Proceedings of the Sixteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2018)

Exploiting Treewidth for Counting Projected Answer Sets*

Johannes K. Fichte
TU Dresden
International Center for Computational Logic
Fakultät Informatik,
01062 Dresden, Germany

Markus Hecher
TU Wien
Insitute of Logic and Computation
Favoritenstraße 9-11 / E192
1040 Vienna, Austria

Abstract

Answer Set Programming (ASP) is an active research area of artificial intelligence. We consider the problem *projected answer set counting (*#PDA*)* for disjunctive propositional ASP. #PDA asks to count the number of answer sets with respect to a given set of *projected atoms*, where multiple answer sets that are identical when restricted to the projected atoms count as only one projected answer set. Our approach exploits small treewidth of the primal graph of the input instance. Finally, we state a hypothesis (3ETH) that one cannot solve 3-QBF in polynomial time in the instance size while being significantly better than triple exponential in the treewidth. Taking 3ETH into account, we show that one can not expect to solve #PDA significantly better than triple exponential in the treewidth.

Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011) is an active research area of artificial intelligence. In ASP, questions are encoded into rules and constraints that form a program over atoms. A disjunctive propositional ASP program is of the form $a_1 \vee \cdots \vee a_\ell \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$ where a_1, …, a_n are distinct propositional atoms for non-negative integers ℓ, m, n such that $\ell \leq m \leq n$. Solutions to the program are so-called answer sets (Brewka, Eiter, and Truszczyński 2011). Recently, the problem *projected answer set counting (*#PDA*)* has received renewed attention (Aziz 2015). #PDA asks to count the number of answer sets with respect to a given set of projected atoms. Particularly, we consider multiple answer sets that are identical when reduced to the projected atoms as only one projected answer set. If we take all atoms as projected, then #PDA is #·coNP-complete (Fichte et al. 2017) and if there are no projected atoms then it is Σ_2^p-complete. However, in general we have the following complexity.

Proposition 1 ($\star$[1])**.** *The problem* #PDA *is #·*$\Sigma_2 P$*-complete.*

A way to solve computationally hard problems is to employ parameterized algorithmics (Cygan et al. 2015), which exploits certain structural restrictions (parameter) in a given input instance and aims to solve in runtime polynomial in the input size and exponential in the parameter. Here, we consider the treewidth of a graph representation of the given input program as structural restriction, namely the *treewidth of the primal graph* (Jakl, Pichler, and Woltran 2009). The *primal graph* G_Π of a program Π has the atoms of Π as vertices and an edge $\{a, b\}$ if there exists a rule $r \in \Pi$ and $a, b \in \mathrm{at}(r)$. Generally speaking, treewidth measures the closeness of a graph to a tree, based on the observation that problems on trees are often easier to solve than on arbitrary graphs. Formally, a *tree decomposition (TD)* of graph $G = (V, E)$ is a pair $\mathcal{T} = (T, \chi)$, where T is a rooted tree, and χ a mapping that assigns to each node t in T a set $\chi(t) \subseteq V$, called a *bag*, such that the following conditions hold: (i) $E \subseteq \bigcup_{t \text{ in } T} \{\{u, v\} \mid u, v \in \chi(t)\}$; and (ii) for each r, s, t, such that s lies on the path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$. Then, $\mathrm{width}(\mathcal{T}) := \max_{t \text{ in } T} |\chi(t)| - 1$. The *treewidth* of G is the minimum width over all TDs of G.

Dynamic Programming on TDs. Algorithms exploiting tree decompositions typically proceed along a TD $\mathcal{T}$ where at each node of $\mathcal{T}$ information is stored in a table by local algorithm $\mathbb{A}$. In particular, the dynamic programming approach for ASP performs the following steps for a given program Π (Fichte et al. 2017):

1. Compute a TD $\mathