Computing Weighted Solutions in ASP: Representation-Based Method vs. Search-Based Method

Duygu Çakmak, Halit Erdoğan, and Esra Erdem

Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul, Turkey

Abstract

For some problems with many solutions, like planning and phylogeny reconstruction, one way to compute more desirable solutions is to assign weights to solutions, and then pick the ones whose weights are over (resp. below) a threshold. This paper studies computing weighted solutions to such problems in Answer Set Programming. We investigate two sorts of methods for computing weighted solutions: one suggests modifying the representation of the problem and the other suggests modifying the search procedure of the answer set solver. We show the applicability and the effectiveness of these methods in reconstructing weighted phylogenies for Indo-European languages. We also compare these methods in terms of computational efficiency; the experimental results show that the search-based method is better.

1 Introduction

In Answer Set Programming (ASP) [10], a computational problem is described as an ASP program whose answer sets correspond to solutions, and answer sets for this program are computed using answer set solvers. Some problems, like planning and phylogeny reconstruction, 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. For such problems, one way to compute more desirable solutions is to assign weights to solutions, and then pick the distinct solutions whose weights are over (resp. below) a threshold. For example, in a planning problem, we can define the weight of a plan in terms of the costs of actions (or action sequences), and then compute the distinct plans whose weights are less than a given value. In puzzle generation, we can define the weight of a puzzle instance by means of some difficulty measure, and then generate difficult puzzles whose weights are over a given value. Motivated by such applications, we study the problem of computing weighted solutions in ASP and show the applicability of our approach in phylogeny reconstruction (i.e., computing leaf-labeled trees, called phylogenies, to model the evolutionary history of a set of species).

We study two sorts of methods for computing weighted solutions and compare them experimentally:

Representation-based methods The idea is to modify the representation of the problem, to compute weighted solutions. In some cases, we do not have to define the weight of a solution explicitly; we can use aggregates (e.g.,

Proceedings of the 17th International RCRA workshop (RCRA 2010):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Bologna, Italy, June 10–11, 2010

sum, count, times) to compute the weight, in the sense of [14, 8, 15]. Some other problems require an explicit ASP definition of the weight of a solution. Phylogeny reconstruction problems we consider are in the latter group: the weight of a phylogeny does not only depend on the weights of some parts of the phylogeny but also some domain-specific information.

Search-based methods These methods do not modify the ASP representation of the problem, but define the weight function externally (e.g., as a C++ program) and modify the search algorithm of the answer set solver to compute solutions over (resp. below) a given threshold. There is no answer set solver that can compute weighted solutions in such a way. In our studies, we consider the solver CLASP [9], and modify its search algorithm to implement the suggested method.

We apply these methods to phylogeny reconstruction. Reconstructing phylogenies for a given set of taxonomic units is important for various research such as historical linguistics, zoology, anthropology, archeology, etc.. For example, a phylogeny of languages may help scientists to better understand human migrations [16]. For a given set of taxonomic units, some existing phylogenetic systems, like that of [1], generate more than one phylogeny that explains the evolutionary relationships between the given taxonomic units. In such cases, to pick the most plausible ones, an expert needs to go over these phylogenies manually and in detail. There is no phylogenetic system that can help experts to order phylogenies with respect to a desirability/plausibility measure that includes also some domain-specific information. In this study, we define the weight of a phylogeny for the family of Indo-European languages studied in [1], in such a way as to reflect the plausibility criterion provided by the historical linguistics. Using this weight function, we show the applicability and effectiveness of the methods above (for computing weighted solutions) in reconstructing plausible phylogenies for Indo-European languages, and compare them in terms of computational efficiency.

