $P_M$ (resp. $P_C$), the maximum weight $w_M$ (resp. $w_C$) of the phylogenies in $P_M$ (resp. $P_C$).
  {find the set $I_M$ of characters which are incompatible with every main phylogeny.}
  $I_M \leftarrow$ ExtraPreprocess($f$,GroupInfo);
  {find the set $I_M$ of informative characters and the list $S_M$ of (informative character-essential state) pairs for all groups; the set $I_G$ of informative characters and the list $S_G$ of (informative character-essential state) pairs for each group}
  $L$, $I_M$, $S_M$, $I_G$, $S_G \leftarrow$ Preprocess($f$, GroupInfo);
  {reconstruct the set $P_M$ of max-weighted main phylogenies, find the minimum number $n_M$ of incompatible characters, the maximum weight $w_M$ for the main phylogenies, the labeling $g_M$ of vertices and the list $IN_M$ of incompatible characters of the phylogenies in $P_M$}
  $n_M$, $w_M$, $P_M$, $g_M$, $IN_M \leftarrow$ ReconstructMainPhylogenies($L$, $I_M$, $S_M$, WeightMeasure);
  {propagate the labels of the leaves of the main phylogeny to the to-be-reconstructed phylogenies for each group}
  $S_G \leftarrow$ ExtractGroupLabels($L$, $P_M$, $g_M$, $S_G$)
  {reconstruct all max-weighted phylogenies for each group}
  $P_G \leftarrow$ ReconstructPhylogeniesForEachGroup($L$, $S_G$, $I_G$, GroupInfo, WeightMeasure);
  {combine the main phylogenies with the phylogenies for each group}
  $P_C$, $n_C$, $w_C \leftarrow$ CombinePhylogenies($L$, $P_M$, $P_G$, GroupInfo, $I$, $S$ )
  return $P_M$, $P_G$, $P_C$, $n_M$, $w_M$, $n_C$, $w_C$
```

Algorithm 11 RECONSTRUCTMAINPHYLOGENIES

```
Require: A set $L$ of leaves, a set $I_M$ of characters, a list $S_M$ of character-
    state pairs, a weight measure WEIGHTMEASURE that maps a phylogeny
    to a number.
Ensure: A list $P_M$ of main phylogenies, a set $I_{IN}$ of incompatible characters.
  $min\_n$ ← FINDMINIMUMNUMBEROFINCOMPATIBLECHARACTERS($L$,
  $I_M$, $S_M$);
  $max\_w$ ← FINDMAXIMUMWEIGHT($L$, $I_M$, $S_M$, WEIGHTMEASURE,
  $min\_n$ );
  $j := 0$; {computed phylogeny count }
  $previousPhylogenies := \emptyset$; {current reconstructed phylogenies which are
  given as constraints for the next computation }
  repeat
    $P_M[j]$, $g_M$[j], $I_{IN}[j]$ ← RECONSTRUCTPHYLOGENY($L$, $I_M$, $S_M$,
    WEIGHTMEASURE, $min\_n$, $max\_w$, $previousPhylogenies$);
    $previousPhylogenies := previousPhylogenies \cup \{P_M[j]\}$;
    $j++$;
  until $P_M[j] = \emptyset$;
  return $min\_n$, $max\_w$, $P_M$, $g_M$, $I_{IN}$
```

Algorithm 12 ReconstructPhylogeniesForEachGroup

```
Require: A set $L$ of leaves, a list $S_G$ where $S_G$[j] is a set of character-state pairs for each group, a set $I_G$ where $I_G$[i] is a set of informative characters for each group, GroupInfo, WeightMeasure.
Ensure: A list $P_G$ of phylogenies where $P_G[j]$ is a set of phylogenies for each group.
  for each group $j$ in GroupInfo do
    $min\_n \leftarrow$ FindMinimumNumberOfInCompatibleCharacters($L$, $I_G$, $S_G[j]$);
    $max\_w \leftarrow$ FindMaximumWeight($L$, $I_G$, $S_G[j]$, WeightMeasure, $min\_n$ );
    $previousPhylogenies := \emptyset$ ; {current reconstructed phylogenies which are given as constraints for the next computation }
    repeat
      $P_G[j] \leftarrow$ ReconstructPhylogeny($L$, $I_G$, $S_G[j]$, WeightMeasure, $min\_n$, $max\_w$, $previousPhylogenies$);
      $previousPhylogenies := previousPhylogenies \cup \{P_G[j]\}$;
      $j++$;
    until $P_G[j] = \emptyset$
  end for
  return $P_G$
```

Algorithm 13 CombinePhylogenies: Combine one phylogeny for each main phylogeny

```
Require: The set $L$ of leaves, the set $P_M$ of main phylogenies, the list $P_G$ of
  phylogenies where $P_G[j]$ is a set of phylogenies for each group, GroupInfo,
  a set $I$ of characters, a list $S$ of (informative character-essential state)
  pairs, a weight measure WeightMeasure that maps every phylogeny to
  a number.
Ensure: The set of combined phylogenies $P_C$, the minimum number $min\_n$
  of incompatible characters of $P_C$ and the maximum weight $max\_w$ of $P_C$.
  $P_C := \emptyset$;
  $k := 0$; { Main phylogeny counter}
  for each phylogeny $p$ in $P_M$ do
    $P_C[k] := P_C[k] \cup \{p\}$;
    for each group $j$ in GroupInfo do
      $P_C := P_C \cup \{P_G[j][0]\}$;
    end for
    $k++$;
    $min\_n \leftarrow$ FindMinimumNumberOfInCompatibleCharacters($L$,
    $P_C$, $I$, $S$ );
    $max\_w \leftarrow$ FindMaximumWeight($L$, $P_C$, $I$, $S$, WeightMeasure,
    $min\_n$ );
    $P_C := \emptyset$;
  end for
  return $P_C$, $max\_w$, $min\_n$
```

Algorithm 14 FindMinimumNumberOfIncompatibleCharacters

```
Require: The set $L$ of leaves, the set $I$ of characters, a list $S$ of (informative
  character-essential state) pairs.
Ensure: The minimum number of incompatible characters $min\_n$.
  $min\_n := 0$;
  $p := \emptyset$;
  while $p$ is empty do
    $p$ = ReconstructPhylogeny($L$, $I$, $S$, min_n);
    $min\_n++$;
  end while
  return $min\_n - 1$
```

Algorithm 15 FindMaximumWeight

```
Require: The set $L$ of leaves, the set $I$ of characters,the list $S$ of (informative character-essential state) pairs, WeightMeasure, the number of incompatible characters $n$.
Ensure: The maximum weight $max\_w$.
  $max\_w := 0$;
  $p := \emptyset$;
  repeat
    $p$ ← ReconstructPhylogeny($L$, $I$, $S$, WeightMeasure, $n$, $max\_w$);
    $max\_w + +$;
  until $p$ is empty
  return $max\_w - 1$
```

Algorithm 16 Preprocess

```
Require: a leaf-labeling function $f : L \times I \to S$ or a leaf-labeling relation $f : L \times I \times S$, GroupInfo.
Ensure: The set of leaves $L$, the set $I_M$ of informative characters for the main phylogeny, the list $S_M$ of (informative character-essential state) pairs for the main phylogeny, the list $I_G$ of informative characters where $I_G[i]$ is the set of informative characters for each group, the list $S_G$ where $S_G[i]$ is the (informative character-essential state) pairs for each group.
  $L$, $I$, $S$ ←ExtractInformation($f$);
  $I_M$, $S_M$ ← FindInformativeCharactersAndEssentialStates($L$, $I$, $S$);
  for each group $i$ in GroupInfo do
    $I_G[i]$, $S_G[i]$ ← FindInformativeCharactersAndEssentialStates($L_i$, $I$, $S_i$);
  end for
  return $L$, $I_M$, $S_M$, $I_G$, $S_G$
```

Algorithm 17 FINDINFORMATIVECHARACTERSANDESSENTIALSTATES

```
Require: The set $L$ of leaves, the set $I$ of characters, the list $S$ of (informative character-essential state) pairs.
Ensure: The set $I_I$ of informative characters, the list $S_E$ of (informative character-essential state) pairs.
  $I_I := \emptyset$;
  $S_E := \emptyset$;
  for each character $i$ in $I$ do
    $intersectionOfStates := \emptyset$;
    for each leaf $l$ in $L$ do
      for each leaf $u$ in $L$ s.t. $u \neq l$ do
        if $g(l,i) = g(u,i) = s \in S$ then
          $intersectionOfStates := intersectionOfStates \cup \{s\}$;
        end if
      end for
    end for
    if $|intersectionOfStates| \geq 2$ then
      $I_I := I_I \cup \{i\}$;
      $S_E := S_E \cup intersectionOfStates$;
    end if
  end for
  return $I_I$, $S_E$
```

Algorithm 18 EXTRAPREPROCESS

```
Require: a leaf-labeling function $f : L \times I \to S$ or a leaf-labeling relation $f : L \times I \times S$, GROUPINFO.
Ensure: The set $I_C$ of characters which can not be identified as incompatible.
  $I_I1 \leftarrow$ FINDINCOMPATIBLECHARACTERSBYCHECKINGCOMMONCHARACTERS($f$, GROUPINFO);
  $I_I2 \leftarrow$ FINDINCOMPATIBLECHARACTERSBYANALYZINGSTATES($f$, GROUPINFO);
  $I_C := I \setminus (I_I1 \cup I_I2)$;
  return $I_C$
```

Algorithm 19 FINDINCOMPATIBLECHARACTERSBYCHECKINGCOMMON-CHARACTERS

```
Require: a leaf-labeling function $f : L \times I \rightarrow S$ or a leaf-labeling relation
  $f : L \times I \times S$, GROUPINFO.
Ensure: The set $I_I$ of characters which are incompatible for every possible
  phylogeny.
  $IntersectionOfCharacters := I$;
  for each group $g$ in GROUPINFO do
    $I_{IG} \leftarrow$ RECONSTRUCTPHYLOGENY($L$, $I$, $S$);
    $IntersectionOfCharacters := IntersectionOfCharacters \cap I_{IG}$;
  end for
  return $IntersectionOfCharacters$
```

Since there may be plausible phylogenies “close” to the optimal one in terms of the number of incompatible characters, we also compute phylogenies with a “small” number of incompatible characters by further increasing the value of c by some units depending on the number of phylogenies computed so far. Similarly, we can decide for the value of the threshold for weight.

All experiments are done on a workstation with two 1.60 GHz Intel Xeon E5310 Quad-Core Processor,and 16 GB RAM, running Centos 64bit(Version 5.3); in these experiments LPARSE v.1.1.2 and GRINGO v.2.0.3 is used as grounders and CLASP v.1.3.1 is used as the answer set solver.

3.7.1 Indo-European Languages

The first data set we have used in our experiments is the Indo-European languages, prepared by Don Ringe and Ann Taylor [69]. The dataset consists of 282 characters and each character is mapped to 1 to 24 states. We also have some domain-specific information on the simple groupings of these languages; the languages can be grouped into 8 groups which are presented in Table 2.

We have constructed weighted phylogenies with PHYLO-ASP with a divide-

Algorithm 20 FindIncompatibleCharactersByAnalyzingStates

