Finding Similar or Diverse Solutions in Answer Set Programming *

Thomas Eiter[1], Esra Erdem[2], Halit Erdoğan[2], and Michael Fink[1]

[1] Institute of Information Systems, Vienna University of Technology, Vienna, Austria
{eiter,fink}@kr.tuwien.ac.at
[2] Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul, Turkey
esraerdem@sabanciuniv.edu,halit@su.sabanciuniv.edu

Abstract. We study finding similar or diverse solutions of a given computational problem, in answer set programming, and introduce offline methods and online methods to compute them using answer set solvers. We analyze the computational complexity of some problems that are related to finding similar or diverse solutions, and show the applicability and effectiveness of our methods in phylogeny reconstruction.

Keywords: similar/diverse solutions, answer set programming, phylogenies

1 Introduction

Although, for many computational problems, the main concern is to find a best solution (e.g., a most preferred product configuration, a shortest plan, a most parsimonious phylogeny), for some problems, computing a subset of good solutions that are diverse or similar may be desirable. For instance, in product configuration, one could be interested in obtaining several diverse configurations of a product instead of checking all possible configurations, to pick one. In planning, it may be desirable to compute a set of plans that are similar to each other, so that, when the plan that is being executed fails, one can switch to a most similar one. Motivated by such applications, we study the problem of computing similar or diverse solutions in answer set programming (ASP), and then show the applicability of our approach to another interesting problem: phylogeny reconstruction (i.e., computing leaf-labeled trees, called phylogenies, to model the evolutionary history of a set of species).

Problems related to computing similar or diverse solutions have been studied in the context of propositional logic [2], and constraint programming [12, 13]. On the other hand, although there are many appealing ASP applications (e.g., product configuration [22], planning [15], phylogeny reconstruction [4]), for which finding similar/diverse solutions could be useful, such problems have not been studied in ASP. The methods we develop in this paper fulfill this need in ASP.

Phylogeny reconstruction is important for research areas as disparate as genetics, historical linguistics, zoology, anthropology, archaeology. For example, a phylogeny

* This work has been supported by TUBITAK Grant 107E229 and the Wolfgang Pauli Institute, Vienna.

of parasites may help zoologists to understand the evolution of human diseases [6]; a phylogeny of languages may help scientists to better understand human migrations [23]. For a given set of taxonomic units, some existing phylogenetic systems, like that of [5, 4], generate more than one phylogeny that explains the evolutionary relationships between the given taxonomic units. There are phylogenetic systems that compute a summary of these phylogenies (a consensus tree [1] or a supertree [21]). However, in such cases, especially when there are too many phylogenies computed by a system, an expert needs to compare these phylogenies in detail, by analyzing the similar/diverse ones with respect to some distance measure, to pick the most plausible ones. Although there are precisely defined measures to find the distance between them [17, 3, 20, 14], there is no phylogenetic system that helps experts to analyze phylogenies by comparing them. The methods we develop in this paper fulfill this need in phylogenetics.

In particular, the main contributions of this paper are as follows.

- We describe two kinds of computational problems related to finding similar/diverse solutions of a given problem, in the context of ASP (Section 2). Both kinds of problems take as input an ASP program P that describes a problem, a distance measure Δ that maps a set of solutions of the problem to a nonnegative integer, and two nonnegative integers n and k. One problem asks for a set S of size n that contains k-similar (resp. k-diverse) solutions, i.e., $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$); the other problem asks, given a set S of n solutions, for a k-close (k-distant) solution s ($s \notin S$), i.e., $\Delta(S \cup \{s\}) \leq k$ (resp. $\Delta(S \cup \{s\}) \geq k$).
- We study the computational complexity of these problems establishing completeness results under reasonable assumptions for the problem parameters (Section 3).
- We introduce an offline method to compute a set of n k-similar (or k-diverse) solutions to a given problem, by computing all solutions in advance and then using some clustering methods to find the similar (diverse) solutions (Section 4).
- We introduce three online methods to compute a set of n k-similar (or k-diverse) solutions to a given problem (Section 5). Online Method 1 reformulates the given program to compute n-distinct solutions and formulates the distance function as an ASP program, so that all n k-similar (k-diverse) solutions can be extracted from an answer set for the union of these ASP programs. Online Method 2 does not modify the given ASP program, but formulates the distance function as an ASP program, so that a k-close (or k-distant) solution can be extracted from an answer set for the union of these ASP programs and a previously computed solution; by iteratively computing k-close (k-distant) solutions, we can compute online a set of n k-similar (or k-diverse) solutions. Online Method 3 does not modify the given program, and does not formulate the distance function as an ASP program, but it modifies the ASP solver, in our case CLASP [10], to compute all n k-similar (or k-diverse) solutions at once.
- We illustrate the applicability of these approaches on the phylogeny reconstruction problem, by defining new distance measures for a set of phylogenies (Section 6), by describing how the offline method and the online methods are applied to find similar/diverse phylogenies (Section 7). After that, we compare the efficiency and effectiveness of these methods on the family of Indo-European languages studied in [4] (Section 8).

ASP programs mentioned below are presented in an extended version `http://people.sabanciuniv.edu/esraerdem/papers/iclp09-extended.pdf`.