To summarize, the main contributions of this paper are as follows:

- We precisely describe the decision/optimization problems for computing weighted solutions in ASP (Section 2).
- We introduce a search-based method for computing weighted solutions, and implement it by modifying the search algorithm of the answer set solver CLASP (Section 3); the ASP representation of the problem is not modified. The new algorithm is called CLASP-W. In this method, the weight of a solution is defined in C++ in a separate file; this allows us to use CLASP-W to compute weighted solutions to various problems.
- We also describe a representation-based method for computing weighted solutions, where the weight of a solution is defined as an ASP program (Section 3). With this method, we can use an existing ASP solver as is.

- We compare the representation-based method and the search-based method in terms of computational efficiency (e.g., computation time and memory consumed) (Section 7), for reconstructing weighted phylogenies for Indo-European languages [1] (Section 5). Experimental results show that the search-based method is better than the representation-based method.

- Extending the applicability of the methods of [6] for computing similar (diverse) solutions, we describe the problem of computing diverse/similar weighted solutions in ASP, introduce a search-based method to compute these solutions, and implement it by modifying the search algorithm of CLASP (Section 6). The new algorithm is called CLASP-NKW.

- While evaluating the methods above experimentally in reconstructing phylogenies for Indo-European languages, we also illustrate their effectiveness in reconstructing plausible phylogenies (Section 5). For that, we introduce a weight function that takes into account the compatibility criterion for reconstructing phylogenies as well as some domain-specific information provided by the historical linguists (Section 4). The experimental results illustrate the usefulness of our methods for phylogenetics.

This paper extends [3] by a detailed discussion on the representation-based method and the search-based method for computing weighted solutions as well as a discussion on computing similar/diverse weighted solutions. It also extends the discussion of experimental results.

2 Weighted Solutions

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

> AT LEAST (resp. AT MOST) w-WEIGHTED SOLUTION: Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a weight measure ω that maps a solution for P to a nonnegative integer, and a nonnegative integer w, decide whether a solution S exists for P such that $\omega(S) \geq w$ (resp. $\omega(S) \leq w$).

For instance, suppose that $\mathcal{P}$ describes the phylogeny reconstruction problem for Indo-European languages, and that ω describes the total weight of the characters compatible with the to-be-constructed phylogeny and takes into account some domain-specific information. Then finding phylogenies whose weights are at least 45 is an instance of the problem above.

Suppose that the ASP program $\mathcal{P}$ is a propositional normal logic program and deciding $\omega(S) \leq w$ (resp. $\omega(S) \geq w$) for a given w is in NP. Then, we can conclude that

Theorem 1. AT LEAST *(resp.* AT MOST*)* w-WEIGHTED SOLUTION *is NP-complete.*

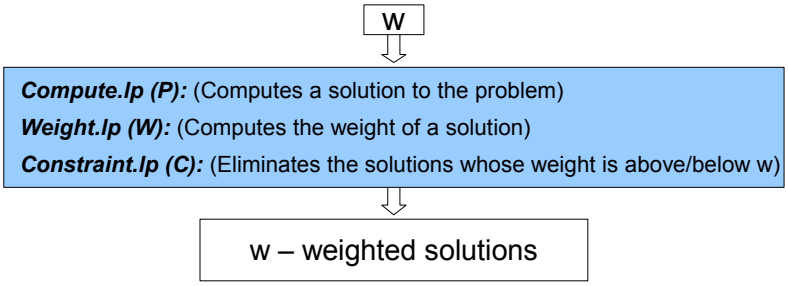

Figure 1: Representation-based method: computing at most/least w-weighted solutions.

Membership follows from the fact that we can not only guess a candidate solution S but also a witness for $\omega(S) \leq w$ (resp. $\omega(S) \geq w$), and check in polynomial time whether $\omega(S) \leq w$ (resp. $\omega(S) \geq w$). For hardness, we can reduce the answer-set existence for normal propositional programs, which is an NP-complete problem, to this problem.

3 Computing Weighted Solutions

We study two sorts of methods, representation-based and search-based, for computing at least/most w-weighted solutions of a given problem P, given an ASP program $\mathcal{P}$, a weight measure ω that maps a solution to a nonnegative integer, and a nonnegative integer w.

