Finding Similar/Diverse Solutions in Answer Set Programming

THOMAS EITER

Institute of Information Systems, Vienna University of Technology, Vienna, Austria
E-mail: eiter@kr.tuwien.ac.at

ESRA ERDEM and HALIT ERDOGAN

Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey
E-mail: {esraerdem,halit}@sabanciuniv.edu

MICHAEL FINK

Institute of Information Systems, Vienna University of Technology, Vienna, Austria
E-mail: fink@kr.tuwien.ac.at

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

For some computational problems (e.g., product configuration, planning, diagnosis, query answering, phylogeny reconstruction) computing a set of similar/diverse solutions may be desirable for better decision-making. With this motivation, we have studied several decision/optimization versions of this problem in the context of Answer Set Programming (ASP), analyzed their computational complexity, and introduced offline/online methods to compute similar/diverse solutions of such computational problems with respect to a given distance function. All these methods rely on the idea of computing solutions to a problem by means of finding the answer sets for an ASP program that describes the problem. The offline methods compute all solutions of a problem in advance using the ASP formulation of the problem with an existing ASP solver, like CLASP, and then identify similar/diverse solutions using some clustering methods (possibly in ASP as well). The online methods compute similar/diverse solutions of a problem following one of the three approaches: by reformulating the ASP representation of the problem to compute similar/diverse solutions at once using an existing ASP solver; by computing similar/diverse solutions iteratively (one after other) using an existing ASP solver; by modifying the search algorithm of an ASP solver to compute similar/diverse solutions incrementally. We have modified CLASP to implement the last online method and called it CLASP-NK. In the first two online methods, the given distance function is represented in ASP; in the last one however it is implemented in C++. We have showed the applicability and the effectiveness of these methods using CLASP or CLASP-NK on two sorts of problems with different distance measures: on a real-world problem in phylogenetics (i.e., reconstruction of similar/diverse phylogenies for Indo-European languages), and on several planning problems in a well-known domain (i.e., Blocks World). We have observed that in terms of computational efficiency (both time and space) the last online method outperforms the others; also it allows us to compute similar/diverse solutions when the distance function cannot be represented in ASP (e.g., due to some mathematical functions not supported by the ASP solvers) but can be easily implemented in C++.

KEYWORDS: similar/diverse solutions, answer set programming, similar/diverse phylogenies, similar/diverse plans

1 Introduction

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) with respect to some well-described criterion. On the other hand, in many real-world applications, computing a subset of good solutions that are similar/diverse may be desirable for better decision-making. For one reason, the given computational problem may have too many good solutions, and the user may want to examine only a few of them to pick one. Also, in many real-world applications the users usually take into account furthermore criterion that are not included in the formulation of the optimization problem; in such cases, good solutions similar to a best one may also be useful. Here are some examples from several domains illustrating the usefulness of finding similar/diverse solutions.

Product configuration Consider, for instance, a variation of the example given in (Hebrard et al. 2005) about buying a car. Suppose that there is a product advisor that asks customers about their constraints/preferences about a car, and then lists the available ones that match their constraints/preferences. However, such a list may be too long. In that case, the customer might ask for a few cars that not only suit her constraints/references but also are as diverse as possible. Then, if she likes one particular car among them, she might ask for other cars that are as similar as possible to this particular car. Also, the customer may have other (possibly secondary) criterion that the product advisor has not asked about; and thus the best alternatives listed by the product advisor may not cover some of the good possibilities. Then, the user may ask for a couple of good enough configurations that are distant from a set of best configurations.

Planning Given an initial state, goal conditions, and a description of actions, planning is the problem of finding a sequence of actions (i.e., a plan) that would lead the initial state to a goal state. Planning is applied in various domains, such as robotics, web service composition, and genome rearrangement. 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. For instance, consider a variation of the example given in (Srivastava et al. 2007) in connection with modeling web service composition as a planning problem (McIlraith and Son 2002): suppose that the web service engine computes a plan/composition; then it can compute a set of compositions similar to this particular one, so that if a failure occurs while executing one composition, an alternative composition which is less likely to be failing simultaneously can be used (Chafle et al. 2006). Alternatively, let us consider planning in the context of robotics in a dynamic environment with uncertainties. If the plan failure is, for instance, due to some collisions with an obstacle as in the scenarios presented in (Caldiran et al. 2009), the agent may want to find a plan that is distant from the previously computed plan so that it does not collide with the obstacle again.

Phylogeny reconstruction Phylogeny reconstruction is the problem of inferring a leaf-labeled rooted tree (i.e., phylogeny) that would characterize the evolutionary relations between a family of species based on their shared traits. Phylogeny reconstruction is important for research areas as disparate as genetics, historical linguistics, zoology, anthropology, archaeology, etc.. For example, a phylogeny of parasites may help zoologists to understand the evolution of human diseases (Brooks and McLennan 1991); a phylogeny of languages may help scientists to better understand human migrations (White and O'Connell 1982). For a given set of taxonomic units, some existing phylogenetic systems, like that of (Brooks et al. 2005; Brooks et al. 2007), generate more than one good phylogeny that explains the evolutionary relationships between the given taxonomic units. However, usually 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.

Motivated by such examples, we have studied various problems related to computing similar/diverse solutions in the context of a new declarative programming paradigm, called Answer Set Programming (ASP) (Lifschitz 2008). We have also introduced general offline/online methods in ASP that can be applied to various domains for such computations.

In ASP, a combinatorial search problem is represented as an ASP program whose models (called "answer sets") correspond to the solutions. The answer sets for the given formalism can be computed by special systems called answer set solvers, such as CLASP (Gebser et al. 2007a). Due to the expressive formalism of ASP that allows us to represent, e.g., negation, defaults, aggregates, recursive definitions, and due to the continuous improvements of efficiency of solvers, ASP has been used in a wide-range of knowledge-intensive applications from different fields, such as product configuration (Soininen and Niemelä 1998), planning (Lifschitz 1999), phylogeny reconstruction (Brooks et al. 2007), developing a decision support system for a space shuttle (Nogueira et al. 2001), multi-agent planning (Son et al. 2009), answering biomedical queries (Bodenreider et al. 2008). For many of these applications, finding similar/diverse solutions (and thus the methods we have developed for computing similar/diverse solutions in ASP) could be useful.

The main contributions of this paper can be summarized as follows.

- We have described mainly two kinds of computational problems related to finding similar/diverse solutions of a given problem, in the context of ASP (Section 3). 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 (resp. k-distant) solution s (resp. $s \notin S$), i.e., $\Delta(S \cup \{s\}) \leq k$ (resp. $\Delta(S \cup \{s\}) \geq k$). Note that, by fixing some parameters and minimizing/maximizing others, we can turn them into various related optimization problems.
- We have studied the computational complexity of these decision/optimization problems establishing completeness results under reasonable assumptions for the problem parameters (Section 4).
- We have introduced 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 using ASP and then finding similar (resp. diverse) solutions using some clustering methods, possibly in ASP as well (Section 5.1.1).

- We have introduced three online methods to compute a set of n k-similar (resp. k-diverse) solutions to a given problem (Sections 5.1.2, 5.1.3 and 5.1.4).
  - 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 (resp. k-diverse) solutions can be extracted from an answer set for the union of these ASP programs.
  - Online Method 2 does not modify the ASP program encoding the problem, but formulates the distance function as an ASP program, so that a unique k-close (resp. 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 (resp. k-distant) solutions one after other, we can compute online a set of n k-similar (or k-diverse) solutions.
  - Online Method 3 does not modify the ASP encoding of the problem, and does not formulate the distance function as an ASP program, but it modifies the search algorithm of an ASP solver, in our case CLASP (Gebser et al. 2007b), to compute all n k-similar (or k-diverse) solutions at once. The distance function is implemented in C++; in that sense, Online Method 3 allows for finding similar/diverse solutions when the distance function cannot be defined in ASP.
- We have illustrated the applicability of these approaches on two sorts of problems: phylogeny reconstruction (based on the ASP encoding of the problem as in (Brooks et al. 2007)) and planning (based on the ASP encoding of the Blocks World as in (Erdem 2002)).
  - For phylogeny reconstruction, we have defined novel distance measures for a set of phylogenies (Section 6.1), described how the offline method and the online methods are applied to find similar/diverse phylogenies (Section 6.2), and compared the efficiency and effectiveness of these methods on the family of Indo-European languages studied in (Brooks et al. 2007) (Section 6.3). Since there is no phylogenetic system that helps experts to analyze phylogenies by comparing them, this particular application of our methods also plays a significant role in phylogenetics. In fact, Offline Method and Online Method 3 are integrated in the phylogenetics system PHYLO-ASP (Erdem 2009).
  - For planning, we have considered the action-based Hamming distance of (Srivastava et al. 2007) to measure the distance among plans, and compared the efficiency and effectiveness of the offline method and the online methods on some Blocks World problems (Section 7).

Finding similar/diverse solutions has earlier been studied in the context of propositional logic (Bailleux and Marquis 1999), constraint programming (Hebrard et al. 2005; Hebrard et al. 2007), and automated planning (Srivastava et al. 2007). These studies consider the Hamming distance (Hamming 1950) as a measure to compute distances between solutions. Unlike the problems studied in related work, the problems we have studied are not confined to polynomial-time distance functions with polynomial range. A more detailed discussion on related work is presented in Section 8.

2 Answer Set Programming

We study finding similar/diverse solutions in the context of Answer Set Programming (ASP) (Lifschitz 2008)—a new declarative programming paradigm where the idea is to represent a combinatorial search problem as a "program" whose models (called "answer sets" (Gelfond and Lifschitz 1991)) correspond to the solutions. This is in the vein of SAT solving, which became popular after a surprising success in the area of planning (Kautz and Selman 1992), but offers in comparison features like variables ranging over domain elements, easy definition of transitive closure, and nonmonotonic negation. Furthermore, a range of special constructs, such as aggregates, weight constraints and priorities, that are useful in practical applications are supported by various ASP solvers; for more discussion, see Section 8.

Before we proceed discussing our methods for finding similar/diverse solutions in ASP, let us present the syntax of the kind of programs considered in this paper, and define the concept of an answer set for such programs.[1]

Programs The syntax of formulas, rules and programs is defined as follows. *Formulas* are formed from propositional atoms and 0-place connectives $\top$ and $\bot$ using negation (written as *not*), conjunction (written as a comma) and disjunction (written as a semicolon).

A *rule* is an expression of the form

$$F \leftarrow G \tag{1}$$

where F is an atom or $\bot$, and G is a formula; F is called the *head* and G is called the *body* of the rule. A rule of the form $F \leftarrow \top$ will be identified with the formula F. A rule of the form $\bot \leftarrow F$ (called a *constraint*) will be abbreviated as $\leftarrow F$.

A *(normal nested) program* is a finite set of rules. If bodies of all rules in a program are of the form

$$A_1, \ldots, A_m, A_{m+1}, \ldots, A_n$$

then the program is a *normal program*. A program is *positive* if it does not contain any negation.

Answer Sets To define the concept of an answer set for a program, let us first define the satisfaction relation and the reduct of a program.

The *satisfaction relation* $X \models F$ between a set X of atoms and a formula F is defined recursively, as follows:

- for an atom A, $X \models A$ if $A \in X$
- $X \models \top$
- $X \not\models \bot$
- $X \models (F, G)$ if $X \models F$ and $X \models G$
- $X \models (F; G)$ if $X \models F$ or $X \models G$
- $X \models$ *not* F if $X \not\models F$.

[1] Answer sets are defined for programs of a more general form that may contain classical negation $\neg$ and disjunction (Gelfond and Lifschitz 1991) and nested expressions (Lifschitz et al. 1999) in heads of rules as well. See (Lifschitz 2010) for definitions of answer sets.

We say that X *satisfies* a program Π (symbolically, $X \models \Pi$) if, for every rule $F \leftarrow G$ in Π, $X \models F$ whenever $X \models G$.

The *reduct* F^X of a formula F with respect to a set X of atoms is defined recursively, as follows:

- if F is an atom or a 0-place connective then $F^X = F$
- $(F, G)^X = F^X, G^X$
- $(F; G)^X = F^X; G^X$
- $(\textit{not}\ \ F)^X = \begin{cases} \bot\,, & \text{if } X \models F, \\ \top\,, & \text{otherwise.} \end{cases}$

The *reduct* Π^X of a program Π with respect to X is the set of rules

$$F^X \leftarrow G^X$$

for all rules $F \leftarrow G$ in Π.

Let us first define the answer set for a program Π that does not contain negation. We say that X is an *answer set* for Π, if X is minimal with respect to set inclusion ($\subseteq$) among the sets of atoms that satisfy Π. For instance, the set $\{p\}$ is the answer set for the program consisting of the single rule

$$p \leftarrow \cdot \tag{2}$$

Now consider a program Π that may contain negation. A set X of atoms is an *answer set* for Π if it is the answer set for the reduct Π^X. For instance, the reduct of the program

$$p \leftarrow \textit{not}\ \ \textit{not}\ \ p \tag{3}$$

relative to $\{p\}$ is (2). Since $\{p\}$ is the answer set for (2), $\{p\}$ is an answer set for program (3). Similarly, $\{\}$ is an answer set for program (3) as well.

Representing a Problem in ASP The idea of ASP is to represent a computational problem as a program whose answer sets correspond to the solutions of the problem, and to find the answer sets for that program using an answer set solver.

When we represent a problem in ASP, two kinds of rules play an important role: those that "generate" many answer sets corresponding to "possible solutions", and those that can be used to "eliminate" the answer sets that do not correspond to solutions. Rules (3) are of the former kind: they generate the answer sets $\{p\}$ and $\{\}$. Constraints are of the latter kind. For instance, adding the constraint

$$\leftarrow p$$

to program (3) eliminates the answer sets for the program that contain p.

In ASP, we use special constructs of the form

$$\{A_1, \dots, A_n\}^c \tag{4}$$

(called *choice expressions*), and of the form

$$l \leq \{A_1, \dots, A_m\} \leq u \tag{5}$$

(called *cardinality expressions*) where each A_i is an atom and l and u are nonnegative integers denoting the "lower bound" and the "upper bound" (Simons et al. 2002). Programs using these constructs can be viewed as abbreviations for normal nested programs defined above, due to (Ferraris and Lifschitz 2005). For instance, the following program

$$\{p\}^c \leftarrow$$

stands for program (3). The constraint

$$\leftarrow\ 2 \leq \{p, q, r\}$$

stands for the constraints

$$\begin{array}{l}\leftarrow\ p, q \\ \leftarrow\ p, r \\ \leftarrow\ q, r\cdot\end{array}$$

Expression (4) describes subsets of $\{A_1, \ldots, A_n\}$. Such expressions can be used in heads of rules to generate many answer sets. For instance, the answer sets for the program

$$\{p, q, r\}^c\ \leftarrow \tag{6}$$

are arbitrary subsets of $\{p, q, r\}$. Expression (5) describes the subsets of the set $\{A_1, \ldots, A_m\}$ whose cardinalities are at least l and at most u. Such expressions can be used in constraints to eliminate some answer sets. For instance, adding the constraint

$$\leftarrow\ 2 \leq \{p, q, r\}$$

to program (6) eliminates the answer sets for (6) whose cardinalities are at least 2. Adding the constraint

$$\leftarrow\ \mathit{not}\ \ (1 \leq \{p, q, r\}) \tag{7}$$

to program (6) eliminates the answer sets for (6) whose cardinalities are not at least 1.

We abbreviate the rules

$$\begin{array}{l}\{A_1, \ldots, A_m\}^c \leftarrow \mathit{Body} \\ \leftarrow \mathit{not}\ \ (l \leq \{A_1, \ldots, A_m\}) \\ \leftarrow \mathit{not}\ \ (\{A_1, \ldots, A_m\} \leq u)\end{array}$$

by

$$l \leq \{A_1, \ldots, A_m\}^c \leq u \leftarrow \mathit{Body}\cdot$$

For instance, rules (6), (7) and $\leftarrow\ \mathit{not}\ \ (\{p, q, r\} \leq 1)$ can be written as

$$1 \leq \{p, q, r\}^c \leq 1 \leftarrow$$

whose answer sets are the singleton subsets of $\{p, q, r\}$.

Finding a Solution using an Answer Set Solver Once we represent a computational problem as a program whose answer sets correspond to solutions of the problem, we can use an answer set solver to compute the solutions of the problem. To present a program to an answer set solver, like CLASP, we need to make some syntactic modifications.

