Quantifying Solutions in Answer Set Programming

Halit Erdogan HALIT@SABANCIUNIV.EDU
Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, TURKEY

Abstract

Answer Set Programming (ASP) is a declarative programming paradigm oriented towards solving NP-hard problems. Due to the expressive representation language and efficient solvers, ASP can be useful for a wide-range of knowledge-intensive applications from different fields. We present novel methods to quantify answer sets in ASP for extracting preferred solutions only. We show the effectiveness of these methods on real world applications.

1. Introduction

Answer Set Programming (ASP) (Lifschitz, 2008) is a form of declarative programming oriented towards difficult (NP-hard) search problems. A problem is represented as a logic program whose answer sets correspond to the solutions. The answer sets for the given formalism can be computed by special systems called answer set solvers. Due to the expressive representation language and continuous improvements of efficiency of solvers, ASP can be useful for a wide-range of knowledge-intensive applications from different fields such as: developing a decision support system for a Space Shuttle (Nogueira et al., 2001); phylogeny reconstruction (Brooks et al., 2007); multi-agent planning (Son et al., 2009).

Finding *a* solution to the problem that we are interested in is generally the main objective. But many problems have numerous solutions, instead of a single solution. At that case it might be desirable to compute a subset of preferred solutions, instead of computing all the solutions. Consider, for instance, a product configuration problem: Suppose you want to buy a car. There are various constraints on what type of car you want. A product advisor system might consider your constraints and what is available for sale and offer you a set of cars. But you might not have enough time to consider all the cars that are offered by the system. At that case, it might be desirable to quantify the solutions (cars) offered by the system. For example, you might like to see the similar cars similar to a car that you selected. Or you might like see the cars ranked according to a preference function (from cars to real numbers) that you determined. Motivated by such an application we propose novel methods to quantify solutions in answer set programming. We developed a system called CLASP-NK which is a modification of the existing ASP solver CLASP (Gebser et al., 2007). CLASP-NK takes an ASP description of a problem and a preference function on solutions (in C++); and it outputs quantified solutions to the problem based on the preference function. More details and results can be found in (Eiter et al., 2009); since this paper is a summary of (Eiter et al., 2009).

2. Answer Set Programming

The idea of answer set programming (ASP) (Lifschitz, 2008) is to represent a computational problem as a logic program whose answer sets correspond to the solutions of the problem and to find the answer sets for that program by using an answer set solver.

Two kind of rules play the major role in ASP: those that "generate" many answer sets corresponding to possible solutions, and those that "test" the possible solutions and eliminate the ones that does not correspond to a solution.

For example, recall that a *clique* in a graph is a set of pairwise adjacent vertices. Suppose that we are interested in the *cliques* whose size is at least 10 then we can represent the problem in ASP as follows (Lifschitz, 2008):

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

The first rule correspond to the "generate" part. It generates the possible solutions that contains at least 10 vertices (*select* atoms correspond to the selected vertices). The second rule is a constraint that checks whether the selected vertices correspond to a clique. To use this program, we combine it with a description of the graph, such as:

```
vertex(1..99). % 1,...,99 are vertices
edge(3,7).     % 3 is adjacent to 7
. . .
```

When we run this ASP program in an ASP solver, if there exists a clique of at least size 10 then the solver outputs answer sets. Each answer set corresponds to a clique and the *select* atoms in the answer set defines the vertices that constitutes that clique.

3. Similar Solutions in ASP

Consider the *clique* example of the previous section and suppose that the maximum clique size is three. Assume that for each three consecutive vertices we have a clique: $\{\{1,2,3\},\{4,5,6\},\{7,8,9\},...\}$. In addition we have the following two cliques: $\{\{1,2,4\},\{1,2,5\}\}$. Now suppose that we would like to find the three most similar cliques in the graph (a set of three cliques where the number of differentiating atoms (Hamming Distance) are minimum). At that case three most similar cliques of the graph will be: $\{\{1,2,3\},\{1,2,4\},\{1,2,5\}\}$ since they only differ with one vertex.

In this example we quantify the solutions based on their similarity to other solutions. And we accomplish that by defining a distance measure for a set of solutions. We used the simple Hamming Distance but more complex problems might require more complex distance measures. Motivated by such an example we defined the following decision problem:

> n k-SIMILAR 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 $\Delta(S) \leq k$.

In the next section we introduce methods to solve the problem with ASP.

4. Computing Similar Solutions in ASP

Offline Method We can compute a set of n k-similar solutions to a given problem, by computing all solutions in advance and then using some clustering methods to find the similar solutions. The idea is to make clusters of n solutions, measure the distance of the set of solutions in each cluster, and pick the cluster whose distance is less than k.

Online Method 1 The idea of this method is to solve the problem by describing it in ASP. This method reformulates the given program $\mathcal{P}$ to compute n-distinct solutions, formulates the distance function Δ as an ASP program $\mathcal{D}$, and formulates constraints on the distance function as an ASP program $\mathcal{C}$, so that all n k-similar solutions can be extracted from an answer set for the union of these ASP programs, $\mathcal{P} \cup \mathcal{D} \cup \mathcal{C}$.