The idea behind the representation-based methods is to modify the representation of the problem, to compute weighted solutions. In some cases, we do not have to define the weight of a solution explicitly; we can use aggregates (e.g., sum, count, times) to compute the weight, in the sense of [14, 8, 15]. Some other problems require an explicit definition of the weight of a solution. Phylogeny reconstruction problems we consider are in the latter group: the weight of a phylogeny does not only depend on the weights of some parts of the phylogeny but also some domain-specific information. Therefore, we consider representation-based methods (as outlined in Figure 1) that formulate the weight function ω as an ASP program $\mathcal{W}$, the constraints on the weights of the solutions also as an ASP program C, and then compute at least (resp. most) w-weighted solutions by computing answer sets for the ASP program $\mathcal{P} \cup \mathcal{W} \cup \mathcal{C}$.

Search-based methods (as outlined in Figure 2) on the other hand do not modify the ASP representation of the problem, but define the weight function externally (e.g., as a C++ program) and modify the search algorithm of the answer set solver to compute solutions over (resp. below) a given threshold. There is no answer set solver that can compute weighted solutions in such a way.

In our studies, we consider the solver CLASP [9], and modify its search algorithm to implement the search-based method above. Before describing these mod-

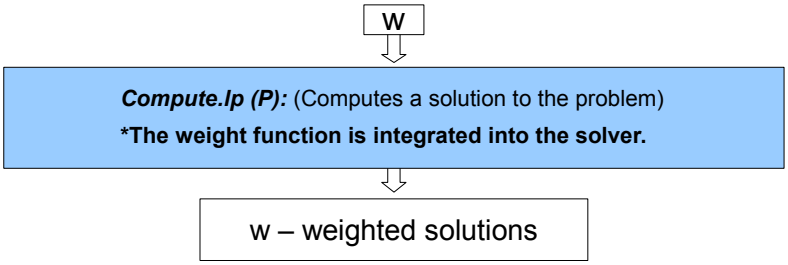

Figure 2: Search-based method: computing at most/least w-weighted solutions.

Algorithm 1 CLASP

Input: An ASP program Π
Output: An answer set A for Π

```
A ← ∅    // current assignment of literals
▽ ← ∅    // set of conflicts
while No Answer Set Found do
  // propagate according to the current assignment and conflicts; update the current assignment
  PROPAGATION(Π, A, ▽)
  if There is a conflict in the current assignment then
    RESOLVE-CONFLICT(Π, A, ▽)    // learn and update the conflict set and do backtracking
  else
    if Current assignment does not yield an answer set then
      SELECT(Π, A, ▽)    // select a literal to continue search
    else
      return A
    end if
  end if
end while
```

ifications, let us describe first CLASP's algorithm. CLASP does a conflict-driven DPLL-like [4] branch and bound search to find an answer set for the program: at each level, it does propagation followed by backtracking or selection of new literals according to the current conflicts. A rough working principle of CLASP is shown in Algorithm 1. As can be seen, CLASP goes through three main steps to find an answer set. In the PROPAGATION step, it decides the literals that have to be included in the answer set due to the current assignment and conflicts. In the RESOLVE-CONFLICT step, it tries to resolve the conflicts encountered in the previous step. If there is a conflict, then CLASP learns it and does backtracking to an appropriate level. If there are no conflicts and the currently selected literals do not represent an answer set, in SELECT, CLASP selects a new literal (based on some heuristics) to continue search.