2 Computational Problems

We are interested in the following two sorts of problems related to computation of a diverse/similar collection of solutions:

> n k-SIMILAR SOLUTIONS (resp.n k-DIVERSE SOLUTIONS)
> Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure Δ that maps a set of solutions for P to a nonnegative integer, and two nonnegative integers n and k, decide whether a set S of n solutions for P exists such that $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$).

> k-CLOSE SOLUTION (resp.k-DISTANT SOLUTION)
> Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure Δ that maps a set of solutions for P to a nonnegative integer, a set S of solutions for P, and a nonnegative integer k, decide whether a solution s ($s \notin S$) for P exists such that $\Delta(S \cup \{s\}) \leq k$ (resp. $\Delta(S \cup \{s\}) \geq k$).

For instance, suppose that $\mathcal{P}$ describes the phylogeny reconstruction problem for Indo-European languages. Then finding three diverse phylogenies is an instance of the former problem. On the other hand, if we already have picked two phylogenies, then finding another phylogeny that differs from these two is an instance of the latter.

The first kind of problems above has two parameters, n and k, so we can fix one and try to minimize (resp. maximize) the distance between solutions to find the most similar (resp. diverse) solutions.

> MAXIMAL k-SIMILAR SOLUTIONS (resp.MAXIMAL k-DIVERSE SOLUTIONS)
> Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure Δ that maps a set of solutions for P to a nonnegative integer, and a nonnegative integer k, find a maximal set S of solutions for P such that $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$) exists.

> n MOST SIMILAR SOLUTIONS (resp.n MOST DIVERSE SOLUTIONS)
> Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure Δ that maps a set of solutions for P to a nonnegative integer, and a nonnegative integer n, find a set S of n solutions for P with the minimum (resp. maximum) distance $\Delta(S)$.

Similarly, in the second class of problems, we can try to minimize (resp. maximize) the distance k between a solution and a set of solutions, to find the most close (resp. distant) solution.

> MOST CLOSE SOLUTION (resp.MOST DISTANT SOLUTION)
> Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure Δ that maps a set of solutions for P to a nonnegative integer, and a set S of solutions for P, find a solution s ($s \notin S$) for P with the minimum (resp. maximum) distance $\Delta(S \cup \{s\})$.

Table 1. Complexity results for computing similar solutions.

#	Problem	Complexity
1	n k-SIMILAR SOLUTIONS	NP
2	k-CLOSE SOLUTION	NP
3	MAXIMAL k-SIMILAR SOLUTIONS	FNP//log
4	n MOST SIMILAR SOLUTIONS	$\mathrm{FP}^{\mathrm{NP}}$ (FNP//log)
5	MOST CLOSE SOLUTION	$\mathrm{FP}^{\mathrm{NP}}$ (FNP//log)
6	k-CLOSE SET	NP

We can generalize k-CLOSE SOLUTION (resp.k-DISTANT SOLUTION) problems to sets of solutions:

> k-CLOSE SET (resp.k-DISTANT SET)
> Given an ASP program $\mathcal{P}$ that formulates a computational problem P, a distance measure Δ that maps a set of solutions for P to a nonnegative integer, a set S of solutions for P, and a nonnegative integer k, decide whether a set S' of solutions for P exists such that $|\Delta(S) - \Delta(S')| \leq k$ (resp. $|\Delta(S) - \Delta(S')| \geq k$).

3 Complexity Results

In this section, we turn our attention to the computational complexity of the problems presented in Section 2. In order to do so, we first have to make some reasonable assumptions on some of the problem parameters.

For the remainder of this section, let the ASP program $\mathcal{P}$ that formulates a computational problem P, be a propositional normal logic program. We assume that the given number n of different solutions to consider (respectively the size of the set S) in instances of the problems n k-SIMILAR SOLUTIONS and n MOST SIMILAR SOLUTIONS is polynomial in the size of the input.

Furthermore, we consider distance measures Δ that map a set of solutions for P to a nonnegative integer (which is usually implicitly done when real values have to be represented for computation). As for computing $\Delta(S)$ for a set of solutions S, in general we assume that deciding whether $\Delta(S) \leq k$ for a given k is in NP. Moreover, we assume that the value of $\Delta(S)$ is bounded by an exponential in the size of S (and thus has polynomially many bits in the size of S).

Under these assumptions, the computational complexity (cf. [18] for a background on the subject) of the problems concerning the computation of similar or diverse solutions we are interested in, is given in Table 1. All entries are completeness results (under usual reductions) and hardness holds even if $\Delta(S)$ is computable in polynomial time. Moreover, the results are the same for the 'symmetric' problems, i.e., when SIMILAR is replaced with DIVERSE, and CLOSE is replaced with DISTANT, respectively.

Membership for problem n k-SIMILAR SOLUTIONS (resp. n k-DIVERSE SOLUTIONS) follows from the fact that we can guess not only a candidate set S (since S is polynomially bounded) but also a witness for $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$), and check in polynomial time whether every $s \in S$ is a solution and that $\Delta(S) \leq k$ (resp.