The syntax of the input language of CLASP is more limited in some ways than the class of programs defined above, but it includes many useful special cases. For instance, the head of a rule can be an expression of one of the forms

$$\begin{array}{l} \{A_1, \ldots, A_n\}^c \\ l \leq \{A_1, \ldots, A_n\}^c \\ \{A_1, \ldots, A_n\}^c \leq u \\ l \leq \{A_1, \ldots, A_n\}^c \leq u \end{array}$$

but the superscript ^{c} and the sign $\leq$ are dropped. The body can contain cardinality expressions but the sign $\leq$ is dropped.

In the input language of CLASP, :- stands for $\leftarrow$, and each rule is followed by a period.

A group of rules that follow a pattern can be often described in a compact way using "(schematic) variables". Variables must be capitalized. For instance, the program Π_n

$$p_i \leftarrow \textit{not} \ \ p_{i+1} \qquad (1 \leq i \leq n)$$

can be presented to CLASP as follows:

```
index(1..n).
p(I) :- not p(I+1), index(I).
```

Here `index` is a "domain predicate" used to describe the range of variable `I`.

Variables can be also used "locally" to describe the list of formulas in a cardinality expression. For instance, the rule

$$1 \leq \{p_1, \ldots, p_n\} \leq 1$$

can be expressed in CLASP as follows

```
index(1..n).
1{p(I) : index(I)}1.
```

CLASP finds an answer set for a program in two stages: first it gets rid of the schematic variables using a "grounder", like GRINGO, and then it finds an answer set for the ground program using a DPLL-like branch and bound algorithm (outlined in Algorithm 2).

3 Computational Problems

We study various problems related to finding similar/diverse solutions to a computational problem P formulated in ASP. For that, we assume that the problem is represented as a nonnested normal program $\mathcal{P}$ whose answer sets characterize solutions of the problem. More precisely, let $Sol(P)$ denote the set of solutions of P and let $\mathcal{AS}(\mathcal{P})$ denote the set of answer sets of $\mathcal{P}$. Then, there is a many-to-one mapping of $\mathcal{AS}(\mathcal{P})$ onto $Sol(P)$. Moreover, given an answer set of $\mathcal{P}$ the corresponding solution from $Sol(P)$ can efficiently be extracted. We also assume that a distance function that maps a set S of solutions to a number is defined, to measure how similar/diverse the solutions are in S. To this end, we consider set-distance measures $\Delta : 2^{Sol(P)} \mapsto \mathbb{N}_0$ on solutions for P.

We are mainly interested in 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 some 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 the ASP program $\mathcal{P}$ describes the phylogeny reconstruction problem for Indo-European languages as in (Brooks et al. 2005); so each answer set of $\mathcal{P}$ represents a phylogeny for Indo-European languages. Using this ASP program with an existing ASP solver, one can compute many phylogenies for the same input dataset and with the same input parameters. Instead of analyzing all of these phylogenies manually, a historical linguist may ask for, for instance, 3 phylogenies whose diversity is at least 20 with respect to some domain-independent or domain-dependent distance function Δ; this problem is an instance of n k-diverse solutions problem where $n = 3$ and $k = 20$. On the other hand, a historical linguist may have found two phylogenies P_1 and P_2 that are plausible, for instance, based on some archeological evidence, and she may want to infer a similar phylogeny whose distance from $\{P_1, P_2\}$ is at most 10; this problem is an instance of k-close solution problem where $k = 10$.

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.

> 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 $\Delta(S)$.
>
> MAXIMAL n k-SIMILAR SOLUTIONS (resp. MAXIMAL 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 a nonnegative integer k, find a $\subseteq$-maximal set S of at most n solutions for P such that $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$) exists.

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 closest (resp. most distant) solution.

> CLOSEST 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 $\Delta(S \cup \{s\})$.

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 ($S' \neq S$) exists such that $|\Delta(S) - \Delta(S')| \leq k$ (resp. $|\Delta(S) - \Delta(S')| \geq k$).

Usually an expert is interested in several kinds of problems to be able to systematically analyze solutions. For instance, a historical linguist may want to find 3 most diverse phylogenies; and after identifying one particular plausible phylogeny among them, she may want to compute another phylogeny that is the closest. An example of such an analysis is shown in Section 6.3 for understanding the classification of Indo-European languages.

4 Complexity Results

Before we discuss how the computational problems described in the previous section can be solved in ASP, let us turn our attention to the computational complexity of the problems presented in Section 3. In order to do so, we first make some reasonable assumptions on some of the problem parameters.

In the following we assume that given an answer set s of $\mathcal{P}$, extracting a solution of P from s can be accomplished in time polynomial wrt. the size of s. Moreover, w.l.o.g. we identify s with the solution it encodes, and sets $S \subseteq Sol(P)$ with corresponding sets of answer sets from $\mathcal{AS}(\mathcal{P})$.

We assume that all numbers are given in binary and that the given number n of different solutions to consider (respectively the size of the set S) for instances of the problems n k-SIMILAR SOLUTIONS, MAXIMAL n k-SIMILAR SOLUTIONS, and n MOST SIMILAR SOLUTIONS is polynomial in the size of the input. The same assumption applies to the size of the sets S' to consider in instances of k-CLOSE SET problems.

Furthermore, we consider distance measures Δ such that deciding whether $\Delta(S) \leq k$ (resp. whether $\Delta(S) \geq k$) 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). Thus, when considering Δ as an input to a problem, we assume that it is given as the description of a non-deterministic Turing machine $M_{\Delta}^{\leq}$, or $M_{\Delta}^{\geq}$, or both, where $M_{\Delta}^{\leq}$ (resp. $M_{\Delta}^{\geq}$) nondeterministically decides $\Delta(S) \leq k$ (resp. $\Delta(S) \geq k$) in time polynomial in the length of its input S and k. Consequently, a witness for a computation of M_{Δ}^{χ} on some input S and k, where $\chi \in \{\leq, \geq\}$ is a sequence of configurations of M_{Δ}^{χ}, such that the input tape contains S and k in the initial configuration, successive configurations correspond to transitions of M_{Δ}^{χ}, and the final configuration accepts. In addition, we say that a Δ is *normal* if $|S| \leq 1$ implies $\Delta(S) = 0$.

Under these assumptions, the computational complexity (cf. (Papadimitriou 1994) for a background on the subject) of the problems concerning the computation of similar/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

Table 1. *Complexity results for computing similar solutions.*

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

is replaced with DIVERSE, and CLOSE is replaced with DISTANT, respectively. The proofs are included in Appendix A.

Theorem 1
Problem n k-SIMILAR SOLUTIONS (resp. n k-DIVERSE SOLUTIONS) is NP-complete. Hardness holds even if $\Delta(S)$ is computable in constant time and for any normal Δ.

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 via the program $\mathcal{P}$ (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. For a hardness result resorting to partial Hamming distance, one can confer (Bailleux and Marquis 1999).

In our experiments with phylogeny reconstruction, by Theorem 1, we know that deciding the existence of n k-similar (resp. k-diverse) phylogenies is NP-complete, if the distance measure is the nearest neighbor interchange distance (DasGupta et al. 1997) whose computation is beyond polynomial time, or if the distance measure is the nodal distance or comparison of descendants distance (both defined in Section 6.1 that are computable in polynomial time. Also, in planning, if we consider the Hamming distance (Hamming 1950) (as defined in Section 7), which is polynomially computable, or the edit distance involving transpositions, which is conjectured to be NP-hard (Bafna and Pevzner 1998), deciding the existence of n k-similar (resp. k-diverse) plans is NP-complete. Therefore, it makes sense to find similar/diverse phylogenies/plans using ASP.

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 constant time and for any normal Δ.

When looking for maximal sets of solutions, we face a function problem; here we also assume a polynomial upper bound on the size of the sets S to consider (given by input n and our corresponding assumption). In particular, MAXIMAL n k-SIMILAR SOLUTIONS (resp. n MAXIMAL k-DIVERSE SOLUTIONS) is solvable in FNP//log. Intuitively, 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 whose associated decision problem is solvable in NP, provided that this value has logarithmically many bits in the size of the input (see, e.g., (Chen and Toda 1995; Eiter and Subrahmanian 1999) for more information on $\mathrm{FNP}//\log$ and other function classes used in this section).

Theorem 3

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

Membership can be shown by computing the maximum cardinality of a set of at most n 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 by a reduction of X-MinModel (cf. (Chen and Toda 1995)). We remark that the slightly different problem asking for a polynomial-size set S of solutions such that $\Delta(S)$ is minimal (respectively maximal), again under the assumption that this value has logarithmically many bits, is also $\mathrm{FNP}//\log$-complete. For this variant, hardness can be shown, e.g., for $\Delta(S)$ that takes the minimal (respectively maximal) 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.

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.

$\mathrm{FP}^{\mathrm{NP}}$-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 some suitable 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.

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

Theorem 5

Problem CLOSEST 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 constant time and for any normal Δ.

Discussion The results above, summarized in Table 1, show that computing similar solutions is intractable in general. This already holds under the reasonable assumption that the distance measure Δ is normal, where all considered decision problems are NP-complete.

The precise complexity characterization of the search problems (MAXIMAL n k-SIMILAR SOLUTIONS, n MOST SIMILAR SOLUTIONS, and CLOSEST SOLUTION) reveals some information about the type of algorithm we can expect to be suitable for solving these problems in practice (for background, see (Chen and Toda 1995) and references therein). In particular, we may not expect that they can be solved by parallelization to NP-problems in polynomial time, i.e., solve in parallel polynomially many NP-problems, e.g., SAT instances, and then combine the results. On the other hand, for problem MAXIMAL n k-SIMILAR SOLUTIONS this is possible under randomization, i.e., with high probability of a correct outcome, due to the characteristics of $\mathrm{FNP}//\log$, while this is not the case for the problems n MOST SIMILAR SOLUTIONS and CLOSEST SOLUTION in the general case. Rather, the results suggest that consecutive, dependent calls to NP oracles are needed. Intuitively, backtracking-style algorithms, which explore the search space to find solutions and then see to (dis)prove optimality by finding better solutions, appropriately reflect adaptivity.

However, from a worst-case complexity perspective, a simple realization of such a scheme may not be optimal, as far too many solution improvements (exponentially resp. polynomially many under "small" distance values) may happen until an optimal solution is found; here a two phase algorithm (first compute the optimal solution cost in binary search and then a solution of that cost, e.g., with backtracking) gives better guarantees. In practice, one may intertwine bound and solution computation and conduct a binary search over computations of solutions within a given bound.

In the next section, we consider first solving the search problem analog of the decision problem n k SIMILAR SOLUTIONS, using different approaches, ranging from declarative encodings in ASP over the explicit respectively implicit set of solutions, to a generalized backtracking algorithm for evaluation ASP programs. We then consider solving the related search problems n MOST SIMILAR SOLUTIONS and MAXIMAL n k-SIMILAR SOLUTIONS based on the above considerations.

5 Computing Similar/Diverse Solutions

Now we have a better understanding of the computational problems, let us present our computational methods to find n k-similar/diverse solutions, n most similar/diverse solutions and maximal n k-similar/diverse solutions for a given computational problem P. Since the computation of similar solutions and diverse solutions are symmetric, for simplicity, let us only focus on the problems related to similarity. In the following, suppose that the problem P is described by an ASP program `Solve.lp`.

```
Algorithm 1 Offline Method
Input: A set $S$ of solutions, a distance function $d : S \times S \mapsto \mathbb{N}$, and two nonnegative
  integers $n$ and $k$.
Output: A set $C$ of $n$ solutions whose distance is at most $k$.
  $V \leftarrow$ Define a set of $|S|$ vertices, each denoting a unique solution in $S$;
  $E = \{\{v_i, v_j\} \mid v_i \neq v_j,\ v_i, v_j$ denote $s_i, s_j \in S,\ d(s_i, s_j) \leq k\}$;
  $C \leftarrow$ Find a clique in of size $n$ in $\langle V, E \rangle$;
  return C
```

5.1 Computing n k-Similar Solutions

To compute a set of n solutions whose distance is at most k, we introduce an offline method and three online methods.

5.1.1 Offline Method

In the offline method, we compute the set S of all the solutions for P in advance using the ASP program `Solve.lp`, with an existing ASP solver. Then, we use some clustering method to find similar solutions in S. The idea is to form clusters of n solutions, measure the distance of each cluster, and pick the cluster whose distance is less than or equal to k.

We can compute clusters of n solutions whose distance is at most k by means of a graph problem: build a complete graph G whose nodes correspond to the solutions in S 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 weight of the clique) is less than or equal to k. The set of vertices in the clique represents n k-similar solutions.

The weight of a clique (or the distance Δ of the solutions in the cluster) can be computed as follows: Given a function d to measure the distance between two solutions, let $\Delta(S)$ be the maximum distance between any two solutions in S. Then n k-similar solutions can be computed by Algorithm 1 where the graph G is built as follows: nodes correspond to solutions in S, and there is an edge between two nodes s_1 and s_2 in G if $d(s_1, s_2) \leq k$. Nodes of a clique of size n in this graph correspond to n k-similar solutions. Such a clique can be computed using the ASP formulation in (Lifschitz 2008), or one of the existing exact/approximate algorithms discussed in (Gutin 2003).

5.1.2 Online Method 1: Reformulation

Instead of computing all the solutions in advance as in the offline method, we can compute n k-similar solutions to the given problem P on the fly. First we reformulate the ASP program `Solve.lp` in such a way to compute n-distinct solutions; let us call the reformulation as `SolveN.lp`. Such a reformulation can be obtained from `Solve.lp` as follows:

1. We specify the number of solutions: `solution(1..n).`
2. In each rule of the program `Solve.lp`, we replace each atom `p(T1,T2,...,Tm)`

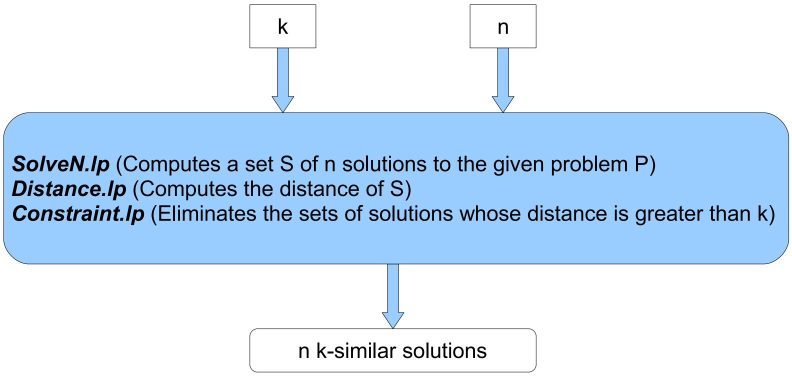
k
n
SolveN.lp (Computes a set S of n solutions to the given problem P)
Distance.lp (Computes the distance of S)
Constraint.lp (Eliminates the sets of solutions whose distance is greater than k)
n k-similar solutions

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

(except the ones specifying the input) with `p(N,T1,T2...,Tm)`, and add to the body `solution(N).`

3. Now we have a program that computes n solutions. To ensure that they are distinct, we add a constraint which expresses that every two solutions among these n solutions are different from each other.

Next we describe the distance function Δ as an ASP program, `Distance.lp`. In addition, we represent the constraints on the distance function (e.g., the distance of the solutions in S is at most k) as an ASP program `Constraint.lp`. Then we can compute n-distinct solutions for the given problem P that are k-similar, by one call of an existing ASP solver with the program `SolveN.lp` $\cup$ `Distance.lp` $\cup$ `Constraint.lp`, as shown in Fig. 1. Let us give an example to illustrate the offline method.

Example 1

Suppose that we want to compute n k-similar cliques in a graph. Assume that the similarity of two cliques is measured by the Hamming Distance: the distance between two cliques C and C' is equal to the number of different vertices, $(C \setminus C') \cup (C' \setminus C)$. The distance of a set S of cliques can be defined as the maximum distance among any two cliques in S.

The clique problem can be represented in ASP (`Solve.lp`) as in (Lifschitz 2008), also shown in Appendix B (Fig. B 1). The reformulation (`SolveN.lp`) of this ASP program as described above can be seen in Fig. B 2 of Appendix B. This reformulation computes n distinct cliques.

The Hamming Distance between any two cliques can be represented by the ASP program (`Distance.lp`) shown in Fig. B 3 of Appendix B.

Finally, Fig. B 4 shows the constraint (`Constraint.lp`) that eliminates the sets whose distance is above k.

An answer set for the union of these three programs, `SolveN.lp` $\cup$ `Distance.lp` $\cup$ `Constraint.lp`, corresponds to n k-similar cliques.

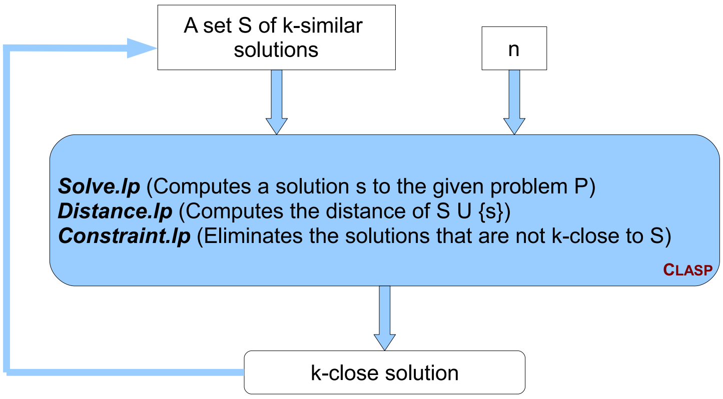
A set S of k-similar solutions
n
Solve.lp (Computes a solution s to the given problem P)
Distance.lp (Computes the distance of S U {s})
Constraint.lp (Eliminates the solutions that are not k-close to S)
CLASP
k-close solution

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.

5.1.3 *Online Method 2: Iterative Computation*

This method does not modify the given ASP program `Solve.lp` as in Online Method 1, but still formulates the distance function and the distance constraints as ASP programs. The idea is to find similar/diverse solutions iteratively, where the i'th solution is k-close to the previously computed $i-1$ solutions (Fig. 2). Here n iterations lead to n solutions whose distance is at most k (i.e., n k-similar solutions).

Note that, like Offline Method and Online Method 1, this method is sound; however, unlike Offline Method and Online Method 1, it is not complete since computation of each solution depends on the previously computed solutions. The method may not return n k-similar solutions (even it exists) if the previously computed solutions comprise a bad solution set.

5.1.4 *Online Method 3: Incremental Computation*

This method is different from the other two online methods in that it does not modify the ASP program `Solve.lp` describing the given computational problem P, it does not formulate the distance function Δ and the distance constraints as ASP programs. Instead, modifies the search algorithm of an existing ASP solver in such a way that the modified ASP solver can compute n k-similar solutions (Fig. 3). In this method, we modify the search algorithm of the ASP solver CLASP (Version 1.1.3); the modified version is called CLASP-NK. The given distance measure Δ is implemented as a C++ program.

Let us describe how we modified CLASP to obtain CLASP-NK. CLASP performs a DPLL-like (Davis et al. 1962; Marques-Silva and Sakallah 1999) branch and bound search to find an answer set for a given ASP program (Algorithm 2): at each level, it "propagates" some literals to be included in the answer set, "selects" new literals to branch on, or "backtracks" to an earlier appropriate point in search while "learning conflicts" to avoid redundant search.

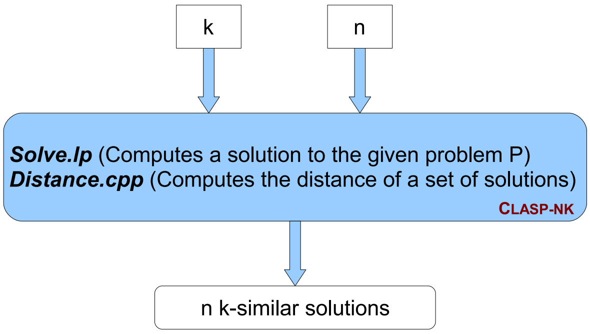
k
n
Solve.lp (Computes a solution to the given problem P)
Distance.cpp (Computes the distance of a set of solutions)
CLASP-NK
n k-similar solutions

Fig. 3. Computing n k-similar solutions, with Online Method 3. CLASP-NKis a modification of the ASP solver CLASP, that takes into account the distance function and constraints while computing an answer set in such a way that CLASP-NK becomes biased to compute similar solutions. Each computed solution is stored by CLASP-NK until a set of n k-similar phylogenies is computed.

Algorithm 2 CLASP

```
Input: An ASP program $\Pi$
Output: 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
    PROPAGATION($\Pi, A, \bigtriangledown$)    // propagate literals
    if There is a conflict in the current assignment then
      RESOLVE-CONFLICT($\Pi, A, \bigtriangledown$)    // learn and update conflicts, and backtrack
    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
```

We modify CLASP's algorithm as shown in Algorithm 3 to obtain CLASP-NK: the red parts show these modifications. that is able to compute n k-similar phylogenies, and call it CLASP-NK. To use CLASP-NK, one needs to prepare an options file, *NKoptions*, to describe the input parameters to compute n k-similar phylogenies, such as the values n and k, along with the names of predicates that characterize solutions and that are considered for computing the distance between solutions. Note that since an answer set (thus a solution) is computed incrementally in CLASP-NK, we cannot compute the distance between a partial solution and a set of solutions with respect to the given distance function Δ. Instead, one needs to implement a heuristic function to estimate a lower bound for the distance between any completion s of a partial solution with a set S of previously computed solutions. If this heuristic function is admissible then it does not underestimate the distance of $S \cup \{s\}$

Algorithm 3 CLASP-NK

```
Input: An ASP program $\Pi$, nonnegative integers $n$, and $k$
Output: A set $X$ of $n$ solutions that are $k$ similar ($n$ $k$-similar solutions)
  $A \leftarrow \emptyset$   // current assignment of literals
  $\bigtriangledown \leftarrow \emptyset$   // set of conflicts
  $X \leftarrow \emptyset$   // computed solutions
  while $|X| < n$ do
    $PartialSolution \leftarrow A$
    $LowerBound \leftarrow$ DISTANCE-ANALYZE$(X, PartialSolution)$   // compute a lower bound for the distance between any completion of a partial solution and the set of previously computed solutions
    PROPAGATION$(\Pi, A, \bigtriangledown)$   // propagate literals
    if Conflict in propagation OR $LowerBound > k$ then
      RESOLVE-CONFLICT$(\Pi, A, \bigtriangledown)$   // learn and update conflicts, and backtrack
    else
      if Current assignment does not yield an answer set then
        SELECT$(\Pi, A, \bigtriangledown)$   // select a literal to continue search
      else
        $X \leftarrow X \cup \{A\}$
        $A \leftarrow \emptyset$
      end if
    end if
  end while
  return X
```

(i.e., it returns a lower bound that is less than or equal to the optimal lower bound for the distance).

Note that similar to Online Method 2, this method is also sound but not complete since each solution depends on all previously computed solutions.

5.2 *Computing n Most Similar Solutions*

Now we have described some computational methods to solve the decision problem n k-SIMILAR SOLUTIONS, let us discuss how we can solve the optimization problem n MOST SIMILAR SOLUTIONS. Let $NKSimilar(n, k)$ be a function that returns—with one of the methods described in the previous subsections—a set S of n solutions which is k-similar; or returns empty set if no such set exists. Using this function, we can find n most similar solutions as follows: First we compute a lower bound and an upper bound for the distance k of a set of n solutions. Then, we perform a binary search within these bounds to find a set S of solutions with the optimal value for k. Computations of a lower bound and an upper bound are usually specific to the particular problem.

5.3 Computing Maximal n k-Similar Solutions

Another optimization problem we are interested in is MAXIMAL n k-SIMILAR SOLUTIONS, which asks for a maximal set of solutions whose distance is at most k. We can solve this problem by modifying Online Method 2: start with a solution (computed using `Solve.lp`), then repeatedly find a solution which is k-close to the previously found solutions until there does not exists such a solution. Recall that Online Method 2 iterates n times; here the iterations continue until no k-close solution is found.

6 Computing Similar/Diverse Phylogenies

Let us now illustrate the usefulness of our methods in a real-world application: reconstruction of evolutionary trees (or phylogenies) of a set of species based on their shared traits. This problem is important for research areas as disparate as genetics, historical linguistics, zoology, anthropology, archeology, etc.. For example, a phylogeny of parasites may help zoologists to understand the evolution of human diseases (Brooks and McLennan 1991); a phylogeny of languages may help scientists to better understand human migrations (White and O'Connell 1982).

There are several software systems, such as PHYLIP (Felsenstein 2009), PAUP (Swofford 2003) or PHYLO-ASP (Erdem 2009), that can reconstruct a phylogeny for a set of taxonomic units, based on "maximum parsimony" (Edwards and Cavalli-Sforza 1964) or "maximum compatibility" (Camin and Sokal 1965) criterion. With some of these systems, such as PHYLO-ASP, we can compute many good phylogenies (most parsimonious phylogenies, perfect phylogenies, phylogenies with most number of compatible traits, etc.) according to the phylogeny reconstruction criterion. In such cases, in order to decide the most "plausible" ones, domain experts manually analyze these phylogenies, since there is no available phylogenetic system that can analyze/compare these phylogenies.

For instance, PHYLO-ASP computes 45 plausible phylogenies for the Indo-European languages based on the dataset of (Brooks et al. 2007). In order to pick the most plausible phylogenies, in (Brooks et al. 2007), the historical linguist Don Ringe analyzes these phylogenies by trying to cluster these phylogenies into diverse groups, each containing similar phylogenies. In such a cases, having a tool that reconstructs similar/diverse solutions would be useful: with such a tool, an expert can compute (instead of computing all solutions) few most diverse solutions, pick the most plausible one, and then compute phylogenies that are close to this phylogeny.

Let us show how our methods can be used for this purpose. Before that, we define a phylogeny and some distance functions to measure the similarity/diversity of phylogenies.

6.1 Distance Measures for Phylogenies

A *phylogeny* for a set of taxa is a finite rooted leaf-labeled binary tree $\langle V, E\rangle$ with a set L of leaves ($L \subseteq V$). 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 labelings of leaves denote the values of shared traits at those nodes. We consider distance measures that depend on topologies of phylogenies, therefore, while defining them we discard these labelings.

There are various measures to compute the distance between two phylogenies (Nye et al. 2006; Robinson and Foulds 1981; Hon et al. 2000; Kuhner and Felsenstein 1994; DasGupta et al. 1997). In the following, first we consider one of these domain-independent functions, the nodal distance measure (Bluis and Shin 2003), to compare two phylogenies; and then we define a distance measure for a set of phylogenies based on the nodal distances of pairwise phylogenies, to show the applicability of our methods for finding n k-similar phylogenies. Then we define a novel distance function that measures the distance of two phylogenies, and a distance function that measures the distance of a set of phylogenies, taking into account some expert knowledge specific to evolution. With this measure we also show the effectiveness of our methods.

6.1.1 Nodal Distance of Two Phylogenies

The *nodal distance* $\mathit{ND}_P(x,y)$ *of two leaves* x and y in a phylogeny P is defined as follows: First, transform the phylogeny P to an undirected graph G where there is an undirected edge $\{i,j\}$ in the graph for each directed edge (i,j) in the phylogeny. Then $\mathit{ND}_P(x,y)$ is equal to the length of the shortest path between x and y in the undirected graph G. For example, consider the phylogeny, P_1 in Fig. 4; 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 phylogeny represents the degree of their relationship in that phylogeny.

Given two phylogenies P_1 and P_2 both with same set L of leaves, the *nodal distance* $D_n(P_1,P_2)$ *of two phylogenies* is calculated as follows:

$$D_n(P_1,P_2) = \sum_{x,y \in L} |\mathit{ND}_{P_1}(x,y) - \mathit{ND}_{P_2}(x,y)|\cdot$$

Here the difference of the nodal distances of two leaves x and y represents the contribution of this pair of leaves to the distance between the phylogenies.

Proposition 1
Given two phylogenies P_1 and P_2 with same set L of leaves and the same leaf-labeling function, $D_n(P_1,P_2)$ can be computed in $O(|L|^3)$ time.

Table 2 shows an example of computing the nodal distance between two phylogenies. Here, the phylogenies are presented in the Newick format, where the sister sub-phylogenies are enclosed by parentheses. For instance, the first tree, P_1, of Fig. 4 can be represented in the Newick format as $(a,(b,c))$.

6.1.2 Descendant Distance of Two Phylogenies

Nodal distance measure computes the distance between two rooted binary trees and does not consider the evolutionary relations between nodes. In that sense, it is a domain-independent distance measure for comparing phylogenies. A distance measure that takes into account these relations and might give more accurate results. Therefore, we define a new distance function based on our discussions with the historical linguist Don Ringe. In particular, we

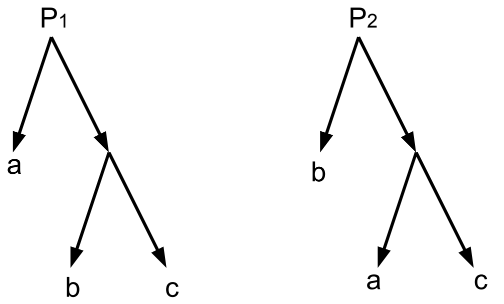

Fig. 4. Two phylogenies $P_1 = (a, (b, c))$ and $P_2 = (b, (a, c))$

Table 2. *In order to compute the nodal distance $D_n(P_1, P_2)$ between the phylogenies $P_1 = (a, (b, c))$ and $P_2 = (b, (a, c))$ shown in Fig. 4, we compute the nodal distances of the pairs of leaves, $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$, and take the sum of the differences. In this case the distance between P_1 and P_2 is 2.*

Pairs of leaves	Distance in P_1	Distance in P_2	Difference
{a,b}	3	3	0
{a,c}	3	2	1
{b,c}	2	3	1
Total distance			2

take into account the following domain-specific information in phylogenetics: the similarities of phylogenies towards their roots are more significant; and thus two phylogenies are more similar if the diversifications closer to their roots are more similar.

For each vertex v of a tree $\langle V, E\rangle$, let us define the descendants of x as follows:

$$\mathit{desc}(v) = \begin{cases} \{v\} & v \text{ is a leaf} \\ \mathit{desc}(u) \cup \mathit{desc}(u') & \text{otherwise } (v, u), (v, u') \in E, u \neq u' \end{cases}$$

and the depth of a vertex v as follows:

$$\mathit{depth}(v) = \begin{cases} 0 & v \text{ is the root} \\ 1 + \mathit{depth}(u) & \text{otherwise}(u, v) \in E\cdot \end{cases}$$

For every depth i ($0 \leq i \leq \mathit{max}_{v\in V}\mathit{depth}(v)$), we define the sets of vertices "covered" by the vertices at depth i:

$$\mathit{covered}(i) = \{\mathit{desc}(v) \mid \mathit{depth}(v) = i\}\cdot$$

and we define a weight function $\mathit{weight}(i)$ that assigns a number to each depth i. The idea is to assign bigger weights to smaller depths so that two phylogenies are more similar if the

Table 3. *In order to compute the descendant distance $D_l(P_1, P_2)$ between the phylogenies $P_1 = (a, (b, c))$ and $P_2 = (b, (a, c))$ shown in Fig. 4, for each depth level, we multiply the number of vertices that have different descendants with the weight of that depth level. Then, we add up the products to find the total distance between P_1 and P_2. The descendant distance between P_1 and P_2 is 4.*

Depth	Weight P_1	Number of pairs of vertices that have different descendant sets
0 (root)	2	0
1	1	4
2	0	3

Distance = $2 \times 0 + 1 \times 4 + 0 \times 3 = 4$

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.

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 $v' \in V'$:

$$f(v, v') = \begin{cases} 1 & \mathit{desc}(v) \neq \mathit{desc}(v') \\ 0 & \text{otherwise} \end{cases}$$

Let *weight* be a function mapping every depth to a number. 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 ($0 \leq i \leq \min\{\max_{v\in V} \mathit{depth}(v), \max_{v'\in V'} \mathit{depth}(v')\}$), by the following measure:

$$\begin{aligned} g(0, T, T') = {} & \mathit{weight}(0) \times f(R, R') \\ g(i, T, T') = {} & g(i-1, T, T') + \\ & \mathit{weight}(i) \times \textstyle\sum_{x\in V, y\in V', \mathit{depth}(x)=\mathit{depth}(y)=i} f(x, y), \quad i > 0\cdot \end{aligned}$$

and the similarity of two trees as follows:

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

Proposition 2
Given two trees P_1 and P_2 with same set L of leaves and the same leaf-labeling function, $D_l(P_1, P_2)$ can be computed in $O(|L|^3)$ time.

Table 3 shows an example of computing the distance between two trees shown in Fig. 4.

6.1.3 Distance of a Set of Phylogenies

In the previous subsections, we defined distance functions for measuring the distance between two phylogenies. However, the problems that we defined in Section 3 requires a distance function that measures the distance of a set of phylogenies. We can define the distance of a set of phylogenies based on the distances among pairwise phylogenies. For instance, the distance of a set S of phylogenies can be defined as the maximum distance among any two phylogenies in S.