We modify the CLASP's algorithm (Algorithm 1) to compute at least (resp. at most) w-weighted solutions, as shown in Algorithm 2; the parts in red denote these modifications. The new algorithm is called CLASP-W. Note that compared to CLASP, CLASP-W has an additional procedure called WEIGHT-ANALYZE; therefore, in order to use CLASP-W, we need to implement the WEIGHT-ANALYZE

Algorithm 2 CLASP-W

Input: An ASP program Π and a nonnegative integer w
Output: An answer set for Π, that describes an at least (resp. at most) w-weighted solution

```
A ← ∅    // current assignment of literals
▽ ← ∅    // set of conflicts
while A does not represent an answer set do
    // propagate according to the current assignment and conflicts;update the current assignment
    PROPAGATION(Π, A, ▽)
    // compute an upper (resp. lower) bound for the weight of a solution that contains A
    weight ← 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 propagation OR weight < w then
        RESOLVE-CONFLICT (Π, A, ▽)    // learn and update the conflict set and do backtracking
    end if
    if Current assignment does not yield an answer set then
        SELECT(Π, A, ▽)    // select a literal to continue search
    else
        return A
    end if
end while
return false
```

function according to the weight measure of the specific domain.

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 (resp. smaller) than w. Therefore; we set the current assignment of literals as conflict in that case. After setting an assignment as conflict, CLASP learns that assignment and does backtracking and never selects those assignment in the further stages of the search.

4 Weighted Phylogeny Reconstruction Problem

The evolutionary relations between species (or "taxonomic units") based on their shared traits can be modeled as a phylogeny—a tree whose leaves represent the species, internal vertices represent their ancestors and edges in between represent the relationships between them. The problem of phylogeny reconstruction asks for "plausible" phylogenies for a given set of taxonomic units. The plausibility of phylogenies can be evaluated with respect to domain-specific information (e.g., biological evidence, archeological evidence, historical linguistics) provided by the experts.

There have been various studies to compute plausible phylogenies (cf. [1]). We consider a character-based cladistics approach with respect to the compatibility criterion, as in [1]. In this approach, we describe each taxonomic unit with a set of "characters"—traits that every taxonomic unit can instantiate in a variety of ways. The taxonomic units that instantiate the character in the same way are assigned the same "state" of that character. Once we describe the state of every character for every taxonomic unit, we can reconstruct phylogenies using the "compatibility" criterion: our goal is to find a phylogeny with a small number of incompatible characters. The problem of reconstructing a phylogeny with at most k incompatible characters (let us call this problem as k-CP) is NP-hard [5].

While reconstructing phylogenies, some characters may give more information than the others. For instance, to model the evolutionary history of a family of languages, morphological/phonological characters are more informative than lexical characters. In order to emphasize the role of such characters in reconstructing a phylogeny, we define the concept of a weighted phylogeny.

Before we define weighted phylogenies, let us define a phylogeny. A *phylogeny* for a set of taxonomic units 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 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.

A *weighted phylogeny* is a phylogeny (V, E, L, I, S, f) along with a weight function Φ that maps every character $i \in I$ to a nonnegative integer. The *weight of a phylogeny* (V, E, L, I, S, f) can be defined in various ways with respect to Φ; in the following, we consider the weight of a phylogeny as the sum of the weights of all characters that are compatible with that phylogeny.

We are interested in computing weighted phylogenies for a given set of taxonomic units, and thus the following decision problem:

> AT LEAST w-WEIGHTED k-COMPATIBLE PHYLOGENIES (wk-WCP): Given three sets L, I, S, a function f from $L \times I$ to S, a function Φ from I to non-

negative integers, and two nonnegative integers w and k, decide the existence of a phylogeny (V, E, L, I, S, f) with at most k incompatible characters and whose weight is at least w.

Note that wk-WCP is NP hard: we can reduce k-CP to wk-WCP by taking $\Phi(i) = 1$ for every $i \in I$.

5 Computing Weighted Phylogenies for Indo-European Languages

We can compute weighted phylogenies for the family of Indo-European languages described in [1], using the representation-based method or the search-based method described in Section 3. Before we show the applicability of these methods, let us define the weight Φ of a character for languages, and the weight of a phylogeny (V, E, L, I, S, f) for Indo-European languages.