$\Delta(S) \geq k$). For hardness, one simply reduces answer-set existence for normal, propositional programs to this problem, which is an NP-complete problem. However, hardness holds also for nodal distance of trees (a distance measure introduced in Section 6 for comparing phylogenies) encoded in a program (which naturally uses auxiliary atoms).

Theorem 1. *Problem n k-SIMILAR SOLUTIONS (resp. n k-DIVERSE SOLUTIONS) is NP-complete. Hardness holds even if $\Delta(S)$ is computable in polynomial time.*

For a hardness result resorting to partial Hamming distance confer [2]. By similar arguments we obtain NP-completeness for problem k-CLOSE SOLUTION (resp. k-DISTANT SOLUTION).

Theorem 2. *Problem k-CLOSE SOLUTION (resp. k-DISTANT SOLUTION) is NP-complete. Hardness holds even if $\Delta(S)$ is computable in polynomial time.*

When looking for maximal (wrt. subset inclusion) solutions, we face a function problem; here we assume that any S of size larger than n is clipped to any subset S' of S of size n. In particular, MAXIMAL k-SIMILAR SOLUTIONS (resp. MAXIMAL k-DIVERSE SOLUTIONS) is solvable in $\mathrm{FNP}//\log$. Intuitively, $\mathrm{FNP}//\log$ is the class of function problems solvable in polynomial time using a nondeterministic Turing Machine with output tape that may consult once an oracle that computes the optimal value of an optimization problem solvable in NP. A requirement is that this value has logarithmically many bits in the size of the input (see, e.g., [7, 9] for more information on $\mathrm{FNP}//\log$ and other function classes used in this section).

Membership can be shown by computing the cardinality of a maximal set of solutions S using the oracle. Note that since $|S|$ is polynomially bounded in the size of the input, it has logarithmically many bits as required. Then, one can nondeterministically compute a set S of respective size together with a witness for $\Delta(S) \leq k$, and check in polynomial time that this is indeed the case.

Hardness can be shown, e.g., for $\Delta(S)$ that takes the maximal (resp. minimal) Hamming distance between answer sets in S on a subset of the atoms; note that such a *partial Hamming distance* is a natural measure for problem encodings, where the disagreement on output atoms is measured. This measure is not unrelated to the ones introduced for comparing phylogenies in Section 6; one can polynomially reduce nodal distance to partial Hamming distance, and vice versa also partial Hamming distance to nodal distance of trees (allowing auxiliary atoms in the LP encoding).

Theorem 3. *Problem MAXIMAL k-SIMILAR SOLUTIONS (resp. MAXIMAL k-DIVERSE SOLUTIONS) is $\mathrm{FNP}//\log$-complete. Hardness holds even if $\Delta(S)$ is computable in polynomial time.*

$\mathrm{FP}^{\mathrm{NP}}$-membership of n MOST SIMILAR SOLUTIONS (resp. n MOST DIVERSE SOLUTIONS) is obtained by first using the NP-oracle to compute the minimum distance using binary search (deciding polynomially many n k-SIMILAR SOLUTIONS problems). Then, the oracle is used to compute S in polynomial time. Hardness follows from a reduction of the Traveling Salesman Problem (TSP). Notably, if the distances are polynomial in the size of the input, i.e., if the value of $\Delta(S)$ is polynomially bounded in the size of S, then the problem is $\mathrm{FNP}//\log$-complete.

Theorem 4. *Problem* n MOST SIMILAR SOLUTIONS *(resp.* n MOST DIVERSE SOLUTIONS*) is* $\mathrm{FP}^{\mathrm{NP}}$*-complete, and* $\mathrm{FNP}//\log$*-complete if the value of* $\Delta(S)$ *is polynomial in the size of* S*. Hardness holds even if* $\Delta(S)$ *is computable in polynomial time.*

Proceeding similarly as before, completeness for $\mathrm{FP}^{\mathrm{NP}}$(resp. $\mathrm{FNP}//\log$ if $\Delta(S)$ is small) is obtained for MOST CLOSE SOLUTION (and for MOST DISTANT SOLUTION).

Theorem 5. *Problem* MOST CLOSE SOLUTION *(resp.* MOST DISTANT SOLUTION*) is* $\mathrm{FP}^{\mathrm{NP}}$*-complete, and* $\mathrm{FNP}//\log$*-complete if the value of* $\Delta(S)$ *is polynomial in the size of* S*. Hardness holds even if* $\Delta(S)$ *is computable in polynomial time.*

For the generalization of k-CLOSE SOLUTION (resp. of k-DISTANT SOLUTION) to sets, namely k-CLOSE SET (resp. k-DISTANT SET), NP-completeness holds by similar arguments as for the former problem(s).

Theorem 6. *Problem* k-CLOSE SET *(resp.* k-DISTANT SET*) is* NP*-complete. Hardness holds even if* $\Delta(S)$ *is computable in polynomial time.*

4 Offline Methods

We introduce an offline method to compute a set of n k-similar (resp. k-diverse) solutions to a given problem, by computing all solutions in advance and then using some clustering methods to find the similar (diverse) solutions. The idea is to make clusters of n solutions, measure the distance of the set of solutions in each cluster, and pick the cluster whose distance is less (resp. greater) than k.