```
Require: a leaf-labeling function $f : L \times I \rightarrow S$ or a leaf-labeling relation
  $f : L \times I \times S$, a vertex-labeling function $g : V \times I \rightarrow S$GroupInfo.
Ensure: The set $I_I$ of characters which are incompatible for every possible
  phylogeny.
  $I_I := \emptyset$;
  for each character $i$ in $I$ do
    for each group $r$ in GroupInfo do
      for each group $h$ in GroupInfo s.t. $r \neq h$ do
        $statesOfFirstGroup := \emptyset$;
        $statesOfSecondGroup := \emptyset$;
        for each element $t$ of group $r$ in GroupInfo do
          for each element $u$ of group $h$ in GroupInfo do
            $statesOfFirstGroup := statesOfFirstGroup \cup g(t, i)$;
            $statesOfSecondGroup := statesOfSecondGroup \cup g(u, i)$;
          end for
        end for
        $intersectionSet := statesOfFirstGroup \cap statesOfSecondGroup$;
        $uniqueIntersection := \emptyset$;
        for each state $s$ in $intersectionSet$ do
          for each element $t$ of group $r$ in GroupInfo do
            for each element $u$ of group $h$ in GroupInfo do
              if $s \in g(t, i)$ AND $s \in g(u, i)$ then
                for all states $t$ in $intersectionSet$ s.t. $t \neq s$ do
                  if $t \notin g(t, i)$ AND $t \notin g(u, i)$ then
                    $uniqueIntersection := uniqueIntersection \cup \{s\}$;
                  end if
                end for
              end if
            end for
          end for
          if $|uniqueIntersection| \geq 2$ then
            $I_I := I_I \cup \{i\}$;
          end if
        end for
      end for
    end for
  end for
  return $I_I$
```

Table 2: Eight Indo-European language groups

Abbreviation	Language groups	Languages
AN	*proto-Anatolian*	Hittite, Luvian, Lycian
TO	*proto-Tocharian*	Tocharian A, Tocharian B
IC	*proto-Italo-Celtic*	Oscan, Umbrian, Latin, Old Irish, Welsh
GE	*proto-Germanic*	Old English, Old High German, Old Norse, Gothic
GA	*proto-Greco-Armenian*	Ancient Greek, Classical Armenian
BS	*proto-Balto-Slavic*	Lithuanian, Latvian, Old Prussian, Old Church Slavonic
IIR	*proto-Indo-Iranian*	Old Persian, Avestan, Vedic
AL	*Albanian*	Albanian

and-conquer approach. The result of our experiments can be summarized as follows:

- We have reconstructed main phylogenies for all groups using the weight measure W4 and found max-weighted phylogenies with 0 incompatible characters and a maximum weight of 42. We have also computed w-weighted phylogenies whose weight is at least 32. All main phylogenies found for all groups are identified as plausible by experts. Results are presented in Table 3.

- We have reconstructed phylogenies for each group using the weight measure W4 and with respect to a main phylogeny. For example, in Table 7, the phylogenies are computed with respect to the Phylogeny 7 in Table 3: we consider the labels of roots of these phylogenies as same as the labels of these vertices in Phylogeny 7. The maximum weights and the minimum number of incompatible characters of each phylogeny is given in Table 7.

- We have computed complete phylogenies by combining the main phylogenies for all groups and the phylogeny for each group. Results are presented in

Table 8.

The results have fulfilled our expectations: The phylogeny with the minimum number of incompatible characters and the maximum weight is the most plausible one from the point of view of Don Ringe.

Computing phylogenies for all groups is essential, since experts are usually interested in deep evolution of languages: They have more information on the “newer” languages compared to the “older"" languages. In that sense, the phylogenies in Table 3 are important for historical linguists.

As a result of the interaction between “newer” languages, there may be several possible phylogenies for each group as in seen in Table 7. These results are acceptable. However, one can find more accurate solutions for the groups of Indo-European languages than the solutions represented in Table 7, by taking into account the domain-specific information on groupings of languages in these groups.

3.7.2 *Quercus* Species

The second dataset we have experimented with is for the genus *Quercus* (oak trees) prepared by Yasin Bakış [3]. The dataset consists of 47 *Quercus* populations in different parts of Turkey. There are 37 characters for these populations; each character is mapped to 1 to 8 states. In addition, we have some domain-specific information about the expected groupings of the species and subgroupings: the populations can be grouped into 14 subgroups, and these subgroups can be grouped into 3 classes. According to this information, a phylogeny is preferable if these species are closer to each other with respect to this hierarchical grouping information.

We have constructed weighted phylogenies with Phylo-ASP with a divide-and-conquer approach for *Quercus*:

- We have reconstructed main phylogenies for all groups using the weight mea-

Table 3: Main phylogenies for all Indo-European language groups

	Phylogeny	n	w
1	(AN,((((GA,((IIR,BS),GE)),AL),IC),TO))		
2	(AN,(((GA,(AL,((IIR,BS),GE))),IC),TO))	0	43
3	(AN,(((((GA,(IIR,BS)),GE),IC),AL),TO))	0	43
4	(AN,(((((GA,(IIR,BS)),GE),AL),IC),TO))	0	43
5	(AN,((((GA,((IIR,BS),GE)),IC),AL),TO))	0	43
6	(AN,(((GA,((AL,(IIR,BS)),GE)),IC),TO))	0	43
7	(AN,((((GA,(AL,(IIR,BS))),GE),IC),TO))	0	43
8	(AN,(((((GA,(IIR,BS)),AL),GE),IC),TO))	0	43
9	(AN,((((GA,(IIR,(BS,GE))),IC),AL),TO))	0	43
10	(AN,((((GA,(IIR,(BS,GE))),AL),IC),TO))	0	42
11	(AN,(((GA,(AL,(IIR,(BS,GE)))),IC),TO))	0	42
12	(AN,(((((GA,(BS,GE)),IIR),IC),AL),TO))	0	42
13	(AN,(((((GA,(BS,GE)),IIR),AL),IC),TO))	0	42
14	(AN,(((GA,((AL,(BS,GE)),IIR)),IC),TO))	0	42
15	(AN,((((GA,(AL,(BS,GE))),IIR),IC),TO))	0	42
16	(AN,(((((GA,(BS,GE)),AL),IIR),IC),TO))	0	42
17	(AN,((((GA,(IIR,BS)),(AL,GE)),IC),TO))	0	42
18	(AN,(((GA,((AL,GE),(IIR,BS))),IC),TO))	0	41
19	(AN,((((GA,IIR),(AL,(BS,GE))),IC),TO))	0	41
21	(AN,((((GA,AL),((IIR,BS),GE)),IC),TO))	0	39
22	(AN,(((((GA,AL),GE),(IIR,BS)),IC),TO))	0	39
23	(AN,(((((GA,AL),(IIR,BS)),GE),IC),TO))	0	39
24	(AN,(((((GA,IIR),AL),(BS,GE)),IC),TO))	0	39
25	(AN,(((((GA,IIR),(BS,GE)),IC),AL),TO))	0	39
26	(AN,(((((GA,IIR),(BS,GE)),AL),IC),TO))	0	39
27	(AN,((((GA,AL),(IIR,(BS,GE))),IC),TO))	0	39
28	(AN,(((((GA,AL),(BS,GE)),IIR),IC),TO))	0	38
29	(AN,(((((GA,AL),IIR),(BS,GE)),IC),TO))	0	38
30	(AN,((((GA,(IIR,BS)),GE),(AL,IC)),TO))	0	38

Table 4: Main phylogenies for all Indo-European language groups

	Phylogeny	n	w
31	(AN,(((GA,((IIR,BS),GE)),(AL,IC)),TO))	0	36
32	(AN,(((GA,(IIR,(BS,GE))),(AL,IC)),TO))	0	36
33	(AN,((((GA,(BS,GE)),IIR),(AL,IC)),TO))	0	35
34	(AN,(((GA,(((AL,GE),BS),IIR)),IC),TO))	0	35
35	(AN,((((GA,((AL,GE),IIR)),BS),IC),TO))	0	34
36	(AN,(((GA,(((AL,GE),IIR),BS)),IC),TO))	0	34
37	(AN,((((((GA,IIR),BS),GE),IC),AL),TO))	0	34
38	(AN,((((GA,((IIR,GE),BS)),IC),AL),TO))	0	33
39	(AN,((((((GA,IIR),GE),AL),BS),IC),TO))	0	33
40	(AN,((((((GA,IIR),AL),GE),BS),IC),TO))	0	33
41	(AN,((((((GA,IIR),BS),AL),GE),IC),TO))	0	33
42	(AN,((((GA,((IIR,GE),BS)),AL),IC),TO))	0	33
43	(AN,((((((GA,IIR),GE),BS),AL),IC),TO))	0	33
44	(AN,((((((GA,IIR),AL),BS),GE),IC),TO))	0	33
45	(AN,((((((GA,IIR),GE),BS),IC),AL),TO))	0	33
46	(AN,((((((GA,IIR),BS),GE),AL),IC),TO))	0	33
47	(AN,(((GA,(AL,((IIR,GE),BS))),IC),TO))	0	33
48	(AN,(((((GA,(IIR,GE)),BS),IC),AL),TO))	0	33
49	(AN,(((GA,((AL,(IIR,GE)),BS)),IC),TO))	0	33
50	(AN,(((((GA,(IIR,GE)),BS),AL),IC),TO))	0	33
51	(AN,((((GA,(AL,(IIR,GE))),BS),IC),TO))	0	33
52	(AN,(((((GA,(IIR,GE)),AL),BS),IC),TO))	0	33
53	(AN,((((GA,IIR),(BS,GE)),(AL,IC)),TO))	0	33
54	(AN,((((((GA,AL),BS),GE),IIR),IC),TO))	0	32
55	(AN,((((((GA,AL),BS),IIR),GE),IC),TO))	0	32
56	(AN,((((((GA,AL),GE),IIR),BS),IC),TO))	0	32
57	(AN,((((((GA,AL),IIR),GE),BS),IC),TO))	0	32
58	(AN,((((((GA,AL),IIR),BS),GE),IC),TO))	0	32
59	(AN,((((((GA,AL),GE),BS),IIR),IC),TO))	0	32
60	(AN,(((((GA,IIR),BS),(AL,GE)),IC),TO))	0	32

Table 5: Main phylogenies for all Indo-European language groups

	Phylogeny	n	w
61	(AN,((((GA,IIR),((AL,GE),BS)),IC),TO))	0	31
62	(AN,(((((GA,IIR),(AL,GE)),BS),IC),TO))	0	31
63	(AN,((((GA,AL),((IIR,GE),BS)),IC),TO))	0	31
64	(AN,(((((GA,AL),BS),(IIR,GE)),IC),TO))	0	29
65	(AN,(((((GA,AL),(IIR,GE)),BS),IC),TO))	0	29
66	(AN,(((((GA,IIR),GE),BS),(AL,IC)),TO))	0	29
67	(AN,(((((GA,IIR),BS),GE),(AL,IC)),TO))	0	26
68	(AN,(((GA,((IIR,GE),BS)),(AL,IC)),TO))	0	26
69	(AN,((((GA,(IIR,GE)),BS),(AL,IC)),TO))	0	26
70	(AN,((((GA,(AL,GE)),(IIR,BS)),IC),TO))	0	26
71	(AN,((((GA,GE),(AL,(IIR,BS))),IC),TO))	0	21
72	(AN,(((((GA,GE),(IIR,BS)),IC),AL),TO))	0	19
73	(AN,(((((GA,GE),(IIR,BS)),AL),IC),TO))	0	19
74	(AN,(((((GA,GE),AL),(IIR,BS)),IC),TO))	0	19
75	(AN,((((GA,((AL,GE),BS)),IIR),IC),TO))	0	16
76	(AN,(((((GA,(AL,GE)),IIR),BS),IC),TO))	0	14
77	(AN,(((((GA,(AL,GE)),BS),IIR),IC),TO))	0	14
78	(AN,((((GA,GE),(IIR,BS)),(AL,IC)),TO))	0	14
79	(AN,((((((GA,GE),BS),IIR),IC),AL),TO))	0	12
80	(AN,((((((GA,GE),BS),IIR),AL),IC),TO))	0	12
81	(AN,((((((GA,GE),BS),AL),IIR),IC),TO))	0	12
82	(AN,((((((GA,GE),IIR),AL),BS),IC),TO))	0	12
83	(AN,((((((GA,GE),IIR),BS),IC),AL),TO))	0	12
84	(AN,((((((GA,GE),IIR),BS),AL),IC),TO))	0	12
85	(AN,((((((GA,GE),AL),IIR),BS),IC),TO))	0	12
86	(AN,((((((GA,GE),AL),BS),IIR),IC),TO))	0	12
87	(AN,(((((GA,GE),IIR),BS),(AL,IC)),TO))	0	12
88	(AN,(((((GA,GE),BS),IIR),(AL,IC)),TO))	0	5
89	(AN,((((((GA,BS),AL),GE),IIR),IC),TO))	0	5
90	(AN,((((((GA,BS),GE),IIR),AL),IC),TO))	0	2

Table 6: Main phylogenies for all Indo-European language groups

	Phylogeny	n	w
91	(AN,((((((GA,BS),GE),AL),IIR),IC),TO))	0	2
92	(AN,((((((GA,BS),IIR),GE),IC),AL),TO))	0	2
93	(AN,((((((GA,BS),AL),IIR),GE),IC),TO))	0	2
94	(AN,((((((GA,BS),IIR),GE),AL),IC),TO))	0	2
95	(AN,((((((GA,BS),IIR),AL),GE),IC),TO))	0	2