5.1 Weight of a Phylogeny for Indo-European

As discussed in [1], out of all given characters and character states, only informative characters[1] and their essential states[2] play a role in reconstructing phylogenies. Therefore, we consider informative characters and their essential states only while reconstructing phylogenies for Indo-European languages. We have observed that some informative characters have less number of essential states whereas some have many; the ones with many essential states give more information as to how the languages are related to each other. Based on this domain-specific information, we define the weight Φ of an informative character $i \in I$ with respect to a phylogeny (V, E, L, I, S, f), as the number of languages that are mapped to an essential state for that character:

$$\Phi_{L,I,S,f}(i) = |\{l \in L : f(i, l, s) = s, i \text{ is informative, } s \text{ is essential}\}| \quad (1)$$

In the following, we drop the subscript from $\Phi_{L,I,S,f}$ when L, I, S, f are clear from the context.

We can define the weight of a phylogeny (V, E, L, I, S, f) for Indo-European languages simply as the sum of the weights of compatible characters with that phylogeny:

$$\textit{weight}(V, E, L, I, S, f) = \sum_{i \in I, i \text{ is informative}} \Phi(i) \quad (2)$$

To get more plausible phylogenies, we also incorporate further domain-specific information. It is told us by historical linguist Don Ringe that it is least likely that Greco-Armenian (GA) languages are siblings with Balto-Slavic (BS) languages.

[1] A character is *informative* if it has at least 2 essential states.

[2] 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 $f(l_1, j) = f(l_2, j) = s$.

Similarly, but not as least likely as GA and BS, is the grouping of GA with Germanic (GE) languages. If the to-be-reconstructed phylogenies have such odd groupings of languages, we reduce some amount from the total weight of the phylogeny making sure that the weight of a phylogeny is not negative.

5.2 The Representation-Based Method

The representation-based method (Section 3, Figure 1) suggests describing the weight of a phylogeny as an ASP program W, the weight constraints as an ASP program C, and then compute weighted phylogenies by means of computing answer sets for the program $P \cup W \cup C$ where P is a program that describes phylogeny reconstruction. We describe ASP programs in the language of LPARSE [13]. We take P as the phylogeny program of [1].

We describe the weight of a phylogeny (defined in the previous section) as an ASP program W in four parts, as follows. Suppose that `PW, W` denote phylogeny weights, `IC` denotes an incompatible character, `CW` denotes a character weight, and `C` denotes a character. First we describe the weight `CW` of an informative character `IC` as in (1):

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

In the second part, first we define the sum of the weights of characters compatible with the to-be-reconstructed phylogeny, as in (2):

```
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).
```

Observe in the third and the fifth rules that the incompatible characters do not contribute to the weight of a phylogeny.

In the third part, we take into account the domain-specific information discussed in the previous section. For instance, it is least likely that Greco-Armenian (GA) languages be siblings with Balto-Slavic (BS) languages. Therefore, we decrease some amount, say `30`, from the weight of a phylogeny if it groups GA and BS as siblings. For Indo-European, historical linguist Don Ringe identified 4 odd groupings. We define the sum `W` of deductions due to all these odd groupings:

```
totalDeductions(W) :- addDeductions(W,4).
addDeductions(W,1) :- oddGroup(1,W).
addDeductions(W+PW,G+1) :- oddGroup(G+1,PW), addDeductions(W,G).
```

Finally, in the fourth part, we define the weight of a phylogeny by subtracting the total deductions of odd groupings from the total weights of characters:

```
weightOfPhylogeny(PW-W) :- totalWeightOfChars(PW),
  totalDeductions(W).
```

We describe the weight constraints, to ensure that the weight of the phylogeny is larger than w, as another ASP program C as follows:

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

5.3 The Search-Based Method

The search-based method (Section 3, Fig 2) suggests defining the weight measure in C++ (in a separate file) and use it with CLASP-W (the modified version of CLASP that computes weighted solutions).