We can solve this problem by means of a graph problem: build a complete graph G whose nodes correspond to solutions and edges are labeled by distances between the corresponding solutions; and decide whether there is a clique C of size n in G whose weight (i.e., the distance of the set of solutions denoted by the clique) is less than k (resp. greater than k). The set of vertices in the clique represents n k-similar phylogenies. Such a clique can be computed using ASP, or one of the existing exact/approximate algorithms.

5 Online Methods

We introduce three online methods to compute a set of n k-similar (or k-diverse) solutions to a given problem P, given an ASP program $\mathcal{P}$ that represents P and a distance function Δ that maps a set of solutions of P to a nonnegative integer.

Online Method 1 (Fig. 1) reformulates the given program $\mathcal{P}$ to compute n-distinct solutions, formulates the distance function Δ as an ASP program $\mathcal{D}$, and formulates constraints on the distance function as an ASP program $\mathcal{C}$, so that all n k-similar (k-diverse) solutions can be extracted from an answer set for the union of these ASP programs, $\mathcal{P} \cup \mathcal{D} \cup \mathcal{C}$. Such a reformulation of $\mathcal{P}$ can be obtained in two stages. First, we copy every rule of the program n times: the i'th copy of the rule is obtained from r by replacing every atom $p(t_1, t_2, ..., t_m)$ in r with $p(i, t_1, t_2, ..., t_m)$. Now we have a program that computes n solutions to the problem P. Then, we add a constraint to ensure that no two solutions are same.

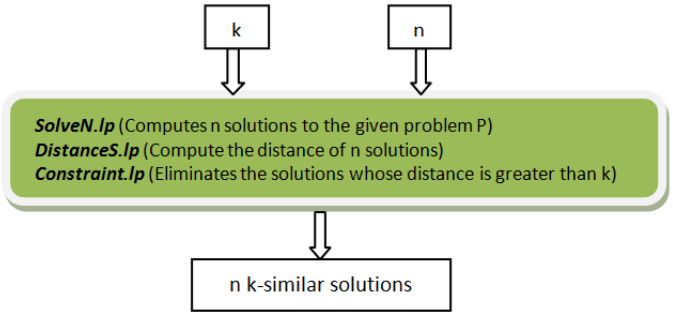

Fig. 1. Computing n k-similar solutions, with Online Method 1.

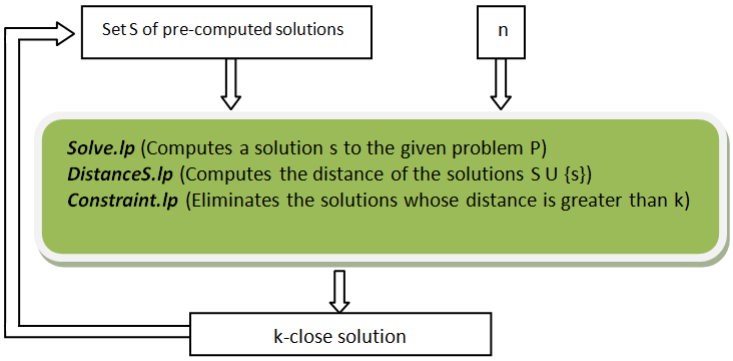

Fig. 2. Computing n k-similar solutions, with Online Method 2. Initially $S = \emptyset$. In each run, a solution is computed and added to S, until $|S| = n$. The distance function and the constraints in the program ensures that when we add the computed solution to S, the set stays k-similar.

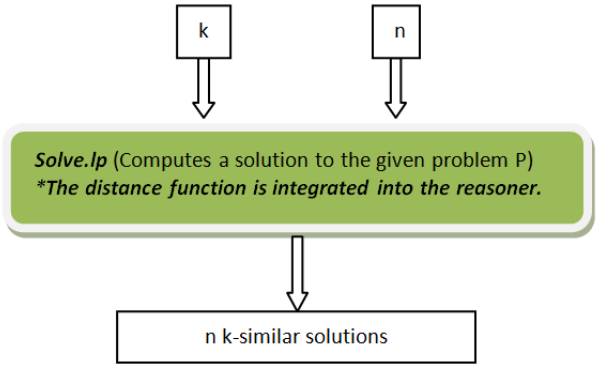

Fig. 3. Computing n k-similar solutions, with Online Method 3. We implement the distance function into the ASP reasoner, so that the ASP reasoner becomes biased to compute similar solutions. Each computed solution is stored by the reasoner until a set of n k-similar solutions is computed.

Online Method 2 (Fig. 2) does not modify the given ASP program $\mathcal{P}$, but formulates the distance $\Delta(S)$ of a given set S of solutions as an ASP program $\mathcal{D}$, and constraints

on the distance function as an ASP program $\mathcal{C}$, so that a k-close (or k-distant) solution can be extracted from an answer set for $\mathcal{P} \cup \mathcal{D} \cup \mathcal{C}$. By iteratively computing a k-close (k-distant) solution, we can compute online a set of n k-similar (or k-diverse) solutions.

Online Method 3 (Fig. 3) does not modify the given program, and does not formulate the distance function as an ASP program, but it modifies the ASP solver CLASP to compute all n k-similar (or k-diverse) solutions at once.