Online Method 2 This is an approximate method to solve the problem. In this method we do not modify the given ASP program $\mathcal{P}$, but formulate the distance $\Delta(S)$ of a given set S of solutions as an ASP program $\mathcal{D}$, and constraint on the distance function as an ASP program $\mathcal{C}$, so that a k-close solution can be extracted from an answer set for $\mathcal{P} \cup \mathcal{D} \cup \mathcal{C}$. By iteratively computing a k-close solution, we can compute a set of n k-similar solutions.

Online Method 3 This is also an approximate method to solve the problem. This method does not modify the given program, and does not formulate the distance function as an ASP program, but it modifies the ASP solver CLASP (Gebser et al., 2007) to compute all n k-similar solutions at once. As a result, we developed a system called CLASP-NK which is capable of computing n k-similar solutions. CLASP-NK takes the ASP program $\mathcal{P}$ and the C++ definition of Δ as input and outputs n k-similar solutions.

CLASP performs a type of branch and bound algorithm. In each step it decides an atom to be added to the answer set (branch). And according to that atom it propagates other atoms that shall be included in the answer set. Then it checks whether there is a conflict by considering the rules of the program. If there exists such a conflict CLASP learns the conflict (to not to repeat it) and performs a backtracking (bound).

CLASP-NK contains a slight modification on the bounding procedure of CLASP. At each step —after CLASP decides the atoms to include to the answer set— CLASP-NK performs an extra check based on the input distance measure. If the currently selected atoms (at the level we are in) violates the distance measure (in the context of similar solutions it means that it is impossible to compute a solution which is similar to the previously computed solutions if we continue branching) then we set those atoms as conflict and perform a backtracking. Therefore, CLASP-NK is forced to compute a similar solution to the previously computed solutions

5. Computing Similar Phylogenies

Phylogenetics studies the evolutionary relationships between taxonomic units (e.g., species) based on their shared traits. These relations are represented as a tree whose leaves represent the taxa, internal vertices represent their ancestors, and edges represent the relationships between them. Such a tree is called "phylogenetic tree" or "phylogeny". Phylogeny reconstruction is an NP-hard problem and there exists ASP programs that can infer phylogenies (Brooks et al., 2007). But in many cases the phylogeny reconstruction programs output numerous phylogenies that describe the historical evolution of the same species. At those cases experts go over these phylogenies manually to

find the most plausible ones. Instead of computing all the phylogenies, the experts want to compute a set of similar phylogenies to perform better analysis. Therefore we defined n k-similar phylogenies problem analogous to the n k-similar solutions problem. We used the ASP program of (Brooks et al., 2007) to compute a phylogeny (solution), and we used a distance measure from the literature to compute the distance between phylogenies. And we run some experiments to test the methods described in the previous section. The table below shows the performance of each method from the point of view of computation time and memory.

Problem	Offline Method	Online Method 1	Online Method 2	Online Method 3
2 most similar	12.39 sec. 32MB $k = 12$	26.23 sec. 430MB $k = 12$	19.00 sec. 410MB $k = 12$	1.46 sec. 12MB $k = 12$
3 most similar	11.59 sec. 32MB $k = 15$	60.20 sec. 730MB $k = 15$	43.56 sec. 626MB $k = 15$	1.56 sec. 15MB $k = 16$
6 most similar	11.66sec. 32MB $k = 25$	327.28 sec. 1.8GB $k = 25$	178.96 sec. 1.2GB $k = 29$	1.96 sec. 15MB $k = 25$

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

The offline method 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.

6. Conclusion

We introduce one offline and three online methods to compute n k-similar solutions in ASP. We developed a system CLASP-NK (Online Method 3) which is capable of computing n k-similar solutions on the fly. We showed the effectiveness and applicability of our methods on phylogenetics domain.

This paper is a proof-of-principle that CLASP-NK is useful for quantifying solutions in ASP. In this paper, we consider quantifying the solutions based on their similarity to other solutions. CLASP-NK's capabilities are not limited to only computing similar solutions in ASP. We can define any preference function so that CLASP-NK can compute the quantified solutions based on that preference function.

Acknowledgments

I am grateful to my supervisor, Esra Erdem, for excellent advice and encouragement. The central ideas in this work are the product of a close collaboration with Esra Erdem, Thomas Eiter and Michael Fink. I also wish to thank Martin Gebser and Benjamin Kaufmann for their help with CLASP.

References

Brooks, D., Erdem, E., Erdogan, S., Minett, J., & Ringe, D. (2007). Inferring phylogenetic trees using answer set programming. *Autom. Reason.*, *39*, 471–511.

Eiter, T., Erdem, E., Erdogan, H., & Fink, M. (2009). Finding similar or diverse solutions in answer set programming. In (Hill & Warren, 2009), 342–356.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). clasp: A conflict-driven answer set solver. *Proc. of LPNMR* (pp. 260–265). Springer-Verlag.

Hill, P. M., & Warren, D. S. (Eds.). (2009). *Logic programming, 25th international conference, iclp 2009, pasadena, ca, usa, july 14-17, 2009. proceedings*, vol. 5649 of *Lecture Notes in Computer Science*. Springer.

Lifschitz, V. (2008). What is answer set programming? *Proc. of. AAAI* (pp. 1594–1597). AAAI Press.

Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., & Barry, M. (2001). An a-prolog decision support system for the space shuttle. *Proc. of PADL* (pp. 169–183). London, UK: Springer-Verlag.

Son, T., Pontelli, E., & Sakama, C. (2009). Logic programming for multiagent planning with negotiation. In (Hill & Warren, 2009), 99–114.