As discussed in Section 3, we need to define and implement an admissible heuristic function to estimate the upper bound for the weight of a phylogeny from a partial phylogeny. Since the weight of a phylogeny is defined over the weights of incompatible characters, we can define the heuristic function as follows with respect to the set I of characters and the partially constructed answer set A:

$$UB(A, I) = \sum_{i \in I} \Phi(i) - \sum_{incompatible(i) \in A} \Phi(i)$$

UB is admissible because $UB(A, I)$ is greater than or equal to the exact upper bound weight (2) for the phylogeny to-be-reconstructed. Note that building an admissible heuristic function depends on the characteristics of the domain specific weight measure.

6 Computing Similar/Diverse Weighted Solutions

Our work is closely related to [6], which introduces methods for computing similar (resp. diverse) solutions, in that they both introduce methods and tools for analyzing and comparing solutions. In [6], the authors propose defining a distance function to measure the similarity/diversity of solutions, and present a method for integrating this distance measure in CLASP to compute n k-similar (resp. diverse) solutions (i.e., n solutions whose distance is less (resp. greater) than or equal to k). They call this modified version of CLASP as CLASP-NK.

We extend the applicability of our methods and the methods of [6], by considering more general problems as follows:

> n k-SIMILAR (resp. k-DIVERSE) AT LEAST (resp. AT MOST) w-WEIGHTED SOLUTIONS: Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a weight measure ω that maps a solution for P to a nonnegative integer, a distance measure Δ that maps a set of solutions to a nonnegative integer, nonnegative integers w and k, decide whether a set S of n solutions for P exists such that $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$) and for each $s \in S$, $\omega(s) \geq w$ (resp. $\omega(s) \leq w$).

Algorithm 3 CLASP-NKW

Input: An ASP program Π and nonnegative integers w, n and k
Output: A set of n k-similar at least w-weighted solutions

```
A ← ∅    // current assignment of literals
▽ ← ∅    // set of conflicts
X ← ∅    // previously computed answer sets
while |X| < n do
    PROPAGATION(Π, A, ▽)
    weight ← WEIGHT-ANALYZE(A)    // Related to CLASP-W
    distance ← DISTANCE-ANALYZE(A, X)    // Related to CLASP-NK
    if (There is a conflict in propagation) OR (weight < w) OR (distance > k) then
        RESOLVE-CONFLICT (Π, A, ▽)
    end if
    if Current assignment does not yield an answer set then
        SELECT(Π, A, ▽)
    else
        return X ← X ∪ A
    end if
end while
return X
```

We introduce a search-based method for computing n k-similar (resp. k-diverse) at least (resp. at most) w-weighted solutions in ASP, and implement this method by modifying the search algorithm CLASP-W as in Algorithm 3; the parts in blue denote these modifications. The modified version is called CLASP-NKW.

Note that AT LEAST (resp. AT MOST) w-WEIGHTED SOLUTION can be solved using CLASP-NK of [6] by defining the distance function in terms of the given weight function ω. However, in general, we cannot compute n k-similar (resp. diverse) at least (resp. most) w-weighted solutions in this way. To be able to solve all four variations of n k-SIMILAR (resp. k-DIVERSE) AT LEAST (resp. AT MOST) w-WEIGHTED SOLUTIONS using CLASP-NK, we need to define four different distance functions (rather than a generic distance function) in terms of the given distance function Δ and the given weight function ω.

7 Experimental Results

We applied the computational methods described above (i.e., the representation-based method, and the search-based method) to reconstruct weighted phylogenies for Indo-European languages. We used the dataset assembled by Don Ringe and Ann Taylor [12]. While computing phylogenies, we also took into account some domain-specific information.

Let us first consider computing an at least w-weighted phylogeny with at most c incompatible characters (Table 1). Based on the results of [1], we consider $c = 16, 17, 18$ and $w = 45$. For each problem, for each method, we present the computation time, the size of the ground program, and the size of the mem-

ory used in computation.[3] For instance, let us consider computing a phylogeny with at most 17 incompatible characters, and whose weight is at least 45. With the representation-method, CLASP takes 15.32 CPU sec.s to compute such a phylogeny; the ground program has 79229 atoms and 1585419 rules; the computation of the phylogeny consumes 369 MB of memory. On the other hand, with the search-based method, CLASP-W takes 1.30 CPU sec.s to compute such a phylogeny; the ground program has 3744 atoms and 55219 rules; the computation of the phylogeny consumes 22 MB of memory.

Observe in Table 1 that in terms of both computation time and the memory used, the search-based method performs better than the representation-method. These results conform with our expectations. The representation-based method explicitly defines the weight function, and thus the program/memory size is larger. The search-based method deals with the time consuming computation of weights of phylogenies, not at the representation level but at the search level: It does not require an ASP representation of the weight function but requires a modification of the solver to guide it find a plausible phylogeny. Therefore, in the search-based methods, the program/memory size is smaller as well as the computation time.

Now let us examine the results of experiments, while computing all at least w-weighted phylogenies with at most c incompatible characters (Table 2): In terms of both computation time and the memory used, the search-based method performs better than the representation-method. For instance, let us consider the problem of computing all phylogenies with at most 17 incompatible characters, and whose weight is at least 45. With the representation-method, CLASP takes 126 CPU sec.s to compute all 8 phylogenies; whereas, with the search-based method, CLASP-W takes 17 CPU sec.s to compute them.

In [1], after computing all 45 phylogenies, the authors examine them manually and identify 34 of them as plausible (1 with 16 incompatible characters, 7 with 17 incompatible characters, 6 with 18 incompatible characters, 3 with 19 incompatible characters, 17 with 20 incompatible characters). With the weight measure defined in Section 5, and the representation/search-based methods described above for computing weighted phylogenies, we could automatically compute all plausible phylogenies with at most 18 incompatible characters.

We also applied our search-based method for computing similar/diverse phylogenies for Indo-European languages, using CLASP-NKW. We considered the nodal distance (as in [6]) to measure the similarity/diversity of a set of phylogenies. For instance, we computed 2 12-similar and at least 45-weighted phylogenies using CLASP-NKW; the computation took 2.3 CPU sec.s. Likewise, we computed 2 24-diverse and at least 45-weighted phylogenies using CLASP-NKW; the computation took 2.7 CPU sec.s. In [1], after examining all 34 plausible phylogenies manually, the authors come up with three forms of tree structures, and then "filter" the phylogenies accordingly. The results of our experiments comply with these

[3] All CPU times are in seconds, for a workstation with a 1.5GHz Xeon processor and 4x512MB RAM, running Red Hat Enterprise Linux (Version 4.3).

Table 1: The representation-based method vs. the search-based method: computing an at least w-weighted phylogeny with at most c incompatible characters.

# of incompatible characters (c)	weight (w)	method	time (CPU sec.s)	program size	memory size (MB)
16	45	representation-based	15.52	# of atoms: 79229 # of rules: 1585419	369
		search-based	1.34	# of atoms: 3744 # of rules: 55219	22
17	45	representation-based	15.32	# of atoms: 79229 # of rules: 1585419	369
		search-based	1.30	# of atoms: 3744 # of rules: 55219	22
18	45	representation-based	15.47	# of atoms: 79229 # of rules: 1585419	369
		search-based	1.10	# of atoms: 3744 # of rules: 55219	22

Table 2: The representation-based method vs. the search-based method: computing all at least w-weighted phylogenies with at most c incompatible characters.

# of incompatible characters (c)	weight (w)	# of phylogenies	method	time (CPU sec.s)	memory size (MB)
16	45	1	representation-based	14.49	369
			search-based	3.67	22
17	45	8	representation-based	126.06	369
			search-based	17	22
18	45	14	representation-based	210.32	369
			search-based	22.35	22

groupings: the similar phylogenies computed by CLASP-NKW are in the same group; the diverse phylogenies are in different groups.