6 Distance Measures for Similar or Diverse Phylogenies

A *phylogenetic tree (phylogeny)* for a set of taxonomic units is a finite rooted leaf-labeled binary tree. To compare a set of phylogenies, and analyze the similar or diverse ones in this set, we can measure the distance of a set of phylogenies by some function Δ. In the following, we introduce a distance function to measure the similarity/diversity of a set of phylogenies, in terms of a distance function for two phylogenies. We present the trees in the Newick format, where the sister subtrees are enclosed by parentheses.

Two distance functions for two phylogenies Among the existing functions for measuring the distance between two trees [17, 3, 20, 14], we consider the distance function of [3] based on the nodal distances in trees. The *nodal distance* $ND_T(x, y)$ between two leaves x and y in a tree T is the number of edges contained in the shortest path from one leaf to the other. For example, consider the tree $(a, (b, c))$; the nodal distance between a and b is 3, whereas the nodal distance between b and c is 2. Intuitively, the nodal distance between two leaves in a tree represents the degree of their relationship in that tree. After defining the nodal distance, [3] measures the distance $D_n(T, T')$ between two leaf-labeled trees T and T', both with the same set L of leaves, as follows:

$$D_n(T, T') = \sum_{(x,y)\in L} |ND_T(x, y) - ND_{T'}(x, y)|.$$

The difference of the nodal distances of two leaves in two trees represents the contribution of these leaves to the distance between the trees. Let $T_1 = (a, (b, c))$ and $T_2 = (c, (a, b))$ be two trees. In order to compute the distance between T_1 and T_2, we compute the nodal distances of the pairs $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$ for both trees and take the sum of the differences. In this case the distance between T_1 and T_2 is 2.

The second distance function we consider is introduced specifically for languages, based on our discussions with the historical linguist Don Ringe. For each vertex x of a tree $\langle V, E\rangle$, let *desc*(x) denote its descendants and *depth*(x) its depth. To define the similarity of two phylogenies $\langle V, E\rangle$ and $\langle V', E'\rangle$, let us first define the similarity of two vertices $v \in V$ and $v' \in V'$: $f(v, v') = 1$ if *desc*$(v) \neq$ *desc*(v'); and $f(v, v') = 0$ otherwise. Let *weight* be a function mapping every depth to a nonnegative integer. Then we can define the similarity of two trees $T = \langle V, E\rangle$ and $T' = \langle V', E'\rangle$, with the roots R and R' respectively, at depth i ($0 \leq i \leq \min\{\max_{v\in V}$ *depth*$(v), \max_{v'\in V'}$ *depth*$(v')\}$), by the following measure:

$g(0, T, T') =$ *weight*$(0) \times f(R, R')$
$g(i + 1, T, T') = g(i, T, T') +$ *weight*$(i + 1) \times \sum_{x\in V, y\in V',}$ *depth*$_{(x)=}$*depth*$_{(y)=i+1} f(x, y)$

and the similarity of two trees as follows:

$$D_l(T, T') = g(\min\{\max_{v \in V} \mathit{depth}(v), \max_{v' \in V'} \mathit{depth}(v')\}, T, T').$$

For instance, for $T_1 = (a, (b, (c, (d, e))))$ and $T_2 = (a, (d, (c, (b, e))))$, $D_l(T_1, T_2) = 8$. The idea is to assign bigger weights to smaller depths so that two phylogenies are more similar if the diversifications closer to the root are more similar. This is motivated by that reconstructing the evolution of languages closer to the root is more important for historical linguists.

A distance function for a set of phylogenies We define a distance function for measuring the distance of a set S of phylogenies, based on a distance function D for two phylogenies: for similarity (resp. diversity) we take the maximum (resp. minimum) of the distances between pairs of phylogenies in S

$$\Delta_D(S) = \max\{D(T_1, T_2) \mid T_1, T_2 \in S\}$$

In the following, we show the applicability of the offline methods and online methods, with the distance functions Δ_{D_n} and Δ_{D_l}.

7 Computation of Similar or Diverse Phylogenies

We can find n k-similar (resp. k-diverse) phylogenies for a set of taxonomic units, with an offline method as described in Section 4. Consider, for instance, a family of languages as the taxonomic units. With the approach of [4], we can compute all the phylogenies for a given set of languages. Then we build a complete graph G whose nodes denote these phylogenies, and the edges are labeled by the distances between phylogenies. Then we can find a clique of size n in G, such that the distance of the set of phylogenies denoted by this clique is less than or equal to k, as follows: remove each edge in G whose label is greater than k; and, ignoring the weights of the edges in the resulting graph, find a clique of size n. The set of vertices in the clique represents n k-similar phylogenies for the given set of taxonomic units.

In the online methods, we consider the ASP program `phylogeny-improved.lp` described in [4], to reconstruct phylogenies.

Online Method 1 suggests finding n k-similar (resp. k-diverse) phylogenies, by reformulating the given ASP program for phylogeny reconstruction, and using an answer set solver to compute all these solutions. A reformulation of `phylogeny-improved.lp`, as suggested by the first online method, can be obtained as follows:

1. We specify the number of solutions: `solution(1..n).`
2. In each rule of the program, we replace each atom `p(T1,T2,...,Tm)` (except the ones specifying the input, like atoms describing the leaves, the labels of the leaves, characters, and states of characters) with `p(N,T1,T2...,Tm)`, and add to the body `solution(N).`
3. Now we have a program that computes n phylogenies. To ensure that they are distinct, for each atom specifying a solution, in this case atoms describing the edges of a phylogeny, we add the rules

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
  UNIT-PROPAGATION(Π, A, ▽)    // propagate according to the current assignment and conflicts, and update the current assignment
  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
```

```
different(S1,S2) :- edge(S1,X1,Y), edge(S2,X2,Y),
   vertex(X2;X1;Y), solution(S1;S2), S1 != S2, X1 != X2.
:- not different(S1,S2), solution(S1;S2), S1 != S2.
```

Online Method 2 suggests finding n k-similar (resp. k-diverse) phylogenies, by iteratively computing a k-close (resp. k-distant) phylogeny. Here we implement a perl script that calls the ASP solver repeatedly, with the phylogeny reconstruction program `phylogeny-improved.lp` and a distance function program, until we compute all n k-similar solutions.

Online Method 3 suggests finding n k-similar (resp. k-diverse) phylogenies, by modifying the ASP solver. Consider for instance the answer set solver CLASP [10]. CLASP does a conflict-driven DPLL-like [8, 16] Branch & Bound search to find an answer set (solution) of 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 UNIT-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 is no conflict and the currently selected literals do not represent an answer set, then, in SELECT, CLASP selects a new literal (based on BERKMIN's heuristic [11]) to continue search.

We can modify CLASP as in Algorithm 2, to compute n k-similar phylogenies. The modified solver, CLASP-NK, has some additional procedures: DISTANCE-ANALYZE identifies the partial phylogeny formed by the currently selected literals, and then computes a lower bound for the distance between a phylogeny that contains this partial phylogeny and the previously computed full phylogenies. Computing an exact lower bound requires enumerating all possible completions of the partial phylogeny, so we

Algorithm 2 CLASP-NK

Input: An ASP program Π, nonnegative integers n, and k, and a set C of atoms considered in computation of the distance function
Output: A set X of n phylogenies that are k similar (n k-similar phylogenies)