96	(AN,((((((GA,BS),GE),IIR),IC),AL),TO))	0	2
97	(AN,((((GA,BS),(IIR,GE)),(AL,IC)),TO))	0	2
98	(AN,((((GA,BS),(AL,(IIR,GE))),IC),TO))	0	0
99	(AN,(((((GA,BS),IIR),(AL,GE)),IC),TO))	0	0
100	(AN,(((((GA,BS),IIR),GE),(AL,IC)),TO))	0	0
101	(AN,(((((GA,BS),GE),IIR),(AL,IC)),TO))	0	0
102	(AN,((((GA,BS),((AL,GE),IIR)),IC),TO))	0	0
103	(AN,(((((GA,BS),(AL,GE)),IIR),IC),TO))	0	0
104	(AN,(((((GA,BS),(IIR,GE)),IC),AL),TO))	0	0
105	(AN,(((((GA,BS),(IIR,GE)),AL),IC),TO))	0	0
106	(AN,(((((GA,BS),AL),(IIR,GE)),IC),TO))	0	0

sure W3 and found max-weighted phylogenies with 9 incompatible characters and a weight of 18. The reason behind using W3 is to reflect the hierarchical grouping information, to compute more preferable phylogenies. We have also computed w-weighted phylogenies whose minimum number of incompatible characters is 14 and whose weight is at least 11. All phylogenies found for all groups are identified as plausible by Yasin Bakış. Results are presented in Table 10 and Table 11.

- Then we have reconstructed phylogenies for each group for each phylogeny for all groups, by using the weight measure W4. For example, in Table 12 and Table 13, the phylogenies are computed based on the main phylogeny 2 in Table 10. The maximum weights and the minimum number of incompatible characters of phylogenies are given in Tables 10 and 11.
- We have computed complete phylogenies by combining the main phylogenies for all groups and the phylogenies for each group. Results are presented in Table 14 and Table 15.

The results are fulfilled our expectations: The first two phylogenies with the minimum number of incompatible characters and the maximum weight is the most plausible ones from the point of view of Yasin Bakış. Also the eighth phylogeny in Table 10 is among the one of the three most plausible phylogenies.

Table 7: Phylogenies for each group for Indo-European languages

GroupName	Phylogeny	n	w
AN	((HI,LY),LU) ((HI,LU),LY) (HI,(LU,LY))	0	0
BS	((OC,(LI,LT)),PR) (OC,((LI,LT),PR)) ((OC,PR),(LI,LT)) (((OC,PR),LT),LI) (((OC,PR),LI),LT)	0	36
IC	(((OI,WE),LA),(OS,UM)) (((OI,WE),(OS,UM)),LA) ((((OI,WE),LA),UM),OS) ((OI,WE),(LA,(OS,UM))) ((((OI,WE),LA),OS),UM) (OI,((LA,(OS,UM)),WE)) ((OI,(LA,(OS,UM))),WE)	0	73
GE	(((OE,OG),GO),ON) (((OE,OG),ON),GO) ((OE,OG),(GO,ON)) (OE,((GO,ON),OG)) ((OE,(GO,ON)),OG)	0	18
IIR	((VE,PE),AV) ((VE,AV),PE) (VE,(AV,PE))	0	0
GA	(AR,GK)	0	0
TO	(TB,TA)	0	0

Table 8: Complete Phylogenies for Indo-European languages. All complete phylogenies in the table is formed by combining a large phylogeny (The column "CP" in this table indicates the index of that large phylogeny in Table 3) from Table 3 and the small phylogenies which are computed for that large phylogeny.

	Phylogeny	n	w	CP
1	(((HI,LY),LU),(((((((OC,(LI,LT)),PR),((VE,PE),AV)),(AR,GK)), ((((OE,OG),GO),ON),AL)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	30	2813	1
2	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),(AR,GK)), (((OE,OG),GO),ON)),(((OI,WE),LA),(OS,UM))),AL),(TB,TA)))	31	2827	2
3	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),(AR,GK)), (((OE,OG),GO),ON)),AL),(((OI,WE),LA),(OS,UM))),(TB,TA)))	31	2827	3
4	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),(AR,GK)), AL),(((OE,OG),GO),ON)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	31	2827	5
5	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),AL),(AR,GK)), (((OE,OG),GO),ON)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	31	2824	4
6	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),(((OE,OG),GO) ,ON)),(AR,GK)),AL),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2807	6
7	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),(((OE, OG),GO),ON)),(AR,GK)),(((OI,WE),LA),(OS,UM))),AL),(TB,TA)))	34	2807	8
8	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),(((OE, OG),GO),ON)),AL),(AR,GK)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2806	7
9	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),((VE,PE),AV)),AL), (((OE,OG),GO),ON)),(AR,GK)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2804	9
10	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)), ((VE,PE),AV)),(AR,GK)),(((OI,WE),LA),(OS,UM))),AL),(TB,TA)))	34	2780	10
11	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)), ((VE,PE),AV)),(AR,GK)),AL),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2780	11
12	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)),(AR,GK)), ((VE,PE),AV)),(((OI,WE),LA),(OS,UM))),AL),(TB,TA)))	34	2780	13
13	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)),(AR,GK)), ((VE,PE),AV)),AL),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2780	14
14	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)),(AR,GK)), ((VE,PE),AV)),AL),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2780	15
15	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)),(AR,GK)), AL),((VE,PE),AV)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2780	17
16	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)),((VE, PE),AV)),AL),(AR,GK)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2779	12
17	(((HI,LY),LU),((((((((OC,(LI,LT)),PR),(((OE,OG),GO),ON)),AL), (AR,GK)),((VE,PE),AV)),(((OI,WE),LA),(OS,UM))),(TB,TA)))	34	2778	16

Table 9: *Quercus* Species

Abbreviation	Species	Populations
AUC	*Q. aucheri*	AUC114, AUC117, AUC118
ILX	*Q. ilex*	ILX113, ILX206
COC	*Q. coccifera*	COC128, COC166, COC176, COC209
TRO	*Q. trojana*	TRO163, TRO185, TRO193, TRO220
CER	*Q. cerris*	BRA 156, BRA158, BRA081, BRA172
LIB	*Q. libani*	LIB151, LIB159
BRA	*Q. brantii*	CER137, CER181, CER210, CER200
ITH	*Q. ithaburensis*	ITH216, ITH186, ITH169, ITH124
PET	*Q. petraea*	PET115, PET087, PET204
INF	*Q. infectoria*	INF145, INF163, INF199, INF183
ROB	*Q. robur*]	ROB084, ROB208, ROB189, ROB140
MAC	*Q. macranthera*	MAC150, MAC140
FRA	*Q. frainetto*	FRA191, FRA208, FRA187
PUB	*Q. pubescens*	PUB133, PUB102, PUB182

Table 10: Main phylogenies for all *Quercus* groups - Part I

	Phylogeny	n	w
1	(((((AUC,(ILX,(FRA,PUB))),COC),PET), (INF,(ROB,MAC))),(((TRO,(BRA,ITH)),LIB),CER))	9	18
2	(((((AUC,(ILX,(FRA,PUB))),COC),PET), (INF,(ROB,MAC))),(((TRO,(LIB,ITH)),BRA),CER))	9	18
3	((((((AUC,ILX),COC),(ROB,MAC)),PET), (((TRO,((LIB,ITH),CER)),BRA),FRA)),(INF,PUB))	9	18
4	((((((AUC,ILX),MAC),PET),COC),(INF,ROB)), ((((TRO,LIB),(CER,ITH)),BRA),(FRA,PUB)))	9	18
5	((((((AUC,COC),MAC),PET),ILX),(INF,ROB)), ((((TRO,LIB),(CER,ITH)),BRA),(FRA,PUB)))	9	18
6	((((AUC,ILX),(COC,(FRA,PUB))),(((TRO,(BRA,LIB)),CER),ITH)), (((INF,PET),ROB),MAC))	9	18
7	(((((AUC,ILX),COC),(FRA,PUB)),(((TRO,CER),(BRA,(LIB,ITH))), (INF,PET))),(ROB,MAC))	9	18
8	((AUC,PUB),((((ILX,COC),(INF,FRA)),((ROB,PET),MAC)), (TRO,((BRA,LIB),(CER,ITH)))))	9	17
9	((((((AUC,ROB),(INF,PET)),(ILX,MAC)),(COC,PUB)), (((TRO,BRA),(LIB,CER)),ITH)),FRA)	10	13
10	((((AUC,ROB),INF),((COC,PUB),(MAC,FRA))),((ILX,PET), ((TRO,LIB),((BRA,ITH),CER))))	12	13
11	((((((AUC,ROB),FRA),COC),(TRO,((BRA,LIB),CER))), ((ILX,((INF,PUB),PET)),MAC)),ITH)	13	13
12	((((((AUC,ROB),FRA),COC),(TRO,((BRA,LIB),CER))), ((ILX,((INF,PET),PUB)),MAC)),ITH)	13	13
13	((((((AUC,INF),(ROB,FRA)),(MAC,PUB)),(COC,PET)), ((ILX,CER),BRA)),((TRO,ITH),LIB))	14	13
14	((((AUC,((BRA,CER),ROB)),(INF,(MAC,FRA))), (TRO,(LIB,ITH))),((ILX,PUB),(COC,PET)))	14	11
15	(((((((AUC,ROB),FRA),COC),(TRO,((BRA,LIB),CER))),MAC), (ILX,((INF,PUB),PET))),ITH)	14	12
16	((((AUC,((BRA,CER),ROB)),((ILX,PUB),(COC,PET))), (TRO,(LIB,ITH))),(INF,(MAC,FRA)))	14	11

Table 11: Main phylogenies for all *Quercus* groups - Part II

	Phylogeny	n	w
17	((((AUC,ILX),(LIB,CER)),((((COC,FRA),(ROB,PET)),PUB),(INF,MAC))),(TRO,(BRA,ITH)))	15	16
18	(AUC,((((((ILX,ITH),(BRA,CER)),(TRO,LIB)),((ROB,PET),PUB)),(INF,(MAC,FRA))),COC))	15	14
19	(AUC,(((((((ILX,(CER,ITH)),LIB),TRO),(INF,(ROB,(PET,(MAC,PUB))))),FRA),BRA),COC))	15	12
20	((AUC,(ROB,FRA)),((((ILX,MAC),PUB),((((TRO,ITH),(LIB,CER)),INF),BRA)),(COC,PET)))	16	11
21	((((((AUC,LIB),CER),TRO),BRA),((((((ILX,INF),FRA),ROB),MAC),PUB),PET)),(COC,ITH))	16	9
22	((((((((((AUC,TRO),(ILX,ITH)),COC),LIB),CER),BRA),((ROB,PET),FRA)),PUB),MAC),INF)	17	15
23	(((((AUC,CER),(TRO,LIB)),(((ILX,((COC,INF),FRA)),ROB),PUB)),BRA),(ITH,(PET,MAC)))	17	10
24	((((((((((AUC,TRO),(ILX,ITH)),COC),LIB),CER),BRA),((ROB,PET),FRA)),PUB),MAC),INF)	17	9

Table 12: Phylogenies for each group for *Quercus* - Part I

GroupName	Phylogeny	n	w
AUC	((AUC114,AUC118),AUC117) ((AUC114,AUC117),AUC118) (AUC114,(AUC117,AUC118))	0	0
ILX	(ILX113,ILX206)	0	0
COC	(((COC128,COC209),COC166),COC176) ((COC128,COC209),(COC166,COC176)) ((COC128,(COC166,COC176)),COC209) (COC128,((COC166,COC176),COC209)) (((COC128,COC209),COC176),COC166)	2	5
TRO	((TRO163,(TRO193,TRO220)),TRO185) (TRO163,(TRO185,(TRO193,TRO220))) (((TRO163,TRO185),TRO193),TRO220) (((TRO163,TRO185),TRO220),TRO193) ((TRO163,TRO185),(TRO193,TRO220))	1	6
BRA	(BRA156,(BRA158,(BRA081,BRA172))) ((BRA156,(BRA158,BRA172)),BRA081) ((BRA156,(BRA081,BRA172)),BRA158)	1	3
LIB	((LIB151,LIB087),LIB159) ((LIB151,LIB159),LIB087) (LIB151,(LIB159,LIB087))	0	0

Table 13: Phylogenies for each group for *Quercus* - Part II

GroupName	Phylogeny	n	w
CER	((CER137,CER181),(CER210,CER200))	0	4
ITH	((ITH216,ITH169),(ITH186,ITH124)) ((ITH216,ITH124),(ITH186,ITH169)) ((ITH216,ITH186),(ITH169,ITH124))	2	4
INF	(INF145,(INF163,(INF199,INF183))) ((INF145,(INF199,INF183)),INF163) (((INF145,INF163),INF199),INF183)	1	0
ROB	((ROB084,(ROB208,ROB140)),ROB189) ((ROB084,ROB189),(ROB208,ROB140)) (ROB084,((ROB208,ROB140),ROB189)) (((ROB084,ROB189),ROB208),ROB140)	0	2
PET	((PET115,PET204),PET087) ((PET115,PET087),PET204)	0	0
MAC	(MAC150,MAC140)	0	0
FRA	((FRA208,FRA187),FRA191) ((FRA208,FRA191),FRA187) (FRA208,(FRA191,FRA187))	0	0
PUB	((PUB133,PUB182),PUB102) ((PUB133,PUB102),PUB182) (PUB133,(PUB102,PUB182))	0	0

Table 14: Complete Phylogenies for genus *Quercus* - Part I. All complete phylogenies in the table is formed by combining a main phylogeny (The column "CP" in this table indicates the index of that main phylogeny in Table 10 and Table 10) and the small phylogenies which are computed for each subgroup.

	Phylogeny	n	w	CP
1	(((((((AUC114,AUC118),AUC117),((ILX113,ILX206), (((FRA208,FRA187),FRA191),((PUB133,PUB182),PUB102)))), (((COC128,COC209),COC166),COC176)),((PET115,PET204), PET087)),(((INF145,INF163),(INF199,INF183)), (((ROB084,ROB189),(ROB208,ROB140)),(MAC150,MAC140)))), (((((TRO163,TRO185),(TRO193,TRO220)),(((LIB151,LIB087), LIB159),((ITH216,ITH169),(ITH186,ITH124)))),((BRA156,BRA081), (BRA158,BRA172))),((CER137,CER181),(CER210,CER200))))	31	282	2
2	((((((((AUC114,AUC118),AUC117),(ILX113,ILX206)),(MAC150,MAC140)), ((PET115,PET204),PET087)),(((COC128,COC209),COC166),COC176)), (((INF145,INF163),(INF199,INF183)),((ROB084,ROB189),(ROB208, ROB140)))),((((((TRO163,TRO185),(TRO193,TRO220)),((LIB151, LIB087),LIB159)),(((CER137,CER181),(CER210,CER200)),((ITH216 ,ITH169),(ITH186,ITH124)))),((BRA156,BRA081),(BRA158,BRA172))), (((FRA208,FRA187),FRA191),((PUB133,PUB182),PUB102))))	31	282	4
3	(((((((AUC114,AUC118),AUC117),(ILX113,ILX206)),(((COC128,COC209), COC166),COC176)),(((FRA208,FRA187),FRA191),((PUB133,PUB182), PUB102))),(((((TRO163,TRO185),(TRO193,TRO220)),((CER137,CER181), (CER210,CER200))),(((BRA156,BRA081),(BRA158,BRA172)),(((LIB151, LIB087),LIB159),((ITH216,ITH169),(ITH186,ITH124))))),(((INF145,INF163), (INF199,INF183)),((PET115,PET204),PET087)))),(((ROB084,ROB189), (ROB208,ROB140)),(MAC150,MAC140)))	31	282	7

Table 15: Complete Phylogenies for genus *Quercus* - Part II. All complete phylogenies in the table is formed by combining a main phylogeny (The column "CP" in this table indicates the index of that main phylogeny in Table 10 and Table 10) and the small phylogenies which are computed for each subgroup.

	Phylogeny	n	w	CP
4	(((((((AUC114,AUC118),AUC117),((ILX113,ILX206),(((FRA208, FRA187),FRA191),((PUB133,PUB182),PUB102)))),(((COC128,COC209), COC166),COC176)),((PET115,PET204),PET087)),(((INF145,INF163), (INF199,INF183)),(((ROB084,ROB189),(ROB208,ROB140)),(MAC150, MAC140)))),(((((TRO163,TRO185),(TRO193,TRO220)),(((BRA156, BRA081),(BRA158,BRA172)),((ITH216,ITH169),(ITH186,ITH124)))), ((LIB151,LIB087),LIB159)),((CER137,CER181),(CER210,CER200))))	31	282	1
5	((((((((AUC114,AUC118),AUC117),(((COC128,COC209),COC166),COC176)), (MAC150,MAC140)),((PET115,PET204),PET087)),(ILX113,ILX206)), (((INF145,INF163),(INF199,INF183)),((ROB084,ROB189),(ROB208, ROB140)))),((((((TRO163,TRO185),(TRO193,TRO220)),((LIB151,LIB087), LIB159)),(((CER137,CER181),(CER210,CER200)),((ITH216,ITH169), (ITH186,ITH124)))),((BRA156,BRA081),(BRA158,BRA172))),(((FRA208, FRA187),FRA191),((PUB133,PUB182),PUB102))))	31	282	5
6	((((((AUC114,AUC118),AUC117),(ILX113,ILX206)),((((COC128,COC209), COC166),COC176),(((FRA208,FRA187),FRA191),((PUB133,PUB182), PUB102)))),(((((TRO163,TRO185),(TRO193,TRO220)),(((BRA156, BRA081),(BRA158,BRA172)),((LIB151,LIB087),LIB159))),((CER137, CER181),(CER210,CER200))),((ITH216,ITH169),(ITH186,ITH124)))), (((((INF145,INF163),(INF199,INF183)),((PET115,PET204),PET087)), ((ROB084,ROB189),(ROB208,ROB140))),(MAC150,MAC140)))	31	282	6
7	((((((((AUC114,AUC118),AUC117),(ILX113,ILX206)),(((COC128 ,COC209),COC166),COC176)),(((ROB084,ROB189),(ROB208,ROB140)), (MAC150,MAC140))),((PET115,PET204),PET087)),(((((TRO163,TRO185), (TRO193,TRO220)),((((LIB151,LIB087),LIB159),((ITH216,ITH169), (ITH186,ITH124))),((CER137,CER181),(CER210,CER200)))),((BRA156, BRA081),(BRA158,BRA172))),((FRA208,FRA187),FRA191))), (((INF145,INF163),(INF199,INF183)),((PUB133,PUB182),PUB102)))	32	235	3
8	((((AUC114,AUC118),AUC117),((PUB133,PUB182),PUB102)), (((((ILX113,ILX206),(((COC128,COC209),COC166),COC176)), (((INF145,INF163),(INF199,INF183)),((FRA208,FRA187),FRA191))), ((((ROB084,ROB189),(ROB208,ROB140)),((PET115,PET204),PET087)), (MAC150,MAC140))),(((TRO163,TRO185),(TRO193,TRO220)),((((BRA156, BRA081),(BRA158,BRA172)),((LIB151,LIB087),LIB159)),(((CER137,CER181), (CER210,CER200)),((ITH216,ITH169),(ITH186,ITH124)))))))	32	235	8

4 Reconstructing Weighted Phylogenetic Networks using ASP

In the previous chapter, we studied phylogenetic trees. However, phylogenetic trees are not fully adequate in the study of evolutionary relations between taxonomic units because they do not represent borrowings. For example, consider languages as taxonomic units. They do not only inherit characteristics from their ancestors (such relations can be represented by phylogenetic trees), but also they may borrow characteristics from other languages. We can represent these borrowings by adding a small number of edges to a phylogenetic tree. Hence, we can construct a phylogenetic network in 2 steps: (1) Build a phylogenetic tree. (2) Obtain a phylogenetic network from the phylogenetic tree by adding a small number of edges.

Similar to phylogenetic trees, some phylogenetic networks can also be more plausible than the others from the point of view of experts. In order to automatically pick more plausible phylogenetic networks, we define some weight functions to reflect the plausibility of networks and introduce methods to compute weighted phylogenetic networks whose weight is among a given threshold.

4.1 Preliminaries

Before describing the computational problems related to weighted phylogenetic network reconstruction, we need to introduce some definitions as in [12].

4.1.1 Temporal Networks

A *temporal phylogeny* is a phylogeny along with a function τ from vertices of the phylogeny to real numbers such that for every edge $\langle u, v \rangle$ of the phylogeny $\tau(u) < \tau(v)$ (See Figure 8). Intuitively, $\tau(v)$ is the time when language v was spoken. We will graphically represent the values of τ by placing a vertical time line to the right of the tree.

A *contact* between two linguistic communities can be represented by a horizontal edge added to a pictorial representation of a temporal phylogeny. The two endpoints of the edge are a simultaneous "event" in the histories of these communities. An event can be represented by a pair $v \uparrow t$, where v is a vertex of the phylogeny and t is a real number.

Consider a temporal phylogeny T; let V be the set of its vertices, R its root, and τ its time function. For every $v \in V \backslash \{R\}$, let $par(v)$ be the parent of v. An *event* is any pair $v \uparrow t$ such that $v \in V \backslash \{R\}$ and t is a real number satisfying the inequalities

$$\tau(par(v)) \leq t \leq \tau(v). \tag{4.1}$$

Events $v \uparrow t$ and $v' \uparrow t'$ are concurrent if $t = t'$. A *contact* is a set consisting of two different concurrent events, denoted by $\{B \uparrow t_1, D \uparrow t_1\}$.

Any finite set C of contacts defines a *temporal (phylogenetic) network* (see Figure 1) - a digraph obtained from T by inserting the elements $v \uparrow t$ of the contacts from C as intermediate vertices and then adding every contact in C as a bidirectional edge.

4.1.2 k-Simple Contacts

A set C of contacts is *k-simple* if

- for every event $v \uparrow t$ that belongs to a contact from C, $t < \tau(v)$ and
- for every vertex v of T there exist at most k distinct number $t_1, .., t_k$ such that t_i belongs to some contact from C.

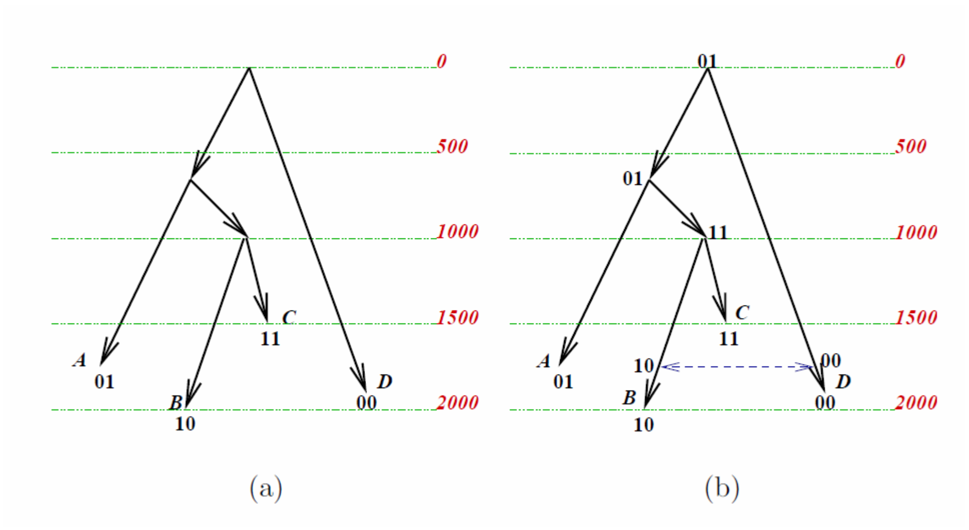


Figure 8: A temporal phylogeny (a), and a perfect temporal network (b) with a lateral edge connecting $B \uparrow 1750$ with $D \uparrow 1750$.

For a vertex $v \in V$, let us denote by $V_c(v)$ of events $v \uparrow t$ in V_C.

If C is k-simple, then the corresponding network (see Figure 9) can be described as follows. The set of its vertices is the union of the set V of vertices of T with the union V_C of the contacts from C. Its set E_C of edges is obtained from the set E of edges of T in two steps. First, for every edge $\langle par(v), v\rangle$ in E, if $V_c(v) \neq \emptyset$, then we replace the edge $\langle par(v), v\rangle$ by $|V_c(v)|+1$ edges.

$$\langle par(v), v \uparrow t_1\rangle, \langle v \uparrow t_1, v \uparrow t_2\rangle, ..., \langle v \uparrow t_{|V_c(v)|}, v\rangle$$

such that $v \uparrow t_i \in V_C(v)$ and for all $i, j\,(i < j)$ iff $(t_i < t_j)$.

Second, for every contact $\langle u \uparrow t_i, v \uparrow t_i\rangle$ in C we add a "bidirectional lateral edge" – the pair of edges

$$\langle u \uparrow t_i, v \uparrow t_i\rangle \text{ and } \langle v \uparrow t_i, u \uparrow t_i\rangle.$$

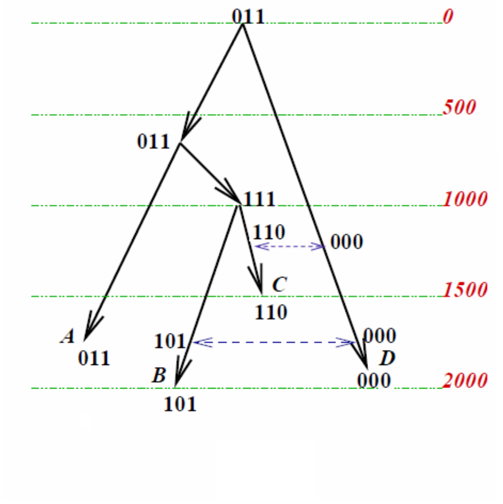


Figure 9: A perfect temporal network with k-simple contacts with 2 lateral edges connecting $D \uparrow 1200$ with $C \uparrow 1200$ and $D \uparrow 1750$ with $B \uparrow 1750$.

A k-simple set C of contacts (and the corresponding network $\langle V \cup V_C, E_C \rangle$) is *perfect* if there exists a function $g : (V \cup V_C) \times I \to S$ such that

(i) for every leaf v of T and every $i \in I$, $g(v,i) = f(v,i)$;

(ii) for every $i \in I$ and every $s \in S$, if the set

$V_{is} = \{x \in V \cup V_C : g(x,i) = s\}$

is nonempty then the digraph $\langle V \cup V_C, E_C \rangle$ has a subgraph with the set V_{is} of vertices that is a rooted tree.

Note that if $k = 1$, then a k-simple set of contacts is also a simple set of contacts as described in [35], since in the simple set of contacts, there is at most one t such that $v \uparrow t$ belongs to some contact from C. Therefore, our definition generalizes simple set of contacts.

4.1.3 Summaries of k-Simple Contacts

The information included in a temporal phylogeny is insufficient for determining the exact dates of the contacts that turn it into a perfect network. To make this idea precise, let us first select for each $v \in V \setminus \{R\}$, $|V_C(v)|$ symbols $v \uparrow_z$ where $(1 \leq z \leq |V_C(v)|)$, and let us denote the set of new symbols by $V_S(v)$. For each vertex $v \in V \backslash \{R\}$ we define a function s that maps each element $v \uparrow t$ in $V_C(v)$ to an element of $V_S(v)$ satisfying the following condition:

- For every $v \uparrow t$ and $v \uparrow t'$ in $V_C(v)$, let $v \uparrow_z = s(v \uparrow t)$ and $v \uparrow_{z'} = s(v \uparrow t')$. Then $t < t'$ iff $z < z'$.

Then we define the "summary" of a k-simple set C of contacts to be the result of replacing each element $v \uparrow t$ of every contact in C with $v \uparrow_z$ with respect to the function s. Thus summaries consist of 2-element subset of the set

$$V \uparrow_k = \{v \uparrow_z : v \in V \backslash \{R\}, 1 \leq z \leq k\}.$$

Intuitively, $v \uparrow_z$ is a language intermediate between $v \uparrow_{z-1}$ ($v \uparrow_{z-1} = par(v)$ if $z = 1$) and $v \uparrow_{z+1}$ ($v \uparrow_{z+1} = v$ if $z = |V_s(v)|$) that was spoken at some unspecified time between $\tau(v \uparrow_{z-1})$ and $\tau(v \uparrow_{z+1})$.

4.2 Weighted Networks

The motivation behind computing weighted networks is similar to computing weighted phylgenetic trees: There may be many possible phylogenetic networks as solutions for a problem and some solutions may be more desirable from the experts' point of view. Therefore, we formulate weight measures to reflect the plausibility of the networks, and we pick distinct solutions over a weight threshold.

The weight function we have formulated for phylogenetic networks is as follows: We define the weight measure of a phylogenetic network in such a way that the bilateral edges are added as close to the root as possible.

Consider a phylogenetic network N with the set V_S of summaries and with the set L of leaves. The weight of the network N is defined as the sum of the weights of all summaries:

$$\sum_{s \in V_S} w(s) \tag{4.2}$$

The weight $w(s)$ of a summary $s = \langle u \uparrow_z, v \uparrow_z \rangle$ is defined as the minimum of two values : the number of vertices in the shortest path of the paths from u to a leaf $l \in L$ (including u and l) and the number of vertices in the shortest path of the paths from v to a leaf $l \in L$(including v and l):

$$w(s) = min(height(u \uparrow_z), height(v \uparrow_z)) \tag{4.3}$$

where $height(v \uparrow_z)$ is the number of vertices in the shortest path of the paths from v to any leaf $l \in L$ (including v and l).

4.3 Problem Definitions

We study the following computational problems related to reconstruction of networks:

Increment to Perfect k-Simple Temporal Network Problem (k-IPSTN)

k-IPSTN is defined by a phylogeny $\langle V, E, L, I, S, f \rangle$, a function

$$v \mapsto (\tau_{min}(v), \tau_{max}(v))$$

from the vertices of the phylogeny to open intervals (in other words, for every $v \in V$, $\tau_{min}(v)$ is a real number or $-\infty$, and $\tau_{max}(v)$ is a real number or $+\infty$, such that $\tau_{min}(v) < \tau_{max}(v)$) and an integer k. A *solution* to the problem is a set of 2-element subsets of $V \uparrow_k$ that is the summary of a perfect k-simple set of contacts for a temporal phylogeny $\langle V, E, L, I, S, f, \tau \rangle$ such that, for all $v \in V$,

$$\tau_{min}(v) < \tau(v) < \tau_{max}(v). \tag{4.4}$$

n-WEIGHTED INCREMENT TO PERFECT k-SIMPLE TEMPORAL NETWORK PROBLEM (nk-IPSTN)

Given a k-IPSTN problem Q with a phylogeny $\langle V, E, L, I, S, f \rangle$, two nonnegative integers l and n and a weight function $w : V \uparrow_k \rightarrow \mathbb{N}$; we want to find solutions X to Q such that the cardinality of X is at most l and the weight of the corresponding phylogenetic network (i.e. $\sum_{x \in X} w(x)$) is at least n.

Note that nk-IPSTN generalizes IPSTN: Consider a weight function that maps every summary to 1, and take k=1.

nk-IPSTN problem can be expressed as a decision problem as follows: Given a k-IPSTN problem Q with a phylogeny $\langle V, E, L, I, S, f \rangle$, two nonnegative integers l and n and a weight function $w : V \uparrow_k \rightarrow \mathbb{N}$; decide the existence of a solution X to Q such that the cardinality of X is at most l and the weight of the corresponding phylogenetic network (i.e. $\sum_{x \in X} w(x)$) is at least n.

Proposition 10. *nk-IPSTN problem is NP-hard.*

Proof. Perfect Phylogeny Problem[10] is a special kind of nk-IPSTN problem, if we

[10]Perfect Phylogeny Problem (PP) is NP-complete as proved in [8]: PP is polynomially equivalent with Triangulating Colored Graph problem and Triangulating Colored Graph Problem is NP-complete.

take the weight function as a function that maps every summary to 1, $k = 0$ and $l = 0$. Therefore, nk-IPSTN problem is NP-hard.

$\square$

Solutions as admissible sets Consider a phylogeny $\langle V, E, L, I, S, f\rangle$ with a root R and a set X of 2-element subsets of $V\uparrow_k$. By V_X we denote the union of all elements of X. By E_X we denote the set obtained from E by replacing, if $V_S(v) \neq 0$, the edge $\langle par(v), v\rangle$ with $|V_S(v)|{+}1$ edges:

$$\langle par(v), v\uparrow_{z_1}\rangle, \langle v\uparrow_{z_1}, v\uparrow_{z_2}\rangle, ..., \langle v\uparrow_{z_{|V_S(v)|}}, v\rangle$$

such that $v\uparrow_{z_i}$ and for all $i, j\,(i < j)\,\text{iff}\,(z_i < z_j)$ and adding, for every element $\{u\uparrow_j, v\uparrow_z j\}$ of X, the edges

$$\langle u\uparrow_j, v\uparrow_j\rangle \,\text{and}\, \langle v\uparrow_j, u\uparrow_j\rangle.$$

We say that X is *admissible* if there exists a function $g : (V \cup V_X) \times I \to S$ such that

(i) for every leaf v of the phylogeny and every $i \in I$, $g(v, i) = f(v, i)$;

(ii) for every $i \in I$ and every $s \in S$, if the set

$$V_{is} = \{x \in V \cup V_X : g(x, i) = s\}$$

is non empty, then the digraph $\langle V \cup V_X, E_X\rangle$ has a subgraph with the set V_{is} of vertices that is a rooted tree.

In the following proposition, Q is an nk-IPSTN problem defined by a phylogeny $\langle V, E, L, I, S, f\rangle$ with a root R, a function $v \mapsto (\tau_{min}(v), \tau_{max}(v))$, an integer k and a weight function $w : V\uparrow_k \to \mathbb{N}$.

Proposition 11. *A set X of 2-element subsets of $V\uparrow_k$ is a solution to Q iff*

(i) X is admissible, and

(ii) there exists a real valued function τ on $V \cup V_X$ such that

(a) for every $v \in V$,

$$\tau_{min}(v) < \tau(v) < \tau_{max}(v),$$

(b) for every $v \in V \backslash \{R\}$,

$$\tau(par(v)) < \tau(v),$$

(c) for every element $v\uparrow_j$ of V_X,

$$\tau(par(v)) < \tau(v\uparrow_j) < \tau(v),$$

(d) for every element $v\uparrow_j$ of V_X,

$$\tau(v\uparrow_j) < \tau(v\uparrow_{j+1}),$$

(e) for every element $\{u\uparrow_j, v\uparrow_j\}$ of X,

$$\tau(u\uparrow_j) = \tau(v\uparrow_j).$$

Proof. Left-to-right. Assume that X is a solution to Q, so that there exist a real-valued function τ on V satisfying (2) and a perfect k-simple set C of contacts for the temporal phylogeny $\langle V, E, I, S, f, \tau\rangle$ such that X is the summary of C. The function from V_C to V_X that maps every event $v\uparrow t$ to $v\uparrow_z$ is a 1-1 correspondence between two sets. If we agree to identify every event $v\uparrow t$ with its image $v\uparrow_z$ under this correspondence then E_C becomes identical to E_X, and the conditions on g in the definition of a perfect set of k-simple contacts turn into the conditions on g in the definition of an admissible set. Consequently, (i) follows from the fact that C is perfect. To prove (ii), extend τ from V to $V \cup V_X$:

$$\tau(v \uparrow_z) = t \,\text{if}\, v \uparrow t \in V_C.$$

Part (a) follows from (2); part (b) follows from the definition of a temporal phylogeny: part (c) follows from (1); part (d) follows from the definition of summaries of k-simple contacts; part (e) follows from the definition of a contact.

right-to-left. Assume that X satisfies conditions (i) and (ii). Consider the temporal phylogeny T that consists of the phylogeny $\langle V, E, I, S, f\rangle$ and the function τ restricted to V. By (a), T satisfies (2). Let C be the set obtained from X by replacing the symbols $v \uparrow_k$ in every element of X with the event $v \uparrow t$ where $t = \tau(v \uparrow_k)$. From (e), we conclude that the elements of C are contacts; by (c) and (d), C is k-simple. It is clear that X is the summary of C. The same reasoning as in the first half of the proof shows that, in view of (i), C is perfect. □

4.4 ASP Formulation

4.4.1 Phylogenetic Network Reconstruction

ASP formulation of phylogenetic network reconstruction is done in two parts as in [35]: In the first part, admissible sets are computed; and in the second part, for each of these admissible sets, whether the equations or inequalities from part (ii) of the statement Proposition 11 have a solution in real numbers $\tau(v)$, $v \in V \cup V_X$.

In phylogenetic networks, due to lateral edges, there may be loops in the graph; that we need to check reachability in such a graph prevents us to use SAT solver or a Constraint Programming system because of the necessity of enumerating all variables.

4.4.2 Weight Functions

The weight measure we have formulated in ASP which is described in Subsection 4.2 is as follows:

We describe the weight of a phylogenetic network as an ASP program in four parts. Suppose that the schematic variable `W` denotes the weight of a bilateral edge, `NW` denotes the network weight and `C1, C2, C, CC` denote the contacts.

First, we give an order to each bilateral edge and we make sure that two different bilateral edges can not have the same order:

```
1{order(U,C1,V,C2,O2):orderrange(O2)}1 :- new(U,C1,V,C2).

:- order(U,C1,V,C2,O), order(U1,CC,V1,C,O), new(U,C1,V,C2),
   new(U1,CC,V1,C), U!=U1.
:- order(U,C1,V,C2,O), order(U1,CC,V1,C,O), new(U,C1,V,C2),
   new(U1,CC,V1,C), C1!=CC.
:- order(U,C1,V,C2,O), order(U1,CC,V1,C,O), new(U,C1,V,C2),
   new(U1,CC,V1,C), V!=V1.
:- order(U,C1,V,C2,O), order(U1,CC,V1,C,O), new(U,C1,V,C2),
   new(U1,CC,V1,C), C!=C2.
```

Second, we find the weight of an edge:

```
weightsOfEdges(O,W) :- order(U,C1,V,C2,O), w(U,C1,V,C2,W).
weightsOfEdges(O,0) :- not weightedEdges(O).

w(U,C1,V,C2,D) :- order(U,C1,V,C2,O),height(O,D).
weightedEdges(O) :- new(U,C1,V,C2), order(U,C1,V,C2,O), w(U,C1,V,C2,W).
```

Third, we find the weight of the network by adding up the weights of edges:

```
weightOfTheNetwork(NW) :- totalWeightOfEdges(NW,k).

totalWeightOfEdges(NW,1) :- weightsOfEdges(1,NW).
totalWeightOfEdges(NW+W,O) :- weightsOfEdges(O,W),
       totalWeightOfEdges(NW,O-1).
```

Finally, we describe the weight constraint, to ensure that the weight of the phylogenetic network is larger than $maxW$, as follows:

```
:- weightOfTheNetwork(NW), NW<maxW.
```

4.5 Computational Methods for Reconstructing Phylogenetic Networks

We have studied two methods for reconstructing weighted phylogenetic networks: the representation-based method and search-based method.

4.5.1 Representation-Based Method

In the representation-based method, we modify the representation of the problem, to compute weighted phylogenetic networks. In order to do that, we formulate the phylogenetic network reconstruction as an ASP program P as described in Subsection 4.4.1 . Then we formulate the weight function as an ASP program W as described in the Subsection 4.4.2. Finally, we compute weighted phylogenetic networks by computing the answer sets for the the ASP program $P \cup W$.

4.5.2 Search-Based Method

Computing weighted phylogenetic networks with the search-based method is very similar to computing weighted phylogenetic trees with the search-based method: In order to compute weighted phylogenetic networks, we define a heuristic function to estimate an upper bound for the weight of a network as we have proposed in Subsection 4.2, and we use the answer set solver CLASP-W to compute weighted phylogenies.

We define the heuristic function to estimate an upper bound of the weight of a network is as follows.

Let A be a partially constructed phylogenetic network of a subset of a complete phylogenetic network N built over a phylogeny P. Let S be the set of summaries of N, S_A be the set of summaries of A, and $maxW$ be the maximum weight that a summary can have. We define the heuristic function with respect to A and S_A:

$$UB(A, S_A) = \sum_{\substack{s \in S_A,\text{ and height of} \\ \text{an event of } s \text{ is defined}}} w(c) + \sum_{\substack{s \notin S_A,\text{ or height of the events} \\ \text{of } s \text{ is not defined}}} maxW \qquad (4.5)$$

4.6 PhyloNet-ASP

PhyloNet-ASP is a tool for reconstructing phylogenetic networks. It is designed and implemented to solve the problems given in Subsection 4.2.

Input: PhyloNet-ASP takes three kinds of input:

- Character states of taxonomic units for every character
- A phylogenetic tree in Newick format

- A nonnegative integer n, which is the maximum number of bilateral edges that a network can have (Optional)
- A nonnegative integer w, which is the is the minimum weight that a phylogenetic network can have (Optional)

Output: PhyloNet-ASP outputs the computed phylogenetic network in extended newick format as well as the uninformative, non-unique and incompatible characters.

The input, other than character states, are optional. Depending on the given input, PhyloNet-ASP can solve different kinds of problems:

- If n is given as an input, PhyloNet-ASP solves k-IPSTN problem: It reconstructs all (or a desired number of) phylogenetic networks with at most k bilateral edges with respect to the character states.
- If w is given as an input, PhyloNet-ASP solves nk-IPSTN problem: It reconstructs all (or a desired number of) phylogenetic networks with at least n weight with respect to the character states and the weight measure.

4.7 Experimental Results

We have reconstructed weighted phylogenetic trees for Indo-European languages and *Quercus* species with our system.

For Indo-European languages, we have picked the phylogeny with the minimum number of incompatible characters and maximum weight from Table 8 and we have constructed a weighted phylogenetic network by adding a contact between Balto-Slavic and Germanic, and a contact between Germanic and Italo-Celtic. The weight of this network is 9.

For *Quercus* species, we have picked the phylogeny with the minimum number of incompatible characters and maximum weight from Table 10 and we have constructed a weighted phylogenetic network.

5 Related Work

The published literature on phylogeny reconstruction can be classified into two categories: phenetic methods based on distances and cladistics methods based on characters.

The most popular phenetic methods are Unweighted Pair Group Method Using Arithmetic Averages (UPGMA) [75] and Neighbour Joining (NJ) [63]. In UPGMA, each species is considered as clusters and the closest two clusters with respect to their distance are joined together and the distance of the joint pair is then recalculated by taking their average. In [2], [21]and [54], phylogenetic trees are computed using this method and its modified versions. In NJ, each pair is checked for being joined and the sum of all branches length is calculated of the resulting tree. The pair with the smallest sum is taken as the closest neighbors and joined together. Then the branch length is recalculated. In [63, 37, 74], this method is used to compute phylogenetic trees.

The most popular cladistics methods are Maximum Parsimony (MP)[27], Maximum Likelihood(ML) and Maximum Compatibility (MC)[19], In MP, all possible trees are evaluated and are assigned a score according to the number of evolutionary changes in the tree. The best tree is then the one that minimized the overall number of mutations. Phylogeny reconstruction with MP is studied in [68]. In Maximum Likelihood, all possible trees are evaluated as in MP, and then the one with the maximum likelihood is picked as the best tree. For a given tree, the likelihood is determined by the probability that a certain evolutionary model has generated the observed data. Phylogeny reconstruction with ML is studied in [47], [87]and [86].

We reconstruct phylogenies with maximum compatibility method. In [7], authors compare two methods: weighted maximum parsimony and weighted maximum compatibility. Weighted maximum parsimony is an extension of parsimony, where the characters are assigned to some weight. The aim of this method is to find a tree in which the total weighted number of character state changes is minimized. Similarly, weighted maximum compatibility is derived from maximum compatibility, whose aim is to find the tree with the maximum weighted compatibility score, which is computed by adding up all the weights of each character which is compatible with the tree. Their experimental results indicate that phylogenies reconstructed with character-based methods are more accurate than the phylogenies reconstructed with distance-based methods for languages.

There exist several phylogeny reconstruction tools which uses these methods. One of the most most widely-distributed phylogeny package is PHYLIP[11] (PHYLogeny Inference Package) which consists of 35 programs for inferring and comparing phylogenies with different methods. Implemented methods for reconstructing phylogenies includes maximum parsimony, maximum compatibility, distance matrix and maximum likelihood methods. The tool (CLIQUE) that implements the maximum compatibility problem considers datasets with binary states only. Therefore, it can not be used for Indo-European languages.

PAUP*[12](Phylogenetic Analysis Using Parsimony) is an another tool which includes phylogeny reconstruction with parsimony, distance matrix, invariants, and maximum likelihood methods.

Apart from reconstructing the phylogenies, there exist some tools to analyze phylogenetic data. These systems can analyze the phylogenies and character evolution. For example, MacClade[13] analyzes the evolution of a variety of character types. Random Cladistics[14] is also an analyzing tool, which uses bootstrapping,

[11]http://evolution.gs.washington.edu/phylip.html
[12]http://paup.csit.fsu.edu/software
[13]http://macclade.org/macclade.html
[14]http://research.amnh.org/ siddall/rc.html

jackknifing, and several kinds of permutation tests.

Our systems Phylo-ASP and PhyloNet-ASP are different from the existing reconstruction systems:

- They are based on the compatibility criterion.
- They can compute weighted phylogenetic trees and weighted phylogenetic networks.
- They can integrate domain-specific information in the process of reconstructing weighted phylogenies and weighted phylogenetic networks.
- They can generate more than one weighted phylogenetic tree/network.
- They can be used to analyze the given data (e.g., identify its informative parts) and/or the given phylogenies (e.g., to check the incompatibility of the evolution of a trait with respect to a phylogeny).

On the other hand, since our systems are based on the compatibility criterion, they can not be used with genomic data due to possible occurrences of backmutations.

6 Conclusion

In this thesis, we have studied constructing weighted phylogenies and networks by using answer set programming and generalized these results to weighted solutions in ASP. Our contributions are as follows :

- We have defined various optimization and decision problems for computing weighted phylogenies and phylogenetic networks and analyzed their computational complexity : Maximum Weighted Compatibility Problem(MWCP),w-weighted compatibility problem(w-WCP) and w-weighted n-compatibility problem(wn-WCP). We have proved that w-WCP and wn-WCP are NP-complete (Proposition 2).

- We have introduced two sorts of computational methods to compute weighted phylogenies and phylogenetic networks: the first class of methods suggests modifying the ASP representation of the problem to compute weighted phylogenies using an existing ASP solver and the other class suggests modifying the search algorithm of the answer set solver to compute weighted phylogenies incrementally based on branch-and-bound. In the representation-based method, weight measure is defined in ASP. In the search-based method, weight measure is defined externally in C++.

- Based on these methods, in order to compute weighted phylogenies for large datasets efficiently, we have introduced a novel divide-and-conquer approach for computing weighted phylogenies by inferring its smaller subtrees. This approach also makes use of domain-specific information provided by the experts. Considering that for instance the earlier ASP-based phylogenetics systems [12],[35] could not compute the whole phylogeny for Indo-European languages automatically (since the dataset is too large), our divide-and-conquer method is useful and effective in reconstructing large phylogenies as also verified by experiments.

- We have generalized the representation-based method and the search-based method to compute weighted solutions in ASP so that other domains can benefit from these mthods as well. For example, in a planning problem, we can define the weight of a plan in terms of the costs of actions, and then compute the distinct plans whose weights are less than a given value.

- We have implemented the search-based method for computing weighted solutions in ASP, by modifying the search algorithm of the answer set solver CLASP (and called it CLASP-W) in the spirit of branch-and-bound. Since the heuristic functions to estimate the weight of a solution are defined externally in C++, we do not need to modify CLASP-W to compute weighted solutions in other domains like planning; we just need to implement the heuristic function in a separate file.

- Based on the divide-and-conquer approach based on computing weighted phylogenies, we have implemented a fully automated system (called PHYLO-ASP) to reconstruct and analyze phylogenies, utilizing CLASP-W. We have also implemented a system called PHYLONET-ASP for reconstructing weighted phylogenetic networks. There is no such phylogenetic system which can help experts to order phylogenies with respect to a plausibility measure that includes also some domain-specific information.

- We have shown the applicability of our methods on two different real datasets: The first one is Indo-European languages, and the second one is the genus *Quercus* (oak trees). In our experiments, we have computed weighted phylogenies with PHYLOASP and weighted phylogenetic networks with PHYLONET-

ASP for these datasets. For Indo-European languages dataset, we have computed 18 phylogenies, all of which are plausible from Don Ringe's point of view. In addition, the phylogeny with the minimum number of incompatible characters and maximum weight is found to be the most plausible phylogeny. For *Quercus* species, we have computed 30 phylogenies which are identified as plausible by Yasin Bakış. In this dataset, multiple phylogenies are identified as the most plausible ones. These most plausible phylogenies also have the minimum number of incompatible characters and maximum weight. All the computed phylogenies being identified as plausible indicates the effectiveness and correctness of our system and our methods. In addition, these test results showed the efficiency and accuracy of the weight measures we have formulated since the maximum weighted trees are the most plausible ones among the others.

- To apply our method to real datasets, we have defined new weight measures for phylogenies and phylogenetic networks. We have defined 2 domain-dependent and 2 domain-independent new weight measures for phylogenetic trees and one new domain-independent weight measure for phylogenetic networks. In order to use them with our second method above, we have defined an admissible heuristic function for each of them, and we have proved their admissibility (Propositions 3, 4, 5 and 6).

Some of our contributions are summarized in the following papers. [14], [15] and [16] introduce the representation-based methods and search-based methods for computing weighted solutions in ASP and shows their applicability and effectiveness on the weighted reconstruction problem for Indo European languages. [17] presents PHYLO-ASP system and [13] presents its underlying divide-and-conquer mechanism.