Let D be one of the distance measures defined in the previous subsection, then the distance of a set S of phylogenies (Δ_D) is defined as follows:

$$\Delta_D(S) = \max\{D(P_1, P_2) \mid P_1, P_2 \in S\}$$

6.2 Computing n k-Similar/Diverse Phylogenies

Analogous to the n k-similar (resp. diverse) solutions, we define the n k-similar (resp. diverse) phylogenies as follows:

> n k-SIMILAR PHYLOGENIES (RESP. n k-DIVERSE PHYLOGENIES)
> Given an ASP program $\mathcal{P}$ that formulates a phylogeny reconstruction problem P, a distance measure Δ_D that maps a set of phylogenies for P to a nonnegative integer, and two nonnegative integers n and k, decide whether a set S of n phylogenies exists such that $\Delta_D(S) \leq k$ (resp. $\Delta_D(S) \geq k$).

Recall that in order to compute n k-similar (resp. diverse) solutions we need an ASP program that computes a solution and a distance measure. We consider the ASP program `phylogeny-improved.lp` described in (Brooks et al. 2007) as our main program that computes a phylogeny. We represent the nodal distance D_n (resp. the descendant D_l) of two phylogenies as the ASP program in Fig. B 8 (resp. Figs. B 9 and B 10) in Appendix B. In addition, we consider the program in Fig. B 11 that computes the total distance of a set of solutions with Δ_D and eliminates the ones whose total distance is greater than k.

For Offline Method, we compute all the phylogenies using `phylogeny-improved.lp`. Then we build a graph of phylogenies as in Subsection 5.1.1. Then, we use the ASP program in Fig. B 1 in Appendix B to find a clique of size n in the constructed graph. This clique corresponds to n k-similar phylogenies.

For Online Method 1, we reformulate the main program `phylogeny-improved.lp` to obtain a program that computes n distinct phylogenies as in Section 5.1.2. The reformulation is shown in Fig.s B 5–B 7 in Appendix B.

For Online Method 3, we define a heuristic function to estimate a low bound for the distance between any completion of a given partial phylogeny and a complete phylogeny.

Let P_c be any complete phylogeny, P_p be any partial phylogeny and L_p be the set of pairs of leaves that appear in P_p. Consider the nodal distance (Section 6.1.1) for comparing two phylogenies. Then we can define a lower bound as follows:

$$\mathcal{LB}_n(P_p, P_c) = \sum_{x,y \in L_p} |\mathit{ND}_{P_c}(x, y) - \mathit{ND}_{P_p}(x, y)|\cdot$$

This lower bound does not overestimate the distance between a phylogeny and any completion of a partial phylogeny.

Proposition 3
Given a partial phylogeny P_p and a complete phylogeny P_c, $\mathcal{LB}_n(P_p, P_c)$ is admissible, i.e., for every completion P of P_p,

$$\mathcal{LB}_n(P_p, P_c) \leq D_n(P, P_c)\cdot$$

Similarly, we can define an upper bound for the differences of nodal distances measure as follows:

$$\mathcal{UB}_n(P_p, P_c) = \sum_{x,y \in L_P} |ND_{P_c}(x,y) - ND_{P_p}(x,y)| + (\binom{l}{2} - \binom{|L_p|}{2}) \times l\cdot$$

where let l denotes the number of leaves in the complete tree.

This upper bound does not underestimate the distance between a phylogeny and any completion of a partial phylogeny.

Proposition 4
Given a partial phylogeny P_p and a complete phylogeny $P_c, \mathcal{UB}_n(P_p, P_c)$ is admissible, i.e., for every completion P of $P_p, \mathcal{UB}_n(P_p, P_c) \geq D_n(P, P_c)\cdot$

As regards the comparison of descendants distance measure, we could not find a tight lower and upper bounds. In our experiments, we consider that the lower bound (resp. upper bound) between a complete phylogeny and any completion of a partial phylogeny is 0 (resp. ∞).

6.3 Experimental Results for Phylogeny Reconstruction

We applied the offline method and the online methods described in Section 5.1 to reconstruct similar/diverse phylogenies for Indo-European languages. We used the dataset assembled by Don Ringe and Ann Taylor (Ringe et al. 2002). As in (Brooks et al. 2007), to compute similar/diverse 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.

In our experiments, we considered the distance measures described in Section 6.1 as in Section 6.2.

Below 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).

Experiments with the Nodal Distance Let us first examine the results of experiments, considering the distance measure Δ_n, based on the nodal distance (Table 4). 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, 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 takes as input an ASP representation of computing n k-similar/diverse phylogenies, which is almost n times as large as the ASP program describing the phylogeny reconstruction problem used in other methods. Therefore, its computational performance may not be as good as the other online methods. Online Method 2 relaxes this requirement a little bit so that the answer set

Table 4. *Computing similar/diverse phylogenies using the nodal distance* Δ_n.

Problem	Offline Method	Online Methods		
		Reformulation (CLASP)	Iterative Comp. (CLASP, perl)	Incremental Comp. (CLASP-NK)
2 most similar ($k = 12$)	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 ($k = 32$)	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 ($k = 15$)	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 ($k = 26$)	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 ($k = 25$)	11.66sec. 32MB $k = 25$	327.28 sec. 1.8GB $k = 25$	178.96 sec. 1.2GB $k = 29$	1.96 sec. 15MB $k = 25$

solver can compute the solutions more efficiently: it takes as input 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. In that sense, its computational performance is better than Online Method 2.

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 3. 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 finding optimal solutions 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.

Experiments with the Nodal Distance Now, let us consider the distance measures Δ_l, based on preference over diversifications (Table 5): 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 5, 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 4, the offline method takes more time/space to compute similar/diverse solutions; this is due to the computation of distances with respect to Δ_l which requires summations. Other results are similar to the ones presented in Table 4.

Accuracy Let us compare the phylogenies computed by different distance measures in terms of accuracy. In (Brooks et al. 2007), 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_l comply with these groupings. For instance, the 2 most similar phylogenies computed by Online Method 1 are in Group 1;

(AN, (TO, (IC, ((GE, AL), (GA, (IIR, BS)))))),
(AN, (TO, (IC, ((GE, AL), (BS, (IIR, GA)))))),

Phylogenies 7 and 8 of (Brooks et al. 2007); both are in Group 1. The 3 most diverse phylogenies computed by Online Method 2 are

(AN, (TO, (IC, (AL, (GE, (GA, (IIR, BS))))))),
(AN, (TO, (AL, (IC, (GE, (GA, (IIR, BS))))))),
(AN, (TO, ((GE, (GA, (IIR, BS))), (AL, IC)))),

Phylogenies 10, 1, 40 of (Brooks et al. 2007); all in different groups. Likewise, the 6 similar phylogenies computed by our methods fall into Group 2.

These results (in terms of computational efficiency and accuracy) show the effectiveness of our methods in phylogeny reconstruction: we can automatically compare many phylogenies in detail.

Table 5. *Computing similar/diverse phylogenies using the descendant distance Δ_l.*

Problem	Offline Method	Online Methods		
		Reformulation (CLASP)	Iterative Comp. (CLASP, perl)	Incremental Comp. (CLASP-NK)
2 most similar $(k = 20)$	365.16 sec. 4.2GB $k = 18$	16.11 sec. 236MB $k = 18$	16.23 sec. 212MB $k = 18$	0.635 sec. 22MB $k = 18$
3 most diverse $(k = 20)$	368.59 sec. 4.2GB $k = 20$	46.11 sec. 659MB $k = 20$	44.21 sec. 430MB $k = 20$	1.014 sec. 22MB $k = 20$
6 most similar $(k = 18)$	368.45 sec. 4.2GB $k = 18$	137.31 sec. 1.8GB $k = 18$	212.59 sec. 1.1GB $k = 18$	0.685 sec. 22MB $k = 20$

7 Computing Similar/Diverse Plans

In order to show the applicability and effectiveness of our methods to other domains, we extend our experiments further with the Blocks World planning problems.

In these experiments, we study the following instance of n k-SIMILAR SOLUTIONS (resp. n k-DIVERSE SOLUTIONS):

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

We take $\mathcal{P}$ as the ASP formulation of the non-concurrent Blocks World as in (Erdem 2002) to compute a plan of length at most l (Fig. B 12 in Appendix B), together with an ASP description of the planning problem instance shown in Fig. 5.

We define the distance $\Delta_h(S)$ of a set S of plans as follows:

$$\Delta_h(S) = \max\{D_h(P_1, P_2) \mid P_1, P_2 \in S, |P_1| \leq |P_2|\}$$

based on the action-based Hamming distance D_h defined in (Srivastava et al. 2007) to measure the distance between two plans. Intuitively, $D_h(P_1, P_2)$ is the number of differentiating actions in each time step of two plans P_1 and P_2. More precisely: let us denote a plan X of length l by a function act_X that maps every nonnegative integer i ($1 \leq i \leq l$) to the i'th action of the plan X, and let us denote by $|X|$ the length of a plan X; then the Hamming Distance $D_h(P_1, P_2)$ between two plans P_1 and P_2 such that $|P_1| \leq |P_2|$ can be defined as follows:

$$D_h(P_1, P_2) = |\{i \mid act_{P_1}(i) \neq act_{P_2}(i), 1 \leq i \leq |P_1|\}| + |P_2| - |P_1|$$

ASP formulations of the distance functions D_h and $\Delta_h(S)$ are presented in Fig.s B 14 and B 15 in Appendix B.

Consider, for instance, a planning problem in the Blocks World that asks for a plan of length less than or equal to 7. Consider two plans, P_1 and P_2, that are characterized by the functions act_{P_1} and act_{P_2} respectively, as follows:

$$\begin{array}{ll} act_{P_1}(1) = moveop(b_1, Table) & act_{P_1}(2) = moveop(b_2, b_1) \\ act_{P_1}(3) = moveop(b_4, Table) & act_{P_1}(4) = moveop(b_3, b_2) \\ act_{P_1}(5) = moveop(b_4, b_3) & act_{P_1}(6) = moveop(b_5, b_4) \end{array}$$

$$\begin{array}{ll} act_{P_2}(1) = moveop(b_1, Table) & act_{P_2}(2) = moveop(b_2, b_1) \\ act_{P_2}(3) = moveop(b_4, b_5) & act_{P_2}(4) = moveop(b_3, b_2) \\ act_{P_2}(5) = moveop(b_4, Table) & act_{P_2}(6) = moveop(b_4, b_3) \\ act_{P_2}(7) = moveop(b_5, b_4) & \end{array}$$

The distance $D_h(P_1, P_2)$ between P_1 and P_2 is 4 since the actions at time steps 3, 5 and 6 are different and P_2 has an additional action (at time step 7).

To be able to apply our Online Method 3 with CLASP-NK to compute n k-similar plans of length at most l, we define a heuristic function $\mathcal{LB}_h$ to estimate a lower bound for the distance between a plan P_c and any plan-completion of a "partial" plan P_p. Intuitively, a partial plan consists of parts of a plan. Let us characterize a partial plan P_p by a partial function act_{P_p} from $\{1, \cdots, l\}$ to the set of actions; that is, act_{P_p} is a function from a subset of $\{1, \cdots, l\}$ to the set of actions. A *plan-completion of a partial plan* P_p is a plan Y of length l' ($l' \leq l$) for the planning problem P such that act_Y is an extension of act_{P_p} to $\{1, \cdots, l'\}$. Then we can define $\mathcal{LB}_h(P_p, P_c)$ for a partial plan P_p and a plan P_c as follows:

$$\begin{array}{rl} \mathcal{LB}_h(P_p, P_c) = & |\{i \mid act_{P_p}(i) \neq act_{P_c}(i), i \in \text{dom}\, act_{P_p}, 1 \leq i \leq |P_c|\}| \\ & + |\{i \mid i \in \text{dom}\, act_{P_p}, |P_c| < i \leq l\}| \end{array}$$

In the Blocks World example above, consider a partial plan P_p characterized by the function act_{P_p} as follows:

$$\begin{array}{ll} act_{P_p}(2) = moveop(b_2, b_1) & act_{P_p}(4) = moveop(b_3, b_2) \\ act_{P_p}(5) = moveop(b_4, Table) & act_{P_p}(7) = moveop(b_5, b_4) \end{array}$$

The lower bound $\mathcal{LB}_h(P_p, P_1)$ for the distance between any completion of P_p and P_1 is computed as follows:

$$\begin{array}{rcl} \mathcal{LB}_h(P_p, P_1) & = & |\{i \mid act_{P_p}(i) \neq act_{P_1}(i), i \in \text{dom}\ act_{P_p}, 1 \leq i \leq 6\}| \\ & & + |\{i \mid i \in \text{dom}\ act_{P_p}, 6 < i \leq 7\}| \\ & = & |\{5\}| + |\{7\}| = 2\cdot \end{array}$$

One completion of P_p is P_2. Note that $\mathcal{LB}_h(P_p, P_1) \leq D_h(P_1, P_2)$. Indeed, the following proposition expresses that $\mathcal{LB}_h$ does not overestimate the distance between P_c and any plan-completion X of P_p.

Proposition 5
For a partial plan P_p and a plan P_c for the planning problem P, $\mathcal{LB}_h(P_p, P_c)$ is admissible.

Similarly, to be able to apply our Online Method 3 with CLASP-NK to compute n k-diverse plans of length at most l, we define a heuristic function $\mathcal{UB}_h(P_p, P_c)$ to estimate an upper bound for the distance between a plan P_c and any plan-completion of P_p:

$$\mathcal{UB}_h(P_p, P_c) = l - |\{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}, 1 \leq i \leq |P_c|\}|\cdot$$

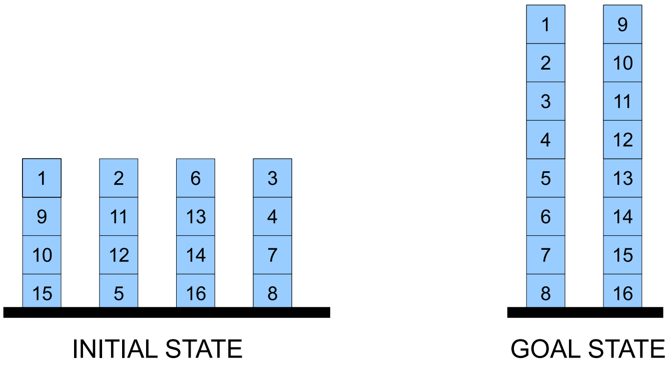

Fig. 5. Blocks World problem.

For instance, for the partial plan P_p and P_1 above,

$$\begin{aligned}\mathcal{UB}_h(P_p, P_1) &= 7 - |\{i \mid \mathit{act}_{P_p}(i) = \mathit{act}_{P_c}(i), i \in \mathrm{dom}\, \mathit{act}_{P_p}, 1 \leq i \leq 6\}| \\ &= 7 - |\{2, 4\}| = 7 - 2 = 5\end{aligned}$$

and $\mathcal{UB}_h(P_p, P_1) \geq D_h(P_1, P_2)$. Indeed, the following proposition expresses that this upper bound function does not underestimate the distance between P_c and any plan-completion X of P_p.

Proposition 6
For a partial plan P_p and a plan P_c for the planning problem P, $\mathcal{UB}_h(P_p, P_c)$ is admissible.

We performed some experiments with the ASP formulation, planning problem, and distance measures above, to find 2 most similar plans, 2 most diverse plans, 3 most similar plans, 3 most diverse plans, 6 most similar plans. Table 6 summarizes the results of these experiments.

It can be observed that the planning problem in Fig. 5 has too many solutions to the problem (more than 50.000), and it is intractable to compute all of them in advance and then the distances between all pairwise solutions. Therefore, instead of computing all the solutions in advance, we compute a subset of them (around 200) which is small enough to construct a distance graph, and apply our Offline Method in this way. However, these 200 solutions are not diverse enough, and thus, although we can find many very similar solutions, it is hard to find diverse solutions; for instance, we can find 6 1-similar solutions but we can find only 3 6-diverse solutions.

Online Method 1 performs the worst in comparison with the other online methods, as in our experiments with phylogeny reconstruction problems, due to the large ASP program (Fig. B 13 in Appendix B) used for computing n distinct plans.

Online Method 2 is comparable with Online Method 3 in terms of computing similar solutions. After computing a solution, computing a 1-similar solutions has a very small search space and CLASP can find a similar solution in a short time. On the other hand, computing a 21-diverse solution has a huge search space. Therefore, performance of computing diverse solutions with Online Method 2 is worse than that of Online Method 3.