```
X ← ∅    // computed phylogenies
A ← ∅    // current assignment of literals
▽ ← ∅    // set of conflicts
while |X| < n do
   PartialSolution ← CurSelCon(A, C)    // the atoms that are marked as considered and that are currently selected constitute a partial solution
   d ← DISTANCE-ANALYZE(X, PartialSolution)    // compute a lower bound for the distance between partial solution and previously computed phylogenies
   if d > k then
      RESOLVE-CONFLICT(Π, A, ▽)
   end if
   UNIT-PROPAGATION(Π, A, ▽)
   if there is a conflict in the current assignment then
      RESOLVE-CONFLICT(Π, A, ▽)
   else
      if current assignment does not yield an answer set then
         SELECT(Π, A, ▽)
      else
         X ← X ∪ A
         A ← ∅    // start searching for a new solution
      end if
   end if
end while
return X
```

compute an approximate lower bound by a heuristic function $LB(T, T')$ that estimates the distance (from below) between a complete phylogeny T and a complete phylogeny that contains a partial phylogeny T' with leaves L':

$$LB(T, T') = \sum_{(x,y) \in L'} |ND_T(x, y) - ND_{T'}(x, y)|.$$

Since this heuristic function is admissible (i.e., its value is always less than or equal to the exact lower bound), CLASP-NK does not miss a solution (n k-similar phylogenies) if one exists. This function is also monotonic in the number of leaves in partial phylogeny: if the partial phylogeny grows, then the distance increases also. If the lower bound $LB(T, T')$ is greater than k, then there is no need for CLASP-NK to search for a solution. In such a case, CLASP-NK marks the currently selected literals as a conflict, learns this conflict, and does the necessary backtracking. The rest of the algorithm is the same as that of CLASP except that CLASP-NK searches until it finds n solutions.

We can use other distance functions for CLASP-NK or we can compute similar/diverse solutions to other problems (e.g., planning, product configuration). For that, we need to modify CLASP-NK: we need to implement a suitable admissible distance measure, and change the DISTANCE-ANALYZE function of CLASP-NK.

8 Experimental Results

We applied the computational methods described above (i.e., the offline method, and the three online methods) to reconstruct similar/diverse phylogenies for Indo-European languages. We used the dataset assembled by Don Ringe and Ann Taylor [19]. As in [4], to compute such phylogenies, we considered the language groups Balto-Slavic (BS), Italo-Celtic (IC), Greco-Armenian (GA), Anatolian (AN), Tocharian (TO), Indo-Iranian (IIR), Germanic (GE), and the language Albanian (AL). While computing phylogenies, we also took into account some domain-specific information about these languages.

Let us first examine the results of experiments, considering the distance measures Δ_{D_n}, based on the nodal distance (Table 2). We present the results for the following computations: 2 most similar solutions, 2 most diverse solutions, 3 most similar solutions, 3 most diverse solutions, 6 most similar solutions. We solve these optimization problems by iteratively solving the corresponding decision problems (n k-SIMILAR/DIVERSE SOLUTION). For each method, we present the computation time,[3] the size of the memory used in computation, and the optimal value of k.

Let us first compare the online methods. In terms of both computation time and memory size, Online Method 3 performs the best, and Online Method 2 performs better than Online Method 1. These results conforms with our expectations: Online Method 1 requires an ASP representation of computing n k-similar/diverse phylogenies, and such a program may be too large for an answer set solver to compute an answer set for. Online Method 2 relaxes this requirement a little bit so that the answer set solver can compute the solutions more efficiently: it requires an ASP representation of phylogeny reconstruction, and an ASP representation of the distance measure, and then computes similar/diverse solutions one at a time. However, since the answer set solver needs to compute the distances between every two solutions, the computation time and the size of memory do not decrease much, compared to those for Online Method 1. Online Method 3 deals with the time consuming computation of distances between solutions, not at the representation level but at the search level; so it does not require an ASP representation of the distance function but requires a modification of the solver.

The offline method takes into account the previously computed 8 phylogenies for Indo-European languages (with at most 17 incompatible characters), and computes similar/diverse solutions using ASP as explained in Section 7. The offline method is more efficient, in terms of both computation time and memory, than Online Methods 1 and 2 since it does not compute phylogenies. On the other hand, the offline method is less efficient, in terms of both computation time and memory, than Online Method 3, since it requires both representation and computation of distances between solutions.

Here both the offline method and Online Method 1 guarantee to find an optimal solution, by iteratively solving the corresponding decision problems. On the other hand, Online Methods 2 and 3 compute similar/diverse solutions with respect to the first computed solution, and thus may not find the optimal value for k, as observed in the computation of 3 most similar phylogenies.

[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 2. Results of experiments, using the distance Δ_{D_n} based on the nodal distance.

Problem	Offline Method	Online Method 1	Online Method 2	Online Method 3
2 most similar	12.39 sec. 32MB $k = 12$	26.23 sec. 430MB $k = 12$	19.00 sec. 410MB $k = 12$	1.46 sec. 12MB $k = 12$
2 most diverse	11.81 sec. 32MB $k = 32$	21.75 sec. 430MB $k = 32$	18.41 sec. 410MB $k = 24$	1.01 sec. 15MB $k = 32$
3 most similar	11.59 sec. 32MB $k = 15$	60.20 sec. 730MB $k = 15$	43.56 sec. 626MB $k = 15$	1.56 sec. 15MB $k = 16$
3 most diverse	11.91sec. 32MB $k = 26$	46.32 sec. 730MB $k = 26$	44.67 sec. 626MB $k = 21$	0.96 sec. 15MB $k = 26$
6 most similar	11.66sec. 32MB $k = 25$	327.28 sec. 1.8GB $k = 25$	178.96 sec. 1.2GB $k = 29$	1.96 sec. 15MB $k = 25$

Table 3. Results of experiments, using the distance Δ_{D_l} based on preferred diversifications.

Problem	Offline Method	Online Method 1	Online Method 2
2 most similar	365.16 sec. (4.2GB)	16.11 sec. (236MB)	16.23 sec. (212MB)
3 most diverse	368.59 sec. (4.2GB)	46.11 sec. (659MB)	44.21 sec. (430MB)
6 most similar	368.45 sec. (4.2GB)	137.31 sec. (1.8GB)	212.59 sec. (1.1GB)

Now, let us consider the distance measures Δ_{D_l}, based on preference over diversifications (Table 3): two phylogenies are more similar if the diversifications closer to the root are more similar. Here we consider the similarities of diversifications until depth 3 (inclusive). We present the results for the following computations: 2 most similar solutions, 3 most diverse solutions, 6 most similar solutions. In Table 3, for each method, we present the computation time, the size of the memory used in computation, and the optimal value of k. Unlike what we have observed in Table 2, the offline method takes more time/space to compute similar/diverse solutions; this is due to the computation of distances with respect to Δ_{D_l} which requires summations, and representing summations in the language of LPARSE is not trivial. Other results are similar to the ones presented in Table 2.