Future Work Recall that our methods for reconstructing phylogenies in this thesis are based on the compatibility criterion. It may be interesting to extend our methods by considering different criteria such as maximum parsimony. Computing phylogenies with ASP using maximum parsimony criteria is reasonable, since the problem still remains not tractable. This extension can be achieved by introducing new ASP formulations and new weight measures, and by modifying the preprocessing algorithms. This extension can be useful for comparing different criterion and comparing the representation-based method with the search-based method.

Our experiments can be extended to different domains, such as Turkic languages. Moreover, such an extension may improve our weight functions. On the other hand, these experiments can benefit the studies on Turkic languages.

Our search-based method for computing weighted solutions in ASP is based on branch and bound. It may be useful to propagate some literals as the search is "bounded". Such a modification may lead to a more efficient solver for computing weighted solutions in ASP.

References

[1] Mario Alviano, Wolfgang Faber, and Nicola Leone. Disjunctive asp with functions: Decidable queries and effective computation. *TPLP*, 10(4-6):497–512, 2010.

[2] Abdullah N. Arslan and Peyman Bizargity. Phylogeny by top down clustering using a given multiple alignment. In *BIBE*, pages 809–814, 2007.

[3] Yasin Bakiş. *Morphometric Analysis of Oak (Quercus L.) Acorns in Turkey.* PhD thesis, Abant Izzet Baysal University, 2005.

[4] Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with a-prolog. *CoRR*, cs.AI/0312040, 2003.

[5] Chitta Baral, Michael Gelfond, and J. Nelson Rushton. Probabilistic reasoning with answer sets. *TPLP*, 9(1):57–144, 2009.

[6] Chitta Baral and Cenk Uyan. Declarative specification and solution of combinatorial auctions using logic programming. In *LPNMR*, pages 186–199, 2001.

[7] F. Barbancon, T. Warnow, D. Ringe, S. Evans, , and L. Nakhleh. An experimental study comparing linguistic phylogenetic reconstruction methods. In *Proceedings of the conference Languages and Genes*, To appear,2009.

[8] Hans L. Bodlaender, Mike R. Fellows, and Tandy J. Warnow. Two strikes against perfect phylogeny. In *Proc. of 19th International Colloquidum on Automata Languages and Programming*, pages 273–283. Springer-Verlag, 1992.

[9] Georg Boenn, Martin Brain, Marina De Vos, and John Fitch. Anton: Composing logic and logic composing. In *LPNMR*, pages 542–547, 2009.

[10] Georg Boenn, Martin Brain, Marina De Vos, and John Fitch. Automatic music composition using answer set programming. *CoRR*, abs/1006.4948, 2010.

[11] Gerhard Brewka. Preferences, contexts and answer sets. In *ICLP*, page 22, 2007.

[12] Daniel R. Brooks, Esra Erdem, Selim T. Erdoğan, James W. Minett, and Don Ringe. Inferring phylogenetic trees using answer set programming. *J. Autom. Reason.*, 39(4):471–511, 2007.

[13] Duygu Cakmak and Esra Erdem. Reconstructing large phylogenies using answer set programming. 2010. in progress.

[14] Duygu Cakmak, Esra Erdem, and Halit Erdogan. Computing weighted solutions in answer set programming. In *LPNMR*, pages 416–422, 2009.

[15] Duygu Cakmak, Esra Erdem, and Halit Erdogan. Computing weighted solutions in asp: Representation-based method vs. search-based method. In *Proc. of the 17th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'10)*, 2010.

[16] Duygu Cakmak, Esra Erdem, and Halit Erdogan. Computing weighted solutions in asp: Representation-based method vs. search-based method. 2010. Submitted to Annals of Mathematics and Artificial Intelligence (AMAI) Journal.

[17] Duygu Cakmak, Esra Erdem, and Halit Erdogan. Phylo-asp: Phylogenetic systematics with answer set programming. 2010. in progress.

[18] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Regular path queries in expressive description logics with nominals. In *IJCAI*, pages 714–720, 2009.

[19] Joseph H. Camin and Robert R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19:311–326, 1965.

[20] Keith L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.

[21] J. P. Davis, S. Akella, and P. H. Waddell. Accelerating phylogenetics computing on the desktop: experiments with executing upgma in programmable logic. In *In Engineering in Medicine and Biology Society (IEMBS*, pages 2864–2868, 2004.

[22] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[23] James P. Delgrande, Torsten Grote, and Aaron Hunter. A general approach to the verification of cryptographic protocols using answer set programming. In *LPNMR*, pages 355–367, 2009.

[24] Jürgen Dix, Thomas Eiter, Michael Fink, Axel Polleres, and Yingqian Zhang. Monitoring agents using declarative planning, 2003.

[25] Jürgen Dix, Ugur Kuter, and Dana Nau. Planning in answer set programming using ordered task decomposition. In *KI 2003 (German National Conference on Artificial Intelligence*, pages 490–504. Springer, 2003.

[26] Deborah East and Miroslaw Truszczynski. More on wire routing with asp. In *Answer Set Programming*, 2001.

[27] A. W. F. Edwards and L. L. Cavalli-Sforza. Reconstruction of evolutionary trees. *Phenetic and Phylogenetic Classification*, 1964.

[28] Thomas Eiter, Gerhard Brewka, Minh Dao-Tran, Michael Fink, Giovambattista Ianni, and Thomas Krennwallner. Combining nonmonotonic knowledge bases with external sources. In *FroCos*, pages 18–42, 2009.

[29] Thomas Eiter, Esra Erdem, Halit Erdogan, and Michael Fink. Finding similar or diverse solutions in answer set programming. In *ICLP*, pages 342–356, 2009.

[30] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[31] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. The dlvk planning system: Progress report. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 541–544, London, UK, 2002. Springer-Verlag.

[32] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded semantics for description logic programs in the semantic web. In *RuleML*, pages 81–97, 2004.

[33] Esra Erdem. Phylo-asp: Phylogenetic systematics with answer set programming. In *LPNMR*, pages 567–572, 2009.

[34] Esra Erdem, Ozan Erdem, and Ferhan Türe. Haplo-asp: Haplotype inference using answer set programming. In *LPNMR*, pages 573–578, 2009.

[35] Esra Erdem, Vladimir Lifschitz, and Don Ringe. Temporal phylogenetic networks and logic programming. *TPLP*, 6:539–558, 2006.

[36] Esra Erdem, Vladimir Lifschitz, and Martin F. Wong. Wire routing and satisfiability planning. In *In Proceedings CL-2000*, pages 822–836. Springer-Verlag. LNCS, 2000.

[37] Jason Evans, Luke Sheneman, and James Foster. Relaxed neighbor joining: a fast distance-based phylogenetic tree construction method. 62, 2006.

[38] Wolfgang Faber, Gerald Pfeifer, Nicola Leone, Tina Dell'Armi, and Giuseppe Ielpa. Design and implementation of aggregate functions in the dlv system. *TPLP*, 8(5-6):545–580, 2008.

[39] Raphael Finkel, Victor W. Marek, and Miroslaw Truszczynski. Constraint lingo: A program for solving logic puzzles and other tabular constraint problems, 2002.

[40] M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming.*

[41] Martin Gebser, Carito Guziolowski, Mihail Ivanchev, Torsten Schaub, Anne Siegel, Sven Thiele, and Philippe Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In *KR*, 2010.

[42] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, pages 260–265. Springer, 2007.

[43] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set enumeration. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 136–148. Springer, 2007.

[44] Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *Proc. of IJCAI*, pages 386–392. AAAI Press/MIT Press, 2007.

[45] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceeding of the Fifth Logic Programming Symposium*, pages 1070–1080, 1988.

[46] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[47] Stéphane Guindon and Olivier Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic biology*, 52(5):696–704, October 2003.

[48] Keijo Heljanko and Ilkka Niemelä. Bounded ltl model checking with stable models. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 200–212. Springer-Verlag, 2003.

[49] Willi Hennig. Grundzüge einer theorie der phylogenetischen systematik. *Deutscher Zentralverlag*, 1950.

[50] Willi Hennig. Phylogenetic systematics. *Annu. Rev. Entomol.*, 10:97–116, 1965.

[51] Willi Hennig. Phylogenetic systematics. *University of Illinois Press*, 1966.

[52] Katsumi Inoue and Chiaki Sakama. Abductive framework for nonmonotonic theory change. In *IJCAI*, pages 204–210, 1995.

[53] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[54] H. A. Khan, I. A. Arif, A. H. Bahkali, A. H. Al Farhan, and A. A. Al Homaidan. Bayesian, maximum parsimony and upgma models for inferring the phylogenies of antelopes using mitochondrial markers. *Evolutionary bioinformatics online*, 4:263–270, 2008.

[55] Nicola Leone, Gianluigi Greco, Giovambattista Ianni, Vincenzino Lio, Giorgio Terracina, Thomas Eiter, Wolfgang Faber, Michael Fink, Georg Gottlob, Riccardo Rosati, Domenico Lembo, Maurizio Lenzerini, Marco Ruzzi, Edyta Kalka, Bartosz Nowicki, and Witold Staniszkis. The infomix system for advanced integration of incomplete and inconsistent data. In *SIGMOD Conference*, pages 915–917, 2005.

[56] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[57] Vladimir Lifschitz. What is answer set programming?. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 1594–1597. AAAI Press, 2008.

[58] Fangzhen Lin and Yuting Zhao. Assat: computing answer sets of a logic program by sat solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.

[59] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *In The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[60] Joaäo P. Marques-silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[61] Alessandra Mileo, Davide Merico, and Roberto Bisiani. Wireless sensor networks supporting context-aware reasoning in assisted living. In *PETRA '08:*

Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments, pages 1–2, New York, NY, USA, 2008. ACM.

[62] Alessandra Mileo, Davide Merico, and Roberto Bisiani. Non-monotonic reasoning supporting wireless sensor networks for intelligent monitoring: The sindi system. In *LPNMR*, pages 585–590, 2009.

[63] Saitou N. and M.Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. 4:406–425, 1987.

[64] Luay Nakhleh. *Phylogenetic Networks.* PhD thesis, The university of Texas at Austin, 2004.

[65] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[66] Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics of weight constraint rules. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 1999.

[67] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An a-prolog decision support system for the space shuttle. In *In PADL 2001*, pages 169–183. Springer, 2001.

[68] Jean-Michel Richer, Adrien Goëffon, and Jin-Kao Hao. A memetic algorithm for phylogenetic reconstruction with maximum parsimony. In *EvoBIO '09: Proceedings of the 7th European Conference on Evolutionary Computation,*

Machine Learning and Data Mining in Bioinformatics, pages 164–175, Berlin, Heidelberg, 2009. Springer-Verlag.

[69] D. Ringe, Tandy Warnow, and A. Taylor. Indo-european and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002.

[70] Chiaki Sakama. Learning by answer sets. In Alessandro Provetti and Tran Cao Son, editors, *In AAAI Spring Symposium:Answer Set Programming*, 2001.

[71] Torsten Schaub and Sven Thiele. Metabolic network expansion with answer set programming. In *ICLP*, pages 312–326, 2009.

[72] Torsten Schaub and Kewen Wang. A comparative study of logic programs with preference. In Bernhard Nebel, editor, *IJCAI*, pages 597–602. Morgan Kaufmann, 2001.

[73] Patrik Simons and Timo Soininen. Stable model semantics of weight constraint rules. In *Proc. of LPNMR*, pages 317–331. Springer-Verlag, 1999.

[74] Martin Simonsen, Thomas Mailund, and Christian N. Pedersen. Rapid neighbour-joining. In *WABI '08: Proceedings of the 8th international workshop on Algorithms in Bioinformatics*, pages 113–122, Berlin, Heidelberg, 2008. Springer-Verlag.

[75] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[76] Tran Cao Son and Enrico Pontelli. A constructive semantic characterization of aggregates in answer set programming. *TPLP*, 7(3):355–375, 2007.

[77] Tran Cao Son, Enrico Pontelli, and Chiaki Sakama. Logic programming for multiagent planning with negotiation. In *ICLP*, pages 99–114, 2009.

[78] Tran Cao Son and Chiaki Sakama. Reasoning and planning with cooperative actions for multiagents using answer set programming. In *DALT*, pages 208–227, 2009.

[79] Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors. *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*. Springer, 2009.

[80] Nam Tran and Chitta Baral. Reasoning about triggered actions in ansprolog and its application to molecular interactions in cells. In *KR*, pages 554–564, 2004.

[81] Ferhan Türe and Esra Erdem. Efficient haplotype inference with answer set programming. In *AAAI*, pages 1834–1835, 2008.

[82] Marina De Vos, Tom Crick, Julian Padget, Martin Brain, Owen Cliffe, and Jonathan Needham. A multi-agent platform using ordered choice logic programming. In *In Declarative Agent Languages and Technologies (DALT'05)*, pages 72–88, 2005.

[83] Marina De Vos and Dirk Vermeir. Logic programming agents and game theory, 2001.

[84] Marina De Vos and Dirk Vermeir. Extending answer sets for logic programming agents. *Ann. Math. Artif. Intell.*, 42(1-3):103–139, 2004.

[85] Harold Todd Wareham. On the computational complexity of inferring phylogenetic trees. Technical report, Department of Computer Science, Memorial University of Newfoundland, 1993–2002.

[86] Tiffani L. Williams and Bernard M. E. Moret. An investigation of phylogenetic likelihood methods. pages 79–86, 2003.

[87] Ziheng Yang. Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: Approximate methods. *J. Mol. Evol*, 39:39–306, 1994.