Treewidth-aware Reductions of Normal ASP to SAT– Is Normal ASP Harder than SAT after All? (Extended Abstract)*

Markus Hecher[1,2]

[1] TU Wien, Austria hecher@dbai.tuwien.ac.at
[2] University of Potsdam, Germany

Introduction

Answer Set Programming (ASP) [5] is an active research area of knowledge representation and reasoning. ASP provides a declarative modeling language and problem solving framework [12] for hard computational problems, which has been widely applied. In ASP questions are encoded into rules and constraints that form a program (over atoms), whose solutions are called answer sets.

In terms of computational complexity, the *consistency problem* of deciding the existence of an answer set is well-studied, i.e., the problem is Σ_2^P-complete [9]. Some fragments of ASP have lower complexity though. A prominent example is the class of *head-cycle-free (HCF)* programs, which is a certain generalization of the class of *normal* programs and requires the absence of cycles in a certain graph representation of the program. Deciding whether such a program has an answer set is NP-complete.

There is also a wide range of more fine-grained studies for ASP, also in parameterized complexity [7], where certain (combinations of) parameters [11] are taken into account. In parameterized complexity, the "hardness" of a problem is classified according to the impact of a *parameter* for solving the problem. There, one often distinguishes the runtime dependency of the parameter, e.g., levels of exponentiality [20] in the parameter, required for problem solving. Concretely, under the reasonable *Exponential Time Hypothesis (ETH)* [13], *propositional satisfiability (*SAT*)* is single exponential in the structural parameter treewidth, wheras evaluating *Quantified Boolean formulas* of quantifier depth two is [16] double exponential[3] in the treewidth k.

For ASP there is growing research on treewidth [14,3,4]. Algorithms of these works exploit structural restrictions (in form of treewidth) of a given program, and often run in polynomial time in the program size, while being exponential only in the treewidth. Intuitively, treewidth gives rise to a *tree decomposition*, which allows solving numerous NP-hard problems in parts, cf., divide-and-conquer, and indicates the maximum number of variables one has to investigate in such parts during evaluation. There were also dedicated competitions [8] and notable progresses in SAT [10,6] and other areas [2].

* This is an extended abstract of a paper that appeared at KR'20.

[3] Double exponentiality refers to runtimes of the form $2^{2^{\mathcal{O}(k)}} \cdot n$.

Naturally, there are numerous reductions of ASP (see, e.g., [15,18]) to SAT. These reductions have been investigated in the context of resulting formula size and number of auxiliary variables. However, structural dependency in form of, e.g., treewidth, has not been considered yet. These existing reductions cause only sub-quadratic blow-up in the number of variables (auxiliary variables), which is unavoidable [17] if the answer sets should be preserved (bijectively). However, if one considers the structural dependency in form of treewidth, existing reductions could cause quadratic or even unbounded overhead in the treewidth. On the contrary, we present a novel reduction for HCF programs that increases the treewidth k at most *sub-quadratically* ($k \cdot \log(k)$). This is indeed interesting as there is a close connection [1] between resolution-width and treewidth, resulting in efficient SAT solver runs on instances of small treewidth. As a result, our reduction could improve solving approaches by means of SAT solvers, e.g., [19]. Then, we establish lower bounds under ETH, for exploiting treewidth for concistency of normal programs. This renders normal ASP "harder" than SAT.

Methods and Results

First, we present a novel reduction from HCF programs to SAT, which only requires linearly many auxiliary variables plus a number of auxiliary variables that is linear in the instance size and slightly superexponential in the treewidth of the SAT instance. This is achieved by guiding the whole reduction along a tree decomposition of the program. Inspired by the idea of level mappings [15], we use (local) level mappings for each tree decomposition node. These local level mappings come at the price of losing bijectivity, i.e., there might be duplicate models of the resulting SAT formula for each answer set of the HCF program. However, the reduction only slightly increases the treewidth, i.e., the treewidth of the resulting SAT formula is slightly larger than the treewidth of the program.