Table 6. *Computing similar/diverse plans for the blocks world problem. OM denotes "Out of memory."*

Problem	Offline Method	Online Methods		
		Reformulation (CLASP)	Iterative Comp. (CLASP, perl)	Incremental Comp. (CLASP-NK)
2 most similar ($k = 1$)	- OM -	6 min. 45 sec. 106 MB $k = 1$	6 min. 53 sec. 73 MB $k = 1$	7 min. 17 sec. 111 MB $k = 1$
2 most diverse ($k = 22$)	- OM -	33 min. 28 sec. 213 MB $k = 22$	11 min. 73 MB $k = 22$	7 min. 40 sec. 112 MB $k = 21$
3 most similar ($k = 1$)	- OM -	7 min. 5 sec. 141 MB $k = 1$	7 min. 3 sec. 73 MB $k = 1$	7 min. 21 sec. 112 MB $k = 2$
3 most diverse ($k = 22$)	- OM -	78 min 42 sec. 333 MB $k = 22$	18 min. 49 sec. 73 MB $k = 21$	12 min. 40 sec. 167 MB $k = 21$
6 most similar ($k = 1$)	- OM -	64 min. 42 sec. 584 MB $k = 1$	7 min. 32 sec. 73 MB $k = 1$	7 min. 18 sec. 112 MB $k = 2$

Online Method 3 deals with the Hamming distance computation at the search level. In addition, it does not restart the search process to compute a new solution; instead, it learns the conflicts caused by distance difference while computing a new solution and backtracks to approximate levels to compute similar/diverse solutions. Especially, for the computation of diverse solutions, such a search strategy creates a significant performance gain.

8 Related Work

Finding similar/diverse solutions has been studied in other areas such as propositional logic (Bailleux and Marquis 1999), constraint programming (Hebrard et al. 2007; Hebrard et al. 2005), and planning (Srivastava et al. 2007). Let us briefly describe related work in each area, and discuss the similarities and the differences compared with our approach.

Related Work in Propositional Logic In (Bailleux and Marquis 1999), Bailleux and Marquis study the following problem

> DISTANCE-SAT
> Given a CNF formula Σ, a partial interpretation PI, and a nonnegative integer d, decide whether there is a model I of Σ such that I disagrees with PI on at most d atoms.

This problem is similar to k-CLOSE SOLUTION in that it asks for a k-close solution. On the other hand, it asks for a solution k-close to a partial solution, whereas k-CLOSE SOLUTION asks for a solution that is k-close to a set of previously computed solutions. Also, DISTANCE-SAT considers a distance measure (i.e., partial Hamming distance) computable in polynomial time; whereas k-CLOSE SOLUTION considers any distance measure such that deciding whether the distance of a set of solutions is less than a given k is in NP. Despite these differences, with S containing a single solution and Δ being a partial Hamming distance, k-CLOSE SOLUTION becomes essentially the same as DISTANCE-SAT.

As for the computational complexity analysis, Proposition 1 of (Bailleux and Marquis 1999) shows that in the general case DISTANCE-SAT is NP-complete. However, determining whether Σ has a model that disagrees with a complete interpretation I on at most d variables, where d is a constant, is in P.

To solve DISTANCE-SAT, the authors propose two algorithms, $DP_{distance}$ and $DP_{distance+lasso}$. 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.

Related Work in Constraint Programming In (Hebrard et al. 2007; Hebrard et al. 2005), the authors study various computational problems related to finding similar/diverse solutions. The main decision problems studied in (Hebrard et al. 2005) are the following:

> dDISTANTkSET (resp. dCLOSEkSET)
> Given a polynomial-time decidable and polynomially balanced relation R over strings, a symmetric, reflexive, total and polynomially bounded distance function δ between strings, and some string x, decide whether there is a set S of k strings (i.e., $S \subset \{y \mid (x,y) \in R\}$) such that $min_{y,z \in S}\delta(y,z) \geq d$ (resp. $max_{y,z \in S}\delta(y,z) \leq d$).

which are similar to d-DISTANT SET (resp. d-CLOSE SET): (Hebrard et al. 2005) asks for a set of k solutions d-distant/close to one solution, whereas d-DISTANT/CLOSE SET asks for a set of solutions that is d-close/distant to a set of previously computed solutions. Also, the distance measure considered in dDISTANTkSET (resp. dCLOSEkSET) is computable in polynomial time; in d-DISTANT SET (resp. d-CLOSE SET) deciding whether the distance of a set of solutions is less than a given d is assumed to be in NP.

The main decision problems studied in (Hebrard et al. 2007) are the following:

> dDISTANT (resp. dCLOSE)
> Given a constraint satisfaction problem P with variables ranging over finite domains, a symmetric, reflexive, total and polynomially bounded distance function δ between partial instantiations of variables, and some partial instantiation p of variables of P, decide whether there is a solution s of P such that $\delta(p, s) \geq d$ ($\delta(p, s) \geq d$).

which are similar to d-DISTANT SOLUTION and d-CLOSE SOLUTION. On the other hand, (Hebrard et al. 2007) asks for a solution d-close to a partial solution rather than a set of solutions. Also, the distance measure considered in these problems is computable in polynomial time. However, with S containing a single solution and Δ being computable in polynomial time, d-DISTANT SOLUTION (resp. d-CLOSE SOLUTION) becomes essentially the same as dDISTANT (resp. dCLOSE).

The authors also study some optimization problems related to these problems, similar to the ones that we study above.

As for the computational complexity analysis of these problems, the authors find out that they are all NP-complete; these results comply with the ones presented in Section 4 subject to conditions under which the problems of (Hebrard et al. 2005; Hebrard et al. 2007) above are equivalent to the problems we study in this paper.

Considering partial Hamming distance as in (Bailleux and Marquis 1999), Hebrard et al. 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 (Hebrard et al. 2007; Hebrard et al. 2005).

We note that partial Hamming distance not unrelated to the ones introduced for comparing phylogenies in Section 6.1; 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).

Related Work in Planning In (Srivastava et al. 2007), the authors study the following decision problem:

> dDISTANTkSET (resp. dCLOSEkSET)
> Find a set S of k plans for a planning problem PP, such that $min_{y,z \in S} \delta(y, z) \geq d$ (resp. $max_{y,z \in S} \delta(y, z) \leq d$).

The authors study these problems considering domain-independent distance measures computable in polynomial time (such as Hamming distance or set difference). They present a method (similar to our Online Method 1), where they add global constraints to the underlying constraint satisfaction solver of the GP-CSP planner (Do and Kambhampati 2001). As another method they present a greedy approach (similar to our Online Method 2), where they add global constraints which forces solver to compute k-diverse solutions in each iteration until it computes n solutions. They also present a method (similar to our Online

Method 3) which modifies an existing planner's (Gerevini et al. 2003) heuristic function and computes n k-similar solutions in the search level.

Advantages of using ASP-Based Methods/Tools Our ASP-based methods for computing similar/diverse or close/distant solutions to a given problem have three main advantages compared to other approaches:

- they are not restricted to some domain-independent distance function, like (partial) Hamming distance considered in all the methods/tools above;
- depending on the particular ASP-based method, we can represent domain-independent or domain-specific distance functions in ASP or implement them in C++;
- we can use the definitions of distance functions modularly, without modifying the main problem description or without modifying the search algorithm or the implementation of the solver.

Thus, our ASP-based methods/tools for computing similar/diverse or close/distant solutions are applicable to various problems with different (often domain-specific) distance measures.

In that sense, a user may prefer to use our ASP-based methods/tools for computing similar/diverse or close/distant solutions to a given problem, compared to the SAT-based methods/tools and the CP-based methods/tools, if the user considers a domain-specific distance function but does not want to modify the CP/SAT solvers to be able to compute similar/diverse or distant/close solutions.

Also, our ASP-based methods/tools may be preferred when it is easier to represent the main problem in ASP, due to advantages inherited from the expressive representation language of ASP, such as being able to define the transitive closure. Some sample applications include phylogenetic network reconstruction (Erdem et al. 2006) and wire routing (Coban et al. 2008; Erdem and Wong 2004).

ASP-based methods for computing similar/diverse or close/distant solutions to a given problem are probably most useful for existing well-studied ASP applications, such as phylogeny reconstruction (Brooks et al. 2005; Brooks et al. 2007) or product configuration (Soininen and Niemelä 1998), to be used with domain-specific measures.

9 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 k-similar (resp. k-diverse) solutions; the other asks, given a set of solutions, for a k-close (k-distant) solution s. We have analyzed the computational complexity of these problems, and introduced offline/online methods to solve them. We have applied these offline/online methods to the phylogeny reconstruction problem, and observed their effectiveness in computing similar/diverse phylogenies for Indo-European languages. Similarly we have applied these methods to planning problems, and observed their effectiveness in computing in particular diverse plans in Blocks World. Finally, we have compared our work with related approaches based on other formalisms.

There are many appealing ASP applications for which finding similar/diverse solutions could be useful. In this sense, our methods and implementation (i.e., CLASP-NK) can be useful for ASP. On the other hand, no existing phylogenetic system can compute similar/diverse phylogenies. In this sense, our distance functions, methods, and tools can be useful for phylogenetics. Similarly, no planner can compute similar/diverse plans with respect to a domain-specific measure, our methods and tools can be useful for planning. In general, the ASP-based methods/tools can be useful for finding similar/diverse or close/distant solutions to a problem in two cases: representing the problem in ASP is easier (e.g., if the problem involves recursive definitions like transitive closure), or the distance measure is different from the Hamming distance (implemented in the SAT/CP-based tools).

We are also interested in combinations of the problems studied above (for instance, finding a phylogeny that is the most distant from a given set of phylogenies and that is the closest to another given set of phylogenies), and application of our methods to other problems. The study of these problems is left as a future work.

References

AHUJA, R. K., MAGNANTI, T. L., AND ORLIN, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 93–157.

BAFNA, V. AND PEVZNER, P. 1998. Sorting by transpositions. *SIAM Journal of Discrete Mathematics 11*, 224–240.

BAILLEUX, O. AND MARQUIS, P. 1999. DISTANCE-SAT: complexity and algorithms. In *Proc. of AAAI.* 642–647.

BLUIS, J. AND SHIN, D.-G. 2003. Nodal distance algorithm: Calculating a phylogenetic tree comparison metric. In *Proc. of BIBE.* 87.

BODENREIDER, O., ÇOBAN, Z. H., DOĞANAY, M. C., ERDEM, E., AND KOŞUCU, H. 2008. A preliminary report on answering complex queries related to drug discovery using answer set programming. In *Proc. of ALPSWS.*

BROOKS, D. AND MCLENNAN, D. 1991. *Phylogeny, Ecology, and Behavior: A Research Program in Comparative Biology.* University of Chicago Press, Chicago, IL.

BROOKS, D. R., ERDEM, E., ERDOGAN, S. T., MINETT, J. W., AND RINGE, D. 2007. Inferring phylogenetic trees using answer set programming. *J. Autom. Reasoning 39*, 4, 471–511.

BROOKS, D. R., ERDEM, E., MINETT, J. W., AND RINGE, D. 2005. Character-based cladistics and answer set programming. In *Proc. of PADL.* 37–51.

CALDIRAN, O., HASPALAMUTGIL, K., OK, A., PALAZ, C., ERDEM, E., AND PATOGLU, V. 2009. Bridging the gap between high-level reasoning and low-level control. In *Proc. of LPNMR.*

CAMIN, J. AND SOKAL, R. 1965. A method for deducing branching sequences in phylogeny. *Evolution 19*, 3, 311–326.

CHAFLE, G., DASGUPTA, K., KUMAR, A., MITTAL, S., AND SRIVASTAVA, B. 2006. Adaptation in web service composition and execution. In *Proc. of ICWS.* 549–557.

CHEN, Z.-Z. AND TODA, S. 1995. The complexity of selecting maximal solutions. *Information and Computation 119*, 231–239.

COBAN, E., ERDEM, E., AND TURE, F. 2008. Comparing ASP, CP, ILP on two challenging applications: Wire routing and haplotype inference. In *Proc. of the 2nd International Workshop on Logic and Search (LaSh 2008).*

DASGUPTA, B., HE, X., JIANG, T., LI, M., TROMP, J., AND ZHANG, L. 1997. On distances between phylogenetic trees. In *Proc. of SODA.* 427–436.

DAVIS, M., LOGEMANN, G., AND LOVELAND, D. 1962. A machine program for theorem-proving. *Communications of the ACM 5*, 394–397.

DO, M. B. AND KAMBHAMPATI, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence 132,* 2, 151–182.

EDWARDS, A. AND CAVALLI-SFORZA, L. 1964. *Phenetic and Phylogenetic Classification*, 67–76.

EITER, T. AND SUBRAHMANIAN, V. 1999. Heterogeneous active agents, ii: Algorithms and complexity. *Artif. Intell. 108(1-2)*, 257–307.

ERDEM, E. 2002. Theory and applications of answer set programming. Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin.

ERDEM, E. 2009. PHYLO-ASP: Phylogenetic systematics with answer set programming. In *Proc. of LPNMR*. 567–572.

ERDEM, E., LIFSCHITZ, V., AND RINGE, D. 2006. Temporal phylogenetic networks and logic programming. *Theory and Practice of Logic Programming 6,* 5, 539–558.

ERDEM, E. AND WONG, M. D. F. 2004. Rectilinear Steiner Tree construction using answer set programming. In *Proc. of ICLP*. 386–399.

FELSENSTEIN, J. 2009. Phylip. `http://evolution.genetics.washington.edu/phylip.html`.

FERRARIS, P. AND LIFSCHITZ, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5*, 45–74.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007a. clasp: A Conflict-Driven Answer Set Solver. In *Proc. of LPNMR*. 260–265.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007b. T.: Conflict-driven answer set solving. In *Proc. of IJCAI*. 386–392.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

GEREVINI, A., SAETTI, A., AND SERINA, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *J. Artif. Int. Res. 20,* 1, 239–290.

GUTIN, G. 2003. *Handbook of Graph Theory*. CRC Press, Chapter 5.3 Independent sets and cliques, 389–402.

HAMMING, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal 29,* 2, 147–160.

HEBRARD, E., HNICH, B., O'SULLIVAN, B., AND WALSH, T. 2005. Finding diverse and similar solutions in constraint programming. In *Proc. of AAAI*. 372–377.

HEBRARD, E., O'SULLIVAN, B., AND WALSH, T. 2007. Distance constraints in constraint satisfaction. In *Proc. of IJCAI*. 106–111.

HON, W.-K., KAO, M.-Y., AND LAM, T.-W. 2000. *Algorithms and Computation*. Springer Berlin / Heidelberg, Chapter Improved Phylogeny Comparisons: Non-shared Edges, Nearest Neighbor Interchanges, and Subtree Transfers, 369–382.

KAUTZ, H. A. AND SELMAN, B. 1992. Planning as satisfiability. In *Proc. of ECAI*. 359–363.

KUHNER, M. AND FELSENSTEIN, J. 1994. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates [published erratum appears in mol biol evol 1995 may;12(3):525]. *Mol Biol Evol 11,* 3 (May), 459–468.

LIFSCHITZ, V. 1999. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, 357–373.

LIFSCHITZ, V. 2008. What is answer set programming? In *Proc. of. AAAI*. 1594–1597.

LIFSCHITZ, V. 2010. Thirteen definitions of a stable model. In *Fields of Logic and Computation*. 488–503.

LIFSCHITZ, V., TANG, L. R., AND TURNER, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

MARQUES-SILVA, J. AND SAKALLAH, K. 1999. A search algorithm for propositional satisfiability. *IEEE Trans. Computers 5*, 506–521.

MCILRAITH, S. A. AND SON, T. C. 2002. Adapting Golog for composition of semantic web services. In *Proc. of KR*. 482–496.

NOGUEIRA, M., BALDUCCINI, M., GELFOND, M., WATSON, R., AND BARRY, M. 2001. An a-prolog decision support system for the space shuttle. In *Proc. of PADL*. Springer-Verlag, London, UK, 169–183.

NYE, T. M., LIO, P., AND GILKS, W. R. 2006. A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. *Bioinformatics 22,* 1 (January), 117–119.

PAPADIMITRIOU, C. 1994. *Computational Complexity*. Addison-Wesley.

RINGE, D., WARNOW, T., AND TAYLOR, A. 2002. Indo-European and computational cladistics. *Transactions of the Philological Society 100*, 1, 59–129.

ROBINSON, D. F. AND FOULDS, L. R. 1981. Comparison of phylogenetic trees. *Mathematical Biosciences 53*, 1-2 (February), 131–147.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138*, 181–234.

SOININEN, T. AND NIEMELÄ, I. 1998. Developing a declarative rule language for applications in product configuration. In *Proc. of PADL*. 305–319.

SON, T., PONTELLI, E., AND SAKAMA, C. 2009. Logic programming for multiagent planning with negotiation. In *Proc. of ICLP*. 99–114.

SRIVASTAVA, B., NGUYEN, T. A., GEREVINI, A., KAMBHAMPATI, S., DO, M. B., AND SERINA, I. 2007. Domain independent approaches for finding diverse plans. In *Proc. of IJCAI*. 2016–2022.

SWOFFORD, D. L. 2003. PAUP*: Phylogenetic analysis under parsimony (and other methods). version 4.0. Sinauer Associates, Sunderland, Mass.

WHITE, J. AND O'CONNELL, J. 1982. *A Prehistory of Australia, New Guinea, and Sahul*. Academic, San Diego, CA.

Appendix A Proofs of Theorems

Proof of Theorem 1

Membership: Consider a non-deterministic Turing machine M, operating on input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), n, and k, which guesses S as a set $\{s_1, \ldots, s_n\}$ of n interpretations over the alphabet of $\mathcal{P}$, together with a potential witness w for a computation of $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$) of length polynomial in n. After that, for $1 \leq i \leq n$, M checks whether s_i is an answer set of $\mathcal{P}$ and whether all s_i represent distinct solutions of the problem. It rejects if any of these tests does not succeed. Otherwise, M proceeds by verifying that w is a witness of $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$) given S and k as its input. If so, then M accepts, otherwise it rejects. Since n is polynomial in the size of the input to M, this also holds for the guess of M. Moreover, the subsequent computation of M, i.e., the tests carried out, can be accomplished in time polynomial in n. Therefore, M is a non-deterministic Turing machine which decides n k-SIMILAR SOLUTIONS (respectively n k-DIVERSE SOLUTIONS) in polynomial time, which proves NP-membership for these problems.

Hardness: Let $\phi = \bigwedge_{1 \leq i \leq l} c_i$ be a Boolean formula in conjunctive normal form (CNF) over variables $B = \{b_1, \ldots, b_m\}$, i.e., each c_i is a clause over variables from B. By $\bar{x}$ we denote the complement of a literal x, i.e., $\bar{x} = \neg b$ if $x = b$, and $\bar{x} = b$ if $x = \neg b$. This notation is extended to clauses in the obvious way: $\bar{c} = \bar{x}_1 \wedge \ldots \wedge \bar{x}_{l_c}$ for a clause $c = x_1 \vee \ldots \vee x_{l_c}$.

Consider the normal logic program $\mathcal{P} = \{b_i \leftarrow \textit{not}\ nb_i;\ nb_i \leftarrow \textit{not}\ b_i \mid 1 \leq i \leq m\} \cup \{\leftarrow \bar{c}'_i \mid 1 \leq i \leq l\}$, where $\bar{c}'$ denotes the conjunction obtained from $\bar{c}$ by replacing negative literals $\neg x$ in $\bar{c}$ by nx (and using ',' instead of '$\wedge$'). It is easily verified (and well-known) that $\mathcal{P}$ has an answer set iff ϕ is satisfiable (and that every answer set of $\mathcal{P}$ is in 1-to-1 correspondence with a satisfying assignment of ϕ in the obvious way).

Given $\mathcal{P}$, consider the n k-SIMILAR SOLUTIONS (respectively n k-DIVERSE SOLUTIONS) problem, where $n = 1$, $k = 0$, and for any set S of answer sets of $\mathcal{P}$, the distance measure Δ is defined by $\Delta(S) = 0$. Note that Δ is normal and computable in constant time. Then, there exists a solution to the problem iff there exists a set S of answer sets of $\mathcal{P}$ such that $|S| = 1$, i.e., $\mathcal{P}$ has an answer set. Since $\mathcal{P}$ has an answer set iff ϕ is satisfiable, this proves NP-hardness of the n k-SIMILAR SOLUTIONS (respectively n k-DIVERSE SOLUTIONS) problem. Note that this argument holds for any normal Δ. □

Proof of Theorem 2

Membership: Consider a non-deterministic Turing machine M, operating on input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), a set S of solutions given by answer sets of $\mathcal{P}$, and k. It guesses an interpretation s over the alphabet of $\mathcal{P}$ (which is polynomial in the size of $\mathcal{P}$), together with a potential witness w for a computation of $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$) of length polynomial in $|S|+|s|+\log k$. After that, M checks whether s is an answer set of $\mathcal{P}$ and whether it represents a solution different from all solutions in S. It rejects if any of these tests does not succeed. Otherwise, M proceeds by verifying that w is a witness of $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$) on input $S \cup \{s\}$ and k. If so, then M accepts, otherwise it rejects. The guess of M is polynomial in its input and the subsequent computation of M, i.e., the tests carried out, can be accomplished in polynomial time. Therefore, M is a non-deterministic Turing machine which decides k-CLOSE SOLUTION (respectively k-DISTANT SOLUTION) in polynomial time, which proves NP-membership for these problems.

Hardness: Consider the normal logic program given in the proof of Theorem 1, and the k-CLOSE SOLUTION problem, where $S = \emptyset$, $k = 0$, and for any set S' of answer sets of $\mathcal{P}$, the distance measure Δ is defined by $\Delta(S') = 0$. Note that Δ is normal and computable in constant time. Then, there exists a solution to the problem iff there exists a set S' of answer sets of $\mathcal{P}$ such that $S' \neq \emptyset$, i.e., $\mathcal{P}$ has an answer set, which proves NP-hardness of the k-CLOSE SOLUTION problem. Similarly, the k-DISTANT SOLUTION problem, where $S = \emptyset$, $k = 0$, and $\Delta(S') = 0$, has a solution iff $\mathcal{P}$ has an answer set. Moreover, the above arguments hold for any normal Δ. This proves the claim. □

Proof of Theorem 3

Membership: The problem of computing the cardinality of a maximal solution S of size at most n is an optimization problem for a problem in NP such that the optimal value can be represented in using $\log n$ bits. Let M_{opt} be an oracle for this problem, and consider a non-deterministic Turing machine M', with output tape operating on input $\mathcal{P}$, $M_{\Delta}^{\leq}$ (resp. $M_{\Delta}^{\geq}$), and k. Initially, M' calls M_{opt} with $\mathcal{P}$, $M_{\Delta}^{\leq}$ (resp. $M_{\Delta}^{\geq}$), and k as input to compute the maximum cardinality c of a set of solutions S such that $|S| \leq n$. Then, M' proceeds like the nondeterministic Turing machine M in the proof of Theorem 1 using $n = c$, additionally writing the guessed solution S to its output tape. Since the latter is accomplished in time polynomial in c, M' is a non-deterministic Turing machine with output tape that consults an oracle once for computing the optimal value of an optimization problem solvable in NP. Thus, M' is in FNP//log and decides MAXIMAL n k-SIMILAR SOLUTIONS (respectively MAXIMAL n k-DIVERSE SOLUTIONS).

Hardness: We reduce X-MinModel to the problem of computing MAXIMAL n k-SIMILAR SOLUTIONS. X-MinModel is the following FNP//log-complete problem: Given a Boolean formula in CNF as in the proof of Theorem 1, and a subset X of B, compute an X-minimal model of ϕ, i.e., a satisfying assignment for ϕ, which is subset minimal among all satisfying assignments for ϕ on the variables from X which are assigned true. We identify a truth assignment with the set of Boolean variables that are assigned true, and for a truth assignment s, we use $s|_X$ to denote its restriction to variables from X.

Consider the normal logic program given in the proof of Theorem 1, and the MAXIMAL n k-SIMILAR SOLUTIONS problem, where $n = |X|$, $k = 0$, and Δ is defined as follows. For a given set S of answer sets of $\mathcal{P}$, such that $|S| > 0$, we set $\Delta(S) = 0$ if for every pair of answer sets s_1, s_2 in S, either $s_1|_X \subset s_2|_X$, or $s_2|_X \subset s_1|_X$. Otherwise (and if $S = \emptyset$), $\Delta(S) = 1$. Note that Δ is computable in polynomial time, performing less than $2n^2$ checks for proper containment, where $|S| = n$. Observe also that the answer sets in a set S such that $\Delta(S) = 0$, can be strictly ordered wrt. subset inclusion on their restrictions to X, and that $|X|$ is the maximum cardinality for such a set of answer sets. Given S such that $\Delta(S) = 0$, let s_1 denote the minimal answer set in S wrt. subset inclusion on the restriction to X. The following is trivial: the MAXIMAL n k-SIMILAR SOLUTIONS problem above has a solution iff ϕ is satisfiable. By the problem definition, it also holds for every solution S of the problem that $\Delta(S) = 0$.

We show that if S is a solution of the MAXIMAL n k-SIMILAR SOLUTIONS problem given above, then s_1 is an X-minimal model of ϕ. Towards a contradiction assume that there exists a satisfying assignment s' for ϕ, such that $s'|_X \subset s_1|_X$. Consider $s_0 = s' \cup \{nb \mid b \in B, b \notin s'\}$. Since s' satisfies ϕ, it holds that s_0 is an answer set of $\mathcal{P}$. Moreover

$s_0 \notin S$, since $s_0|_X \subset s_1|_X$ and s_1 is the minimal answer set in S wrt. subset inclusion on the restriction to X. As a consequence, $S \cup \{s_0\} \supset S$ and $\Delta(S \cup \{s_0\}) = 0$. However, since the latter also implies $|S \cup \{s_0\}| \leq n$, this contradicts our assumption that S is a solution of the MAXIMAL n k-SIMILAR SOLUTIONS problem above. We have thus shown that no satisfying assignment s' for ϕ exists, such that $s'|_X \subset s_1|_X$, i.e., that s_1 is an X-minimal model of ϕ. This completes the reduction of X-MinModel to the problem of computing MAXIMAL n k-SIMILAR SOLUTIONS, proving FNP//log-hardness.

For a reduction of X-MinModel to the problem of computing MAXIMAL n k-DIVERSE SOLUTIONS, we simply swap the values of Δ and define: $\Delta(S) = 1$ if $|S| > 0$ and for every pair of answer sets s_1, s_2 in S, either $s_1|_X \subset s_2|_X$, or $s_2|_X \subset s_1|_X$. Otherwise (and if $S = \emptyset$), $\Delta(S) = 0$. FNP//log-hardness follows by analogous arguments. □

Proof of Theorem 4

Membership: Consider a deterministic Turing machine M', with output tape and an oracle for NP-problems, which operates on input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), and n. Initially, M' prepares an integer k_1 of n bits with the less significant half of bits set to 1 and the remaining bits set to 0. Then, M' successively uses its oracle operating as the nondeterministic Turing machine M in the proof of Theorem 1, starting with input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), n, and k_1, performing a binary search for an optimal k. After that, M' once more uses its oracle like the nondeterministic Turing machine M in the proof of Theorem 1, but additionally copying the solution S guessed by the oracle to its output tape. Since the latter is accomplished in time polynomial in n, and since a polynomial number of calls to the oracle is sufficient to complete the binary search, M' is in FP$^{\text{NP}}$and decides n MOST SIMILAR SOLUTIONS (respectively n MOST DIVERSE SOLUTIONS).

If the value of $\Delta(S)$ is polynomial in the size of S, then the problem of computing the maximal value of $\Delta(S)$ over all solutions S is an optimization problem for a problem in NP such that the optimal value can be represented using $\log n$ bits. Let M_{opt} be an oracle for this problem, and consider a non-deterministic Turing machine M'' with output tape operating on input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), and n. Initially, M'' calls M_{opt} with $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), and n as input to compute the value k for $\Delta(S)$ of an optimal solution S. Then, M'' proceeds like the nondeterministic Turing machine M in the proof of Theorem 1, additionally writing the guessed solution S to its output tape. Since the latter is accomplished in time polynomial in n, M'' is in FNP//log and decides n MOST SIMILAR SOLUTIONS (respectively n MOST DIVERSE SOLUTIONS).

Hardness: We reduce the Traveling Salesman Problem (as, e.g., in (Papadimitriou 1994)) to the problem of computing n MOST SIMILAR SOLUTIONS. Consider m cities $1, \ldots, m$, and a non-negative integer distance $d_{i,j}$ between any two cities i and j. The task is to compute a tour visiting all cities once (i.e., a Hamilton Cycle) of shortest length.

For a reduction, consider $\mathcal{P} = \{\ p_{i,j} \leftarrow \textit{not}\ np_{i,j};\ \ np_{i,j} \leftarrow \textit{not}\ p_{i,j};\ \ r_j \leftarrow p_{i,j};$ $\leftarrow \textit{not}\ r_j \mid i \neq j\} \cup \{\leftarrow p_{i,j}, p_{k,j};\ \leftarrow p_{i,j}, p_{i,k} \mid i \neq j, i \neq k, j \neq k\}$, where indices i, j, and k range over $1, \ldots, m$. Every answer set s of $\mathcal{P}$ uniquely corresponds to a Hamilton Cycle encoded by the atoms $p_{i,j}$ in s, and every permutation of the cities gives rise to exactly one answer set of $\mathcal{P}$. This can easily be verified observing that the first two rules encode a nondeterministic guess of a set of atoms $p_{i,j}$. The third and fourth rule are satisfied iff 'every city is reached', i.e., if for every index j there exists an index i, such that $p_{i,j}$ is true. The last two rules are satisfied iff every city 'is reached from at most one different city' and 'reaches at most one different city', i.e., iff for different indices i, j, and k, $p_{i,j}$ and $p_{k,j}$ cannot both be true, as well as $p_{i,j}$ and $p_{i,k}$ cannot both be true.

Given this program, consider the n MOST SIMILAR SOLUTIONS problem, where $n = 1$, and for any set S of answer sets of $\mathcal{P}$, the distance measure Δ is defined by $\Delta(S) = \Sigma_{s \in S} \Sigma_{p_{i,j} \in s}\, d_{i,j}$. Note that Δ is monotonic and thus computable in polynomial time in the size of S. Moreover, S is a solution to this problem iff, by its definition, S contains exactly one answer set s of $\mathcal{P}$, and iff $\Delta(S)$ is minimal among all sets of answer sets of $\mathcal{P}$, thus in particular among elementary such sets. By the definition of Δ, this implies that $S = \{s\}$ is a solution iff s encodes a Hamilton Cycle of minimal cost. This proves $\mathrm{FP}^{\mathrm{NP}}$-hardness for the n MOST SIMILAR SOLUTIONS problem in general.

For a reduction of TSP to the problem of computing n MOST DIVERSE SOLUTIONS, consider $\Delta'(S) = m \times max_d - \Delta(S)$, where max_d is the maximum distance $d_{i,j}$ given. Also Δ' is monotonic and computable in polynomial time, and by analogous arguments $\mathrm{FP}^{\mathrm{NP}}$-hardness follows for the n MOST DIVERSE SOLUTIONS problem in the general case.

If the value of $\Delta(S)$ is polynomial in the size of S, then FNP//log-hardness is obtained by a reduction of X-MinMod: Let $\mathcal{P}$ be the normal logic program in the proof of Theorem 1, and consider the n MOST SIMILAR SOLUTIONS problem, where $n = 1$, and $\Delta(S)$ is given by the minimal (respectively maximal) partial Hamming distance on X between an answer set $s \in S$ and $\emptyset$ (respectively X). It is a straightforward consequence of the definition of Δ, that if $S = \{s\}$ is a solution to this n MOST SIMILAR SOLUTIONS problem (respectively to this n MOST DIVERSE SOLUTIONS problem), then s is an X-minimal model of ϕ (cf. also the proof of Theorem 3). □

Proof of Theorem 5

Membership: Consider a deterministic Turing machine M', with output tape and an oracle for NP-problems, which operates on input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), and S. Initially, M' prepares an integer k_1 of n bits with the less significant half of bits set to 1 and the remaining bits set to 0. Then, M' successively uses its oracle operating as the nondeterministic Turing machine M in the proof of Theorem 2, starting with input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), S, and k_1, performing a binary search for an optimal k. After that, M' once more uses its oracle like the nondeterministic Turing machine M in the proof of Theorem 2, but additionally copying the answer set s guessed by the oracle to its output tape. Since the latter is accomplished in time polynomial in n, and since a polynomial number of calls to the oracle is sufficient to complete the binary search, M' is in $\mathrm{FP}^{\mathrm{NP}}$and decides CLOSEST SOLUTION (respectively MOST DISTANT SOLUTIONS).