In [4], after computing all 34 plausible phylogenies, the authors examine them manually and come up with three forms of tree structures, and then "filter" the phylogenies with respect to these tree structures. For instance, in Group 1, the trees are of the form (AN, (TO, (AL, (IC, (a tree formed for GE, GA, BS, IIR))))); in Group 2, the trees are of the form (AN, (TO, (IC, (a tree formed for GE, GA, BS, IIR, AL)))); in Group 3, the trees are of the form (AN, (TO, ((AL, IC), (a tree formed for GE, GA, BS, IIR)))). The results of our experiments with the distance measure Δ_{D_l} comply with these groupings. For instance, the 2 most similar phylogenies computed by Online Method 1 are in Group 1; the 3 most diverse phylogenies computed by Online Method 2 are in different groups. Likewise, the 6 similar phylogenies computed by our methods fall into Group 2.

9 Related Work

Finding similar or diverse solutions has been studied in propositional logic [2], and in constraint programming [13, 12].

In [2], the authors propose two algorithms, $DP_{distance}$ and $DP_{distance+lasso}$, to solve DISTANCE-SAT—determining that a propositional CNF formula has a model that disagrees with a given partial interpretation on at most d variables. Our modification of CLASP's algorithm is similar to the first algorithm in that both algorithms check whether a partial interpretation computed in the DPLL-like search obeys the given distance constraints. On the other hand, unlike $DP_{distance}$, CLASP also uses conflict-driven learning: when it learns a conflicting set of literals, it will never try to select them in the later stages of the search. $DP_{distance+lasso}$ offers manipulations while selecting a new variable: it creates a set of candidate variables with respect to the distance function, computes weights of these variables relative to the distance function, and selects one with the maximum weight. On the other hand, in SELECT, CLASP creates a set of candidate variables, and selects one of the candidates to continue the search. Using the idea of $DP_{distance+lasso}$, we can modify CLASP further to manipulate the selection of variables with respect to the distance function. However, in the phylogeny reconstruction problem, since the domain of the distance function consists of the edge atoms which are far outnumbered by many auxiliary atoms, in SELECT the set of candidate variables generally consists of only auxiliary variables; due to these cases, the manipulation of the selection of variables is not expected to improve the computational efficiency.

[13, 12] study various computational problems related to finding similar/diverse solutions, considering Hamming distance as in [2]. They present an offline method (similar to our method) that applies clustering methods, and two online methods: one based on reformulation (similar to Online Method 1), the other based on a greedy algorithm (similar to Online Method 2) that iteratively computes a solution that maximizes similarity to previous solutions. The computation of a k-close solution is due to a Branch & Bound algorithm (similar to the idea behind Online Method 3) that propagates some similarity/diversity constraints specific to the given distance function. Our offline/online methods are inspired by these methods of [13, 12], but are not confined to only polynomial-time distance functions with polynomial range.

10 Conclusion

We have studied two kinds of computational problems related to finding similar/diverse solutions of a given problem, in the context of ASP: one problem asks for a set of n solutions that are k-similar (resp. k-diverse); the other one asks for a solution that is k-close (k-distant) to a given set of solutions. We have analyzed the computational complexity of these problems, and introduced offline/online methods to solve them. We have applied these methods to the phylogeny reconstruction problem, and observed their effectiveness in comparing many phylogenies for Indo-European languages.

There are many appealing ASP applications (e.g., product configuration, planning) for which finding similar/diverse solutions could be useful; on the other hand, no existing phylogenetic system can analyze phylogenies by comparing them. In this sense, our methods are useful both for ASP and for phylogenetics.

Acknowledgments Thanks to Martin Gebser and Benjamin Kaufmann for their help with CLASP.

References

1. E.N. Adams. Consensus techniques and the comparison of taxonomic trees. *Syst. Zool*, 21:390–397, 1972.
2. O. Bailleux and P. Marquis. DISTANCE-SAT: complexity and algorithms. In *Proc. of AAAI*, pages 642–647, 1999.
3. J. Bluis and D-G. Shin. Nodal distance algorithm: Calculating a phylogenetic tree comparison metric. In *Proc. of BIBE*, page 87, 2003.
4. D.R. Brooks, E. Erdem, S.T. Erdoğan, J.W. Minett, and D. Ringe. Inferring phylogenetic trees using answer set programming. *JAR*, 39(4):471–511, 2007.
5. D.R. Brooks, E. Erdem, J.W. Minett, and D. Ringe. Character-based cladistics and answer set programming. In *Proc. of PADL*, pages 37–51, 2005.
6. D.R. Brooks and D.A. McLennan. *Phylogeny, Ecology, and Behavior: A Research Program in Comparative Biology*. University of Chicago Press, Chicago, IL, 1991.
7. Z-Z. Chen and S. Toda. The Complexity of Selecting Maximal Solutions. *Information and Computation*, 119:231–239, 1995.
8. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.
9. T. Eiter and V.S. Subrahmanian. Heterogeneous active agents, ii: Algorithms and complexity. *Artif. Intell.*, 108(1-2):257–307, 1999.
10. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proc. of IJCAI*, pages 386–392, 2007.
11. E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. *Discrete Appl. Math.*, 155(12):1549–1561, 2007.
12. E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh. Finding diverse and similar solutions in constraint programming. In *Proc. of AAAI*, pages 372–377, 2005.
13. E. Hebrard, B. O'Sullivan, and T. Walsh. Distance constraints in constraint satisfaction. In *Proc. of IJCAI*, pages 106–111, 2007.
14. W-K. Hon, M-Y. Kao, and T-W. Lam. *Algorithms and Computation*, chapter Improved Phylogeny Comparisons: Non-shared Edges, Nearest Neighbor Interchanges, and Subtree Transfers, pages 369–382. Springer Berlin / Heidelberg, 2000.
15. V. Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.
16. J. Marques-Silva and K. Sakallah. A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 5:506–521, 1999.
17. T.M. Nye, P. Lio, and W.R. Gilks. A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. *Bioinformatics*, 22(1):117–119, January 2006.
18. C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
19. D. Ringe, T. Warnow, and A. Taylor. Indo-European and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002.
20. D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, February 1981.
21. C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, 105:147–158, 2000.
22. T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proc. of PADL*, pages 305–319, 1998.
23. J.P White and J.F. O'Connell. *A Prehistory of Australia, New Guinea, and Sahul*. Academic, San Diego, CA, 1982.