8 Discussion

We studied the problem of computing weighted solutions in ASP, where the weight of a solution is defined explicitly. We introduced two methods to solve this problem. One method is based on the idea of modifying the representation of the problem so that we can use an existing ASP solver to compute weighted solutions. In this method, the weight measure and the weight constraints are encoded as ASP programs, and the main program is modified in an elaboration tolerant way by simply adding these programs. The other method is based on the idea of modifying the search algorithm of an ASP solver so that we can compute weighted solutions without modifying the main program. In this method, the weight of a solution is

implemented as a C++ program (in a separate file), and the search algorithm of the ASP solver CLASP is modified to compute weighted solutions with respect to this program. The modified version of CLASP is called CLASP-W. We compared the search-based method with the representation-based method in reconstructing weighted phylogenies for Indo-European languages. We observed that CLASP-W performs better in terms of computation time and space.

One benefit of the search-based method (other than computational efficiency) is that it allows implementing complex numerical weight measures which cannot be easily encoded in ASP. On the other hand, to take advantage of computational efficiency of this method, one has to devise an admissible heuristic function for the weight measure. Though, in some cases (e.g., when it is easier to represent the weight measure in ASP) it might be desirable to use the representation-based method.

For some problems, we do not have to define the weight of a solution explicitly, and we can use aggregates (e.g., sum, count, times) to compute the weight, in the sense of [14, 8, 15]. For such problems, other approaches assign weights to weak constraints [7, 2] or some rules [11] and then find the answer set that minimizes the sum of the weights of the violated weak constraints or violated rules. However, these representation-based methods are not general enough to handle more complicated weight functions that requires further new definitions, like that of a phylogeny above where we need to define total deductions relative to domain-specific information.

Based on the promising experimental results above, we extended our search-based method to be able to compare weighted solutions; for that we modified CLASP-W in the spirit of [6]. The modified version of CLASP-W is called CLASP-NKW. By this way, given a weight measure ω for a phylogeny and a distance measure Δ for a set of phylogenies (both as C++ programs, in separate files), we can compute similar/diverse weighted phylogenies.

Our methods for computing (similar/diverse) weighted solutions, and the solvers CLASP-W and CLASP-NKW provide useful tools for analyzing/comparing solutions both in ASP and in phylogenetics. In ASP, there are many appealing applications (e.g., product configuration, planning) for which finding weighted and/or similar/diverse solutions could be useful. In phylogenetics, there is no software system that has utilities for analying/comparing phylogenies; in that sense, our search-based methods (including the weight/distance measures) allow experts to automatically analyze phylogenies.

Acknowledgments This work has been supported by TUBITAK Grant 107E229.

References

[1] Daniel R. Brooks, Esra Erdem, Selim T. Erdogan, James W. Minett, and Donald Ringe. Inferring phylogenetic trees using answer set programming. *JAR*,

39(4):471–511, 2007.

[2] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Strong and weak constraints in disjunctive datalog. In *Proc. of LPNMR*, pages 2–17, 1997.

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

[4] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[5] William H. E. Day and David Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, 35(2):224–229, 1986.

[6] Thomas Eiter, Esra Erdem, Halit Erdogan, and Michael Fink. Finding similar or diverse solutions in answer set programming. In *Proc. of ICLP*, 2009.

[7] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Computing preferred and weakly preferred answer sets by meta interpretation in answer set programming. In *Proc. of ASP Workshop*, 2001.

[8] 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.

[9] Martin Gebser, Benjamin Kaufmann, Andr Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *Proc. of IJCAI*, pages 386–392, 2007.

[10] Vladimir Lifschitz. What is answer set programming? In *Proc. of AAAI*, 2008.

[11] Emilia Oikarinen and Matti Järvisalo. Max-ASP: Maximum satisfiability of answer set programs. In *Proc. of LPNMR*, pages 236–249, 2009.

[12] Donald Ringe, Tandy Warnow, and Ann Taylor. Indo-European and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002.

[13] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.

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

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

[16] J.P White and J.F. O'Connell. *A Prehistory of Australia, New Guinea, and Sahul*. Academic, San Diego, CA, 1982.