If the value of $\Delta(S)$ is polynomial in the size of a set S of n solutions, then the problem of computing the maximal value of $\Delta(S \cup \{s\})$ for any solution $S \cup \{s\}$ is an optimization problem for a problem in NP such that the optimal value can be represented using logarithmically many bits in the size of $S \cup \{s\}$. Let M_{opt} be an oracle for this problem, and consider a non-deterministic Turing machine M'' with output tape operating on input $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), and S. Initially, M'' calls M_{opt} with $\mathcal{P}$, $M^{\leq}_{\Delta}$ (resp. $M^{\geq}_{\Delta}$), and S as input to compute the value k for $\Delta(S \cup \{s\})$ of an optimal solution $S \cup \{s\}$. Then, M'' proceeds like the nondeterministic Turing machine M in the proof of Theorem 2, additionally writing the guessed answer set s to its output tape. Since the latter is accomplished in time polynomial in the input, M'' is in FNP//log and decides CLOSEST SOLUTION (respectively MOST DISTANT SOLUTIONS).

Hardness: The respective lower bounds are an immediate consequence of the problem reductions in the proof of the previous Theorem 4. Just observe that for given $\mathcal{P}$ and Δ, the solutions of an n MOST SIMILAR SOLUTIONS problem with input $n = 1$ coincide with the solutions of the CLOSEST SOLUTION problem with input $S = \emptyset$, (and the same holds for the problem n MOST DIVERSE SOLUTIONS with input $n = 1$ and MOST DISTANT SOLUTION with input $S = \emptyset$). It thus suffices to recall that the reductions mentioned above are reductions to problems with input $n = 1$. □

Proof of Theorem 6

Membership: Consider a non-deterministic Turing machine M, operating on input $\mathcal{P}$, $M_{\Delta}^{\leq}$, $M_{\Delta}^{\geq}$, a set S of answer sets of $\mathcal{P}$, and k. Let n be the size of its input. First, M guesses S', such that $|S'|$ is polynomial in n, as a set $\{s'_1, \ldots, s'_m\}$ of interpretations over the alphabet of $\mathcal{P}$, two integers k_1 and k_2 in binary representation of size at most polynomial in n, together with two potential witnesses w_1 and w_3 of $M_{\Delta}^{\leq}$ and of length polynomial in $|S|$ and $|S'|$, respectively, as well as two potential witnesses w_2 and w_4 of $M_{\Delta}^{\geq}$ of length polynomial in $|S|$ and $|S'|$, respectively. After that, M checks whether S' is different from S, as well as whether s'_i is an answer set of $\mathcal{P}$, for $1 \leq i \leq m$. It rejects if any of these tests does not succeed. Otherwise, M proceeds by verifying that w_1 is a witness of $M_{\Delta}^{\leq}$ on input S and k_1, that w_2 is a witness of $M_{\Delta}^{\geq}$ on input S and k_1, as well as that w_3 is a witness of $M_{\Delta}^{\leq}$ on input S' and k_2, and that w_4 is a witness of $M_{\Delta}^{\geq}$ on input S' and k_2. If either test fails M rejects, otherwise it checks whether $|k_1 - k_2| \leq k$ (respectively $|k_1 - k_2| \geq k$), and if so accepts, otherwise it rejects. Note that due to our assumptions that the size of S' to consider is polynomial in n, and that the value of $\Delta(S)$, respectively $\Delta(S')$ is bounded by an exponential in the size of S, respectively in the size of S', the guess of M, which is polynomial in n, is sufficient for deciding the problem. Moreover, the subsequent computation of M, i.e., the tests carried out, can be accomplished in polynomial time. Therefore, M is a non-deterministic Turing machine which decides k-CLOSE SET (respectively k-DISTANT SET) in polynomial time, which proves NP-membership for these problems.

Hardness: Consider the normal logic program given in the proof of Theorem 1, and the k-CLOSE SET problem, where $S = \emptyset$, $k = 0$, and for any set S' of answer sets of $\mathcal{P}$, the distance measure Δ is defined by $\Delta(S') = 0$. Note that Δ is normal and computable in constant time. Then, there exists a solution to the problem iff there exists a set S' of answer sets of $\mathcal{P}$ such that $S' \neq \emptyset$, i.e., $\mathcal{P}$ has an answer set, which proves NP-hardness of the k-CLOSE SET problem. Similarly, the k-DISTANT SET problem, where $S = \emptyset$, $k = 0$, and $\Delta(S') = 0$, has a solution iff $\mathcal{P}$ has an answer set. Again the arguments hold for any normal Δ, which proves the claim. □

Proof of Proposition 1

In order to compute $D_n(P_1, P_2)$, we need to perform $\binom{l}{2}$ nodal distance computations where $l = |L|$ is the number of leaves. Each nodal distance computation can be considered as a shortest path computation in the tree. The reaching algorithm is developed to find shortest paths in acyclic graphs (e.g., trees) in $O(m)$ time where m is the number of edges in the graph (Ahuja et al. 1993). The number of edges in a phylogeny is equal to $2 \times l - 2$; therefore, we can compute $D_n(P_1, P_2)$ in $\binom{l}{2} \times (2 \times l - 2)$ steps which is $O(l^3)$. □

Proof of Proposition 2
Let v be the number of vertices in one tree, then v^2 is an upper bound for the number of the pairs that we can compare their descendants. Therefore, we have at most $O(v^2)$ comparisons where each comparison takes time $O(l)$ since the number of descendants is bounded by l, where $l = |L|$ (assuming that we obtained the descendants of each vertex in a preprocessing step which takes $O(v{\cdot}l)$ time). Since $v = 2 \times l - 1$, $D_l(P_1, P_2)$ can be computed in $(2 \times l - 1)^2 \times l$ steps which is $O(l^3)$. □

Proof of Proposition 3
Let S'_p be a set of all completions of the partial phylogeny P_p. For every $P \in S'_p$, we need to prove that

$$\mathcal{LB}_n(P_p, P_c) \leq D_n(P, P_c)$$

holds.

Let $P_l \in \arg\min_{P \in S'_p}(D_n(P, P_c))$ be a completion with smallest distance. Then it will be enough to prove that

$$\mathcal{LB}_n(P_p, P_c) \leq D_n(P_l, P_c)$$

holds. If we replace $\mathcal{LB}_n$ and D_n with their equivalents, the inequality will look like the following:

$$\sum_{x,y \in L_p} |ND_{P_c}(x,y) - ND_{P_p}(x,y)| \leq \sum_{x,y \in L} |ND_{P_l}(x,y) - ND_{P_c}(x,y)|$$

We can break the right hand side summation into two for L_p and $L \backslash L_p$ as follows:

$$\begin{aligned}&\textstyle\sum_{x,y \in L_p} |ND_{P_c}(x,y) - ND_{P_p}(x,y)| \leq \\ &\textstyle\sum_{x,y \in L_p} |ND_{P_l}(x,y) - ND_{P_c}(x,y)| + \sum_{(x,y) \in L^2 \backslash L_p^2} |ND_{P_l}(x,y) - ND_{P_c}(x,y)|\end{aligned}$$

The distance between x and y is the same for P_p and P_l where $x, y \in L_p$. Therefore, terms cancel each other and we have the following:

$$0 \leq \sum_{(x,y) \in L^2 \backslash L_p^2} |ND_{P_l}(x,y) - ND_{P_c}(x,y)|$$

Since the right hand side is a summation of absolute values, the inequality holds which completes the proof. □

Proof of Proposition 4
Let S'_p be a set of all completions of the partial phylogeny P_p. For every $P \in S'_p$, we need to prove that

$$\mathcal{UB}_n(P_p, P_c) \geq D_n(P, P_c)\cdot$$

Let $P_u \in \arg\max_{p \in S'_p}(D_n(p, P_c))$ be a completion at largest distance. Then it will be enough to prove that

$$\mathcal{UB}_n(P_p, P_c) \geq D_n(P_u, P_c)\cdot$$

If we replace $\mathcal{UB}_n$ and D_n with their definition, the inequality is

$$\sum_{x,y\in L_p} |\mathit{ND}_{P_c}(x,y)-\mathit{ND}_{P_p}(x,y)|+(\binom{l}{2}-\binom{|L_p|}{2})\times l \geq \sum_{x,y\in L} |\mathit{ND}_{P_l}(x,y)-\mathit{ND}_{P_c}(x,y)|\cdot$$

We can break the right hand side summation into two for L_p and $L\backslash L_p$ as follows:

$$\begin{array}{l} \sum_{x,y\in L_p} |\mathit{ND}_{P_c}(x,y) - \mathit{ND}_{P_p}(x,y)| + (\binom{l}{2} - \binom{|L_p|}{2}) \times l \geq \\ \sum_{x,y\in L_p} |\mathit{ND}_{P_u}(x,y) - \mathit{ND}_{P_c}(x,y)| + \sum_{x,y\in L\backslash L_p} |\mathit{ND}_{P_u}(x,y) - \mathit{ND}_{P_c}(x,y)| \end{array}$$

The distance between x and y is same for P_p and P_u where $x, y \in L_p$. Terms cancel each other:

$$(\binom{l}{2} - \binom{|L_p|}{2}) \times l \geq \sum_{x,y\in L\backslash L_p} |\mathit{ND}_{P_u}(x,y) - \mathit{ND}_{P_c}(x,y)|\cdot$$

The maximum nodal distance in a tree is equal to the number of leaves; therefore, each term in the right hand side of the inequality is at most l. Since, there are $(\binom{l}{2} - \binom{|L_p|}{2})$ terms in the right hand side summation, $(\binom{l}{2} - \binom{|L_p|}{2}) \times l$ is greater than or equal to the summation. $\square$

Proof of Proposition 5
Take any plan-completion X of the partial plan P_p. Consider two cases.

Case 1: $|X| \leq |P_c|$. Our goal is to prove that

$$\mathcal{LB}_h(P_p, P_c) \leq D_h(X, P_c)\cdot$$

By the definition of D_h, the distance between X and P_c is:

$$D_h(X, P_c) = |\{i \mid \mathit{act}_X(i) \neq \mathit{act}_{P_c}(i), 1 \leq i \leq |X|\}| + |P_c| - |X|\cdot$$

Since X is a plan-completion of P_p and $|X| \leq |P_c|$, dom $\mathit{act}_{P_p} \subseteq$ dom act_{P_c}; then, by the definition of $\mathcal{LB}_h$:

$$\mathcal{LB}_h(P_p, P_c) = |\{i \mid \mathit{act}_{P_p}(i) \neq \mathit{act}_{P_c}(i), i \in \text{dom } \mathit{act}_{P_p}\}|\cdot$$

Since X is a plan-completion of P_p,

$$\{i \mid \mathit{act}_{P_p}(i) \neq \mathit{act}_{P_c}(i), i \in \text{dom } \mathit{act}_{P_p}\} \subseteq \{i \mid \mathit{act}_X(i) \neq \mathit{act}_{P_c}(i), 1 \leq i \leq |X|\}\cdot$$

Hence,

$$\mathcal{LB}_h(P_p, P_c) \leq |\{i \mid \mathit{act}_X(i) \neq \mathit{act}_{P_c}(i), 1 \leq i \leq |X|\}| + |P_c| - |X| = D_h(X, P_c)\cdot$$

Case 2: $|X| > |P_c|$. Our goal is to prove that

$$\mathcal{LB}_h(P_p, P_c) \leq D_h(P_c, X)\cdot$$

By the definition of D_h, the distance between X and P_c is:

$$D_h(P_c, X) = |\{i \mid \mathit{act}_X(i) \neq \mathit{act}_{P_c}(i), 1 \leq i \leq |P_c|\}| + |X| - |P_c|\cdot$$

By the definition of $\mathcal{LB}_h$:

$$\begin{array}{rl} \mathcal{LB}_h(P_p, P_c) = & |\{i \mid \mathit{act}_{P_p}(i) \neq \mathit{act}_{P_c}(i), i \in \text{dom } \mathit{act}_{P_p}, 1 \leq i \leq |P_c|\}| + \\ & |\{i \mid l \geq i > |P_c|, i \in \text{dom } \mathit{act}_{P_p}\}|\cdot \end{array}$$

Since X is a plan-completion of P_p, act_X extends act_{P_p}, and then

$$\begin{aligned}&\{i \mid act_{P_p}(i) \neq act_{P_c}(i), i \in \text{dom}\, act_{P_p}, 1 \leq i \leq |P_c|\}\\ &\quad\subseteq \{i \mid act_X(i) \neq act_{P_c}(i), 1 \leq i \leq |X|\}\cdot\end{aligned}$$

Since $|X| > |P_c|$,

$$|X|-|P_c| > |\{i \mid |X| \geq i > |P_c|, i \in \text{dom}\, act_{P_p}\}| = |\{i \mid l \geq i > |P_c|, i \in \text{dom}\, act_{P_p}\}|\cdot$$

Hence,

$$\mathcal{LB}_h(P_p, P_c) \leq |\{i \mid act_X(i) \neq act_{P_c}(i), 1 \leq i \leq |X|\}| + |X| - |P_c| = D_h(P_c, X)\cdot$$

□

Proof of Proposition 6

Take any plan-completion X of partial plan P_p. Consider two cases.

Case 1: $|X| \leq |P_c|$. Our goal is to prove that

$$\mathcal{UB}_h(P_p, P_c) \geq D_h(X, P_c)$$

where

$$\begin{aligned}\mathcal{UB}_h(P_p, P_c) &= l - |\{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}\}|,\\ D_h(X, P_c) &= |\{i \mid act_X(i) \neq act_{P_c}(i), 1 \leq i \leq |X|\}| + |P_c| - |X|\cdot\end{aligned}$$

Since $|X| \leq |P_c|$ and X is a plan-completion of P_p, the set

$$\{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}\}$$

does not intersect with the set

$$Y = \{i \mid act_X(i) \neq act_{P_c}(i), 1 \leq i \leq |X|\} \cup \{i \mid |X| < i \leq |P_c|\}\cdot$$

Then

$$\{1, \cdots, l\} \setminus \{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}\}$$

is a superset of Y. Therefore,

$$\begin{aligned}\mathcal{UB}_h(P_p, P_c) &= l - |\{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}\}|\\ &\geq |\{i \mid act_X(i) \neq act_{P_c}(i), 1 \leq i \leq |X|\}| + |P_c| - |X|\\ &= D_h(X, P_c)\cdot\end{aligned}$$

Case 2: $|X| > |P_c|$. Our goal is to prove that

$$\mathcal{UB}_h(P_p, P_c) \geq D_h(P_c, X)$$

where

$$\begin{aligned}\mathcal{UB}_h(P_p, P_c) &= l - |\{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}, 1 \leq i \leq |P_c|\}|,\\ D_h(P_c, X) &= |\{i \mid act_X(i) \neq act_{P_c}(i), 1 \leq i \leq |P_c|\}| + |X| - |P_c|\cdot\end{aligned}$$

Since $|X| > |P_c|$ and X is a plan-completion of P_p, the set

$$\{i \mid act_{P_p}(i) = act_{P_c}(i), i \in \text{dom}\, act_{P_p}, 1 \leq i \leq |P_c|\}$$

does not intersect with the set

$$Y = \{i \mid \mathit{act}_X(i) \neq \mathit{act}_{P_c}(i), 1 \leq i \leq |P_c|\} \cup \{i| \, |P_c| < i \leq |X|\}\cdot$$

Then

$$\{1, \cdot\cdot\cdot, l\} \setminus \{i \mid \mathit{act}_{P_p}(i) = \mathit{act}_{P_c}(i), i \in \text{dom}\, \mathit{act}_{P_p}, 1 \leq i \leq |P_c|\}$$

is a superset of Y. Therefore,

$$\begin{aligned}\mathcal{UB}_h(P_p, P_c) &= l - |\{i \mid \mathit{act}_{P_p}(i) = \mathit{act}_{P_c}(i), i \in \text{dom}\, \mathit{act}_{P_p}, 1 \leq i \leq |P_c|\}| \\ &\geq |\{i \mid \mathit{act}_X(i) \neq \mathit{act}_{P_c}(i), 1 \leq i \leq |P_c|\}| + |X| - |P_c| \\ &= D_h(P_c, X)\cdot\end{aligned}$$

□

Appendix B ASP Formulations

```
c{clique(X) : vertex(X)}c.
:- clique(X), clique(Y), vertex(X), vertex(Y), X!=Y, not edge(X,Y),
   not edge(Y,X).
```

Fig. B 1. ASP formulation of the c-clique problem (a clique of size c).

```
solution(1..n).
c{clique(S,X) : vertex(X)}c :- solution(S).
:- clique(S,X), clique(S,Y), not edge(X,Y), not edge(Y,X), X!=Y.
different(S1,S2) :- clique(S1,X), clique(S2,Y), S1 != S2, X != Y.
:- not different(S1,S2), solution(S1;S2), S1!=S2.
```