Then, we show that certainly we cannot avoid this increase in the treewidth. This is achieved by reducing from a problem that is known to be slightly superexponential [20] to normal ASP and taking care that the treewidth is increased at most linearly. As a result, we establish that under the widely believed ETH, one cannot decide ASP in time $2^{o(k \cdot \log(k))} \cdot n$, with treewidth k and program size n. This is in contrast to the runtime for deciding SAT: $2^{\mathcal{O}(k)} \cdot n$ with treewidth k and size n of the formula. As a result, this establishes that the consistency of normal ASP programs is already harder than SAT using treewidth. Note that this is surprising as both problems are of similar hardness according to classical complexity (NP-complete). In the light of a known result [1] on the correspondence of treewidth and the resolution width applied in SAT solving, this reveals that ASP consistency might be indeed harder than solving SAT. Further, compared to known results restricting to, e.g., modular reductions [15], or involving the need of auxiliary variables [17], this shows that under ETH the increase of treewidth is indeed unavoidable when considering consistency. Therefore this work discusses the complexity of ASP from a different angle and hopefully provides new insights into the hardness of ASP.

References

1. Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. J. Artif. Intell. Res. **40**, 353–373 (2011)
2. Bannach, M., Berndt, S.: Practical access to dynamic programming on tree decompositions. Algorithms **12**(8), 172 (2019)
3. Bichler, M., Morak, M., Woltran, S.: Single-shot epistemic logic program solving. In: IJCAI'18. pp. 1714–1720. ijcai.org (2018)
4. Bliem, B., Morak, M., Moldovan, M., Woltran, S.: The impact of treewidth on grounding and solving of answer set programs. J. Artif. Intell. Res. **67**, 35–80 (2020)
5. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011). https://doi.org/10.1145/2043174.2043195
6. Charwat, G., Woltran, S.: Expansion-based QBF solving on tree decompositions. Fundam. Inform. **167**(1-2), 59–92 (2019)
7. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Dániel Marx, M.P., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)
8. Dell, H., Komusiewicz, C., Talmon, N., Weller, M.: The pace 2017 parameterized algorithms and computational experiments challenge: The second iteration. In: IPEC'17. pp. 30:1—30:13. LIPIcs, Dagstuhl Publishing (2017)
9. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. **15**(3–4), 289–323 (1995). https://doi.org/10.1007/BF01536399
10. Fichte, J.K., Hecher, M., Zisser, M.: An improved gpu-based SAT model counter. In: CP'19. LNCS, vol. 11802, pp. 491–509. Springer (2019)
11. Fichte, J.K., Kronegger, M., Woltran, S.: A multiparametric view on answer set programming. Ann. Math. Artif. Intell. **86**(1-3), 121–147 (2019)
12. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool (2012). https://doi.org/10.2200/S00457ED1V01Y201211AIM019
13. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. of Computer and System Sciences **63**(4), 512–530 (2001). https://doi.org/10.1006/jcss.2001.1774
14. Jakl, M., Pichler, R., Woltran, S.: Answer-set programming with bounded treewidth. In: IJCAI'09. vol. 2, pp. 816–822 (2009)
15. Janhunen, T.: Some (in)translatability results for normal logic programs and propositional theories. Journal of Applied Non-Classical Logics **16**(1-2), 35–86 (2006)
16. Lampis, M., Mitsou, V.: Treewidth with a quantifier alternation revisited. In: IPEC'17. vol. 89, pp. 26:1–26:12. Dagstuhl Publishing (2017)
17. Lifschitz, V., Razborov, A.A.: Why are there so many loop formulas? ACM Trans. Comput. Log. **7**(2), 261–268 (2006)
18. Lin, F., Zhao, J.: On tight logic programs and yet another translation from normal logic programs to propositional logic. In: IJCAI'03. pp. 853–858. Morgan Kaufmann (August 2003)
19. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artif. Intell. **157**(1-2), 115–137 (2004). https://doi.org/10.1016/j.artint.2004.04.004, https://doi.org/10.1016/j.artint.2004.04.004
20. Lokshtanov, D., Marx, D., Saurabh, S.: Slightly superexponential parameterized problems. In: SODA'11. pp. 760–776. SIAM (2011)