Lifting Symmetry Breaking Constraints with Inductive Logic Programming (Extended Abstract)

Alice Tarzariol[1], Martin Gebser[1,2], and Konstantin Schekotihin[1]

[1] University of Klagenfurt, Austria
[2] Graz University of Technology, Austria

1 Introduction

Although many combinatorial problems are relatively easy to model with the current programming paradigms, when the size of input instances starts to grow, they become intractable because of the number of possible solution candidates [7]. In order to deal with large problem instances, the ability to encode symmetry breaking constraints in a program becomes an essential skill for the programmers. However, the identification of symmetric solutions and the formulation of constraints that remove only them is a tedious and time-consuming task. As a result, various tools emerged that exploit properties of permutation groups to automatically compute a set of Symmetry Breaking Constraints (SBCs) [13, 12]. For certain combinatorial problems, finding a solution of the original program requires much more time than using an *instance-specific* symmetry breaking approach, which grounds the original program, computes ground SBCs, composes a new extended program, and solves it [8].

Unfortunately, these observations do not hold for advanced programs, where instance-specific symmetry breaking often requires as much time as it takes to solve the original problem. Moreover, whenever an encoding must be changed, for instance, when a customer provides new requirements to a configuration problem, a programmer might need to write all SBCs from scratch. Finally, the instance-specific approach:

- is not transferable, since the knowledge obtained is limited to a single instance;
- might add many redundant ground constraints to the input program, thus, degrading the solving performance;
- uses the computation of permutation group generators, which is itself a combinatorial problem; and
- generates ground SBCs that are often hard to interpret and comprehend.

Our work aims to generalize the process of discarding redundant solution candidates for instances of a target domain. The goal is to lift the SBCs of small problem instances and to obtain a set of interpretable first-order constraints. Such constraints cut the search space while preserving the satisfiability of a problem for the considered instance distribution, which improves the solving performance, especially in the case of unsatisfiability.

2 Approach

We deal with Answer Set Programming (ASP) [3] programs and assume that the instances of a considered problem follow a specific distribution. Our approach relies on the system SBASS [8] to compute the SBCs of a problem instance, and then we encode this information as an Inductive Logic Programming (ILP) task, so that ILASP [9, 1] – an ILP learning system for ASP – will learn a set of (possibly non-ground) SBCs for the initial program.

Given a (small) problem instance, SBASS computes SBCs by determining permutation groups on a graph built over the input program. It reduces symmetry detection to a graph automorphism problem and makes use of the system SAUCY [2, 4], returning a set of group generators that identify equivalence classes in the assignments of ground atoms. In our approach, we exploit SBASS to identify such generators for one (or possibly more) small problem instance(s). We apply the LEX-leader scheme, following the alphabetical order of atoms, to compute non-dominated assignments and to identify the answer sets matching these assignments.

Dominated answer sets that violate the SBCs computed by SBASS are taken as negative examples for ILASP, while the others are positive examples. Following the syntax of ILASP, we define the inclusions and exclusions according to the atoms of group generators that are entailed or falsified by an answer set, respectively. To define an ILP task, in addition to the positive and negative examples, we need to specify the background knowledge and the language bias [11, 5]. The former consists of the input program along with predicates providing the lexicographic order for terms occurring as arguments of atoms. In turn, the language bias defines the search space of the ILP task, consisting of constraints (only) over predicates from the background knowledge, including those giving the lexicographic order for terms.

To guarantee that the learned constraints generalize for the considered instance distribution, for each satisfiable instance, we provide a general positive example, given by empty inclusions and exclusions of atoms, thus requiring the preservation of satisfiability. In case the learned constraints discard all answer sets of a new (satisfiable) problem instance, we backtrack by forgetting the constraints and adding the instance along with a general positive example.

As a first experiment, we applied our approach to the pigeonhole problem. Using only one instance with three pigeon and four holes, we can learn the following constraints:

```
:- maxpigeon(V1), lessthan(V1,V2), assign(V3,V2).
    % Leave holes with a greater label than maxpigeon empty
:- lessthan(V2,V1), lessthan(V1,V3), assign(V1,V3).
    % For all but the first pigeon:
    % Don't assign the pigeon to a hole with greater label
```

Currently, our framework needs to be guided in the selection of problem instances and language bias. However, in the future, we aim to automate this process in a smarter way, so that learned constraints evolve incrementally [10]. Moreover, we aim to tackle more advanced configuration problems, such as the pigeonhole problem enriched with coloring constraints, the partner units and house configuration problems, as well as encodings of hierarchical problem versions, e.g., pigeonhole over pigeonhole problem.

Lastly, we plan to compare our results with other systems that implement SBCs for ASP, like for instance [6].

References

1. The ILASP system for learning answer set programs. `www.ilasp.com`, accessed: 01-10-2020
2. Saucy. `http://vlsicad.eecs.umich.edu/BK/SAUCY/`, accessed: 01-10-2020
3. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
4. Codenotti, P., Katebi, H., Sakallah, K., Markov, I.: Conflict analysis and branching heuristics in the search for graph automorphisms. In: IEEE 25th International Conference on Tools with Artificial Intelligence (2013)
5. Cropper, A., Dumančić, S., Muggleton, S.: Turning 30: New ideas in inductive logic programming. In: Proceedings of the International Joint Conference on Artificial Intelligence (2020)
6. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: On local domain symmetry for model expansion. Theory and Practice of Logic Programming **16**(5-6), 636–652 (2016)
7. Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F., Shchekotykhin, K.: Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). Theory and Practice of Logic Programming (2016)
8. Drescher, C., Tifrea, O., Walsh, T.: Symmetry-breaking answer set solving. AI Communications (2011)
9. Law, M., Russo, A., Broda, K.: The ILASP system for inductive learning of answer set programs. The Association for Logic Programming Newsletter (2020)
10. Lin, D., Dechter, E., Ellis, K., Tenenbaum, J., Muggleton, S.: Bias reformulation for one-shot function induction. In: Proceedings of the European Conference on Artificial Intelligence (2014)
11. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming (1994)
12. Sakallah, K.: Symmetry and satisfiability. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 10, pp. 289–338. IOS Press (2009)
13. Walsh, T.: Symmetry breaking constraints: Recent results. Proceedings of Twenty-Sixth Conference on Artificial Intelligence (2012)