Fig. B 2. ASP formulation that computes n distinct c-cliques.

```
same(S1,S2,V) :-  clique(S1,V), clique(S2,V), solution(S1;S2),
  vertex(V), S1 < S2.
hammingDistance(S1,S2,c-H) :- H{same(S1,S2,V): vertex(V)}H,
  solution(S1;S2), maximumDistance(H), S1 < S2.
```

Fig. B 3. ASP formulation of the Hamming distance between two cliques.

```
:- solution(S1;S2), hammingDistance(S1,S2,H), H > k,
   maximumDistance(H).
```

Fig. B 4. A constraint that forces the distance among any two solution is less than or equal to k.

```
% generate n rooted trees
solution(1..n). vertex(0..2*k). root(2*k).
internal(X) :- vertex(X), not leaf(X).

2 {edge(N,X,Y) : vertex(Y) : X > Y} 2 :- internal(X), solution(N).

reachable(N,X,Y) :- edge(N,X,Y), vertex(X;Y), X > Y, solution(N).
reachable(N,X,Y) :- edge(N,X,Z), reachable(N,Z,Y), solution(N),
   X > Z, vertex(X;Y;Z).
:- vertex(Y), not reachable(N,X,Y), root(X), Y != X, solution(N).
:- reachable(N,X,X), vertex(X), solution(N).

maxY(N,X,Y) :- edge(N,X,Y), edge(N,X,Y1), Y > Y1,
   vertex(Y;Y1), internal(X), solution(N).
:- maxY(N,X,Y), maxY(N,X1,Y1), Y > Y1, X < X1,
   vertex(Y;Y1), internal(X;X1), solution(N).
```

Fig. B 5. A reformulation of the phylogeny reconstruction program of Brooks et. al., to find n distinct phylogenies: Part 1.

```
% ensure that no tree has more than x incompatible characters
g0(N,X,I,S) :- f(X,I,S), informative_character(I),
   essential_state(I,S), solution(N).
g0(N,Y,I,S) :- g0(N,X,I,S), g0(N,X1,I,S), edge(N,Y,X), edge(N,Y,X1),
   X>X1, internal(Y), vertex(X;X1), informative_character(I),
   essential_state(I,S), solution(N).

marked(N,X,I) :- g0(N,X,I,S), informative_character(I),
   vertex(X), essential_state(I,S), solution(N).

g(N,X,I,S) :- g0(N,X,I,S), informative_character(I),
   vertex(X), essential_state(I,S), solution(N).
{g(N,X,I,S): essential_state(I,S)} 1 :- internal(X),
   not marked(N,X,I), informative_character(I), solution(N).

{root_is(N,X,I,S)} :- g(N,X,I,S), vertex(X),
   informative_character(I), essential_state(I,S), solution(N).
:- root_is(N,X,I,S), root_is(N,Y,I,S),
   vertex(X;Y), X < Y, informative_character(I),
   essential_state(I,S), solution(N).
:- root_is(N,X,I,S), g(N,Y,I,S), reachable(N,Y,X), Y > X,
   vertex(X;Y), informative_character(I), essential_state(I,S),
   solution(N).

reachable_is(N,X,I,S) :- root_is(N,X,I,S),
   vertex(X), informative_character(I), essential_state(I,S),
   solution(N).
reachable_is(N,X,I,S) :- g(N,X,I,S), reachable_is(N,Z,I,S),
   edge(N,Z,X), Z > X, vertex(X;Z), informative_character(I),
   essential_state(I,S), solution(N).

incompatible(N,I) :- g(N,X,I,S), not reachable_is(N,X,I,S),
   vertex(X), informative_character(I), essential_state(I,S),
   solution(N).
:- x+1 {incompatible(N,I) : informative_character(I)}, solution(N).
```

Fig. B 6. A reformulation of the phylogeny reconstruction program of Brooks et. al., to find n distinct phylogenies: Part 2.

```
% make sure that these n trees are distinct

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.
```

Fig. B 7. A reformulation of the phylogeny reconstruction program of Brooks et. al., to find n distinct phylogenies: Part 3.

```
dist(0..m).

% compute the nodal distances using distance_v.

% nodaldistance(S,X,Y,T): the nodal distance between X and Y
%    in the S’th tree is T.
nodaldistance(S,X,Y,T) :- tempnodaldistance(S,X,Y,T),
   not notminnodal(S,X,Y,T), solution(S), leaf(X;Y), dist(T).

% distance_v(S,X,Y,T): the distance between the vertex X and
%    its descendant Y is T in the S’th tree.
distance_v(S,X,Y,1) :- edge(S,X,Y), vertex(X;Y), solution(S).
distance_v(S,X,Z,T+1) :- distance_v(S,X,Y,T), edge(S,Y,Z),
   vertex(X;Y;Z), dist(T), solution(S).

% length of a path between vertices X and Y
tempnodaldistance(S,X,Y,T1+T2) :- distance_v(S,CA,X,T1),
   distance_v(S,CA,Y,T2), X<Y, dist(T1;T2), leaf(X;Y),
   vertex(CA), solution(S).

notminnodal(S,X,Y,T1) :- tempnodaldistance(S,X,Y,T1),
   tempnodaldistance(S,X,Y,T2), T2 < T1, leaf(X;Y),
   dist(T1;T2), solution(S).

% compute the differences of nodal distances of each pairs of
%    leaves in each pairs of trees.

diffnodal(P1,P2,X,Y,abs(D1-D2)) :- nodaldistance(P1,X,Y,D1),
   nodaldistance(P2,X,Y,D2), P2>P1, leaf(X;Y), dist(D1;D2),
   solution(P1;P2).

% compute the distance between each pairs of trees.

% distance_t(P1,P2,T) : the distance between (trees) P1 and P2 is T.
tempdistance(P1,P2,0,1,D) :- diffnodal(P1,P2,0,1,D),
   solution(P1;P2), dist(D).
tempdistance(P1,P2,L1,L2,D+K) :- tempdistance(P1,P2,L1,L2-1,D),
   diffnodal(P1,P2,L1,L2,K), L2-L1 > 1, solution(P1;P2),
   leaf(L1;L2), dist(D;K).
tempdistance(P1,P2,L1,L2,D+K) :- tempdistance(P1,P2,L1-1,k,D),
   diffnodal(P1,P2,L1,L2,K), L2 = L1+1, L2 > 1, solution(P1;P2),
   leaf(L1;L2), dist(D;K).

distance_t(P1,P2,T) :- tempdistance(P1,P2,k-1,k,T), dist(T),
   solution(P1;P2).
```

Fig. B 8. A formulation of the nodal distance function D_n in ASP.

```
dist(0..m).

% at each solution N, define reachability of leaf Y from X
reachableN(N,X,Y) :- edge(N,X,Y), vertex(X), leaf(Y), X > Y,
   solution(N).
reachableN(N,X,Y) :- edge(N,X,Z), reachableN(N,Z,Y),
   solution(N), X > Z, vertex(X;Z), leaf(Y).

% at each solution S, assign depths to vertices Y
depth(S,2*k,0) :- solution(S).
depth(S,Y,T+1) :- depth(S,X,T), edge(S,X,Y),
   vertex(X;Y), depthRange(T), solution(S), T<r.

% vertices V1 and v2 have different descendants
diff(N1,V1,N2,V2) :-  solution(N1;N2), vertex(V1;V2), leaf(X),
   N1 < N2, reachableN(N1,V1,X), not reachableN(N2,V2,X).
diff(N1,V1,N2,V2) :-  solution(N1;N2), vertex(V1;V2), leaf(X),
   N1 < N2, not reachableN(N1,V1,X), reachableN(N2,V2,X).

% definition of the function f
fN(N1,V1,N2,V2,1) :- diff(N1,V1,N2,V2), solution(N1;N2),
   vertex(V1;V2), N1 < N2.
fN(N1,V1,N2,V2,0) :- not diff(N1,V1,N2,V2), solution(N1;N2),
   vertex(V1;V2), N1 < N2.

% definition of the function g
gN(0,N1,N2,0) :- solution(N1;N2), N1 < N2.
gN(D+1,N1,N2,D1) :- gN(D,N1,N2,X), solution(N1;N2), N1 < N2,
   depthRange(D;Y), dist(Z;D1;X), maxdepth2(N1,N2,Y), D<Y,
   depthV2(N1,N2,D+1,2*k,Z), w(D+1,M), D1=X+M*Z.
```

Fig. B 9. An ASP formulation of the descendant distance function D_l for two phylogenies: Part 1

```
% depthV2 computes the summation of f(x,y) over all x,y at the same depth
samedepth(N1,V1,N2,V2,D) :- depth(N1,V1,D), depth(N2,V2,D),
        vertex(V1;V2), solution(N1;N2), N1 < N2,  depthRange(D).

depthV(N1,N2,D,W,0,Z) :- solution(N1;N2), N1 < N2,  depthRange(D),
        vertex(W), samedepth(N1,W,N2,0,D), fN(N1,W,N2,0,Z), dist(Z).
depthV(N1,N2,D,W,0,0) :- solution(N1;N2), N1 < N2,  depthRange(D),
        vertex(W), not samedepth(N1,W,N2,0,D).
depthV(N1,N2,D,W,X+1,Z+Z1) :- solution(N1;N2), N1 < N2,
        depthRange(D), vertex(W), depthV(N1,N2,D,W,X,Z),
        samedepth(N1,W,N2,X+1,D), fN(N1,W,N2,X+1,Z1), dist(Z;Z1),
        vertex(X), X<2*k.
depthV(N1,N2,D,W,X+1,Z) :- solution(N1;N2), N1 < N2,  depthRange(D),
        vertex(W), depthV(N1,N2,D,W,X,Z),
        not samedepth(N1,W,N2,X+1,D), dist(Z;Z1), vertex(X), X<2*k.

depthV2(N1,N2,D,0,Z) :- solution(N1;N2), N1 < N2,  depthRange(D),
        depthV(N1,N2,D,0,2*k,Z), dist(Z).
depthV2(N1,N2,D,X+1,Z+Z1) :- solution(N1;N2), N1 < N2,
        depthRange(D), vertex(X), depthV2(N1,N2,D,X,Z),
        depthV(N1,N2,D,X+1,2*k,Z1), dist(Z;Z1), X<2*k.

% definition of the distance function D_n for two phylogenies
depth2(N1,N2,X) :- depth(N1,Y1,X), depth(N2,Y2,X),
        vertex(Y1;Y2), depthRange(X), solution(N1;N2), N1 < N2.
maxdepth2(N1,N2,X) :- depth2(N1,N2,X), not depth2(N1,N2,X+1),
        depthRange(X), solution(N1;N2), N1 < N2.

distance_t(N1,N2,X) :- gN(D,N1,N2,X), solution(N1;N2), N1 < N2,
        dist(X), depthRange(D), maxdepth2(N1,N2,D).
```

Fig. B 10. An ASP formulation of the descendant distance function D_l for two phylogenies: Part 2

```
% distance of a set of phylogenies
notmaxdistance_t(P1,P2,T1) :- distance_t(P1,P2,T1), distance_t(P3,P4,T2),
   T1 < T2, solution(P1;P2;P3;P4), dist(T1;T2).
delta(T1) :- distance_t(P1,P2,T1), not notmaxdistance_t(P1,P2,T1),
   solution(P1;P2), dist(T1).

% constraints on the distance function, for similarity
:- delta(T), dist(T), T > k.
```

Fig. B 11. An ASP formulation of the distance function Δ_D for a set of phylogenies, and the constraints for k-similarity.

```
% effect of moving a block
on(B,L,T1) :- block(B), location(L), moveop(B,L,T), next(T,T1).

% a block can be moved only when it’s clear
:- location(L), block(B), block(B1), time(T),
   moveop(B,L,T), on(B1,B,T).

% any two blocks cannot be on the same block at the same time
:- 2{on(B1,B,T):block(B1)}, time(T), block(B).

% wherever a block is, it’s not anywhere else
non(B,L1,T) :- time(T), location(L1), location(L), block(B),
   on(B,L,T), not eq(L,L1).

% every block is supported by the table
supported(B,T) :- block(B), time(T), on(B,table,T).
supported(B,T) :- block(B), block(B1), time(T), on(B,B1,T),
   supported(B1,T), not eq(B,B1).
:- block(B), time(T), not supported(B,T).

% no concurrency
:- 2{moveop(B,L,T):block(B):location(L)},time(T).

% inertia
on(B,L,T1) :- location(L), block(B), on(B,L,T), not non(B,L,T1),
   next(T,T1).

% initial values and actions are exogenous
1{non(B,L,0),on(B,L,0)}1 :- block(B), location(L).
:- non(B,L,T), on(B,L,T), block(B), location(L), time(T).

{moveop(B,L,T)} :- block(B), location(L), time(T), T < lasttime.

% auxiliary predicates
time(0..lasttime).
next(T,T+1) :- time(T), lt(T,lasttime).

location(L) :- block(L).
location(table).

goal :- time(T), goal(T).
:- not goal.
```

Fig. B 12. Blocks world formulation.

```
solution(1..n).

% effect of moving a block
on(S,B,L,T1) :- block(B), location(L),
   moveop(S,B,L,T), next(T,T1), solution(S).

% a block can be moved only when it's clear
:- location(L), block(B), block(B1), time(T),
   moveop(S,B,L,T), on(S,B1,B,T), solution(S).

% any two blocks cannot be on the same block at the same time
:- 2{on(S,B1,B,T):block(B1)}, time(T), block(B), solution(S).

% wherever a block is, it's not anywhere else
non(S,B,L1,T) :- time(T), location(L1), location(L), block(B),
   on(S,B,L,T), not eq(L,L1), solution(S).

% every block is supported by the table
supported(S,B,T) :- block(B), time(T), on(S,B,table,T), solution(S).
supported(S,B,T) :- block(B), block(B1), time(T), on(S,B,B1,T),
   supported(S,B1,T), not eq(B,B1), solution(S).
:- block(B), time(T), not supported(S,B,T), solution(S).

% no concurrency
:- 2{moveop(S,B,L,T):block(B):location(L)},time(T), solution(S).

% inertia
on(S,B,L,T1) :- location(L), block(B), on(S,B,L,T),
   not non(S,B,L,T1), next(T,T1), solution(S).

% initial values and actions are exogenous
1{non(S,B,L,0),on(S,B,L,0)}1 :- block(B), location(L), solution(S).
:- non(S,B,L,T), on(S,B,L,T), block(B), location(L), time(T), solution(S).

{moveop(S,B,L,T)} :- block(B), location(L), time(T), T < lasttime, solution(S).

% auxiliary predicates
time(0..lasttime).
next(T,T+1) :- time(T), lt(T,lasttime).

location(L) :- block(L).
location(table).

goal(S) :- time(T), goal(S,T), solution(S).
:- not goal(S), solution(S).

% compute distinct plans
different(S1,S2) :- time(T), moveop(S1,X,Y,T), not moveop(S2,X,Y,T),
   solution(S1;S2), block(X), location(Y), S1 < S2.
different(S1,S2) :- time(T), not moveop(S1,X,Y,T), moveop(S2,X,Y,T),
   solution(S1;S2), block(X), location(Y), S1 < S2.
:- not different(S1,S2), solution(S1;S2), S1 < S2.
```

Fig. B 13. A reformulation of the Blocks World program shown in Fig. B 12, to compute n distinct plans.

```
% for every time step T, check that the T'th actions
% of Plans P1 and P2 are different:
different(P1,P2,T) :- moveop(P1,X,Y,T), not moveop(P2,X,Y,T),
   time(T), solution(P1;P2), block(X), location(Y), P1 < P2.
different(P1,P2,T) :- not moveop(P1,X,Y,T), moveop(P2,X,Y,T),
   time(T), solution(P1;P2), block(X), location(Y), P1 < P2.

% and define the hamming distance between two plans P1 and P2
% in terms of these differences:
hammingdistance(P1,P2,H) :- H{different(P1,P2,T): time(T)}H,
   solution(P1;P2), distRange(H), P1 < P2.
```

Fig. B 14. An ASP formulation of the Hamming distance D_h for two plans.

```
somedistance(H) :- hammingdistance(P1,P2,H),
  solution(P1;P2), distRange(H).
notmaxdistance(H1) :- somedistance(H1), somedistance(H2),
  H2 > H1, distRange(H1;H2).
totaldistance(H) :- not notmaxdistance(H),
  distRange(H), somedistance(H).
```

Fig. B 15. An ASP formulation of the distance Δ_h for a set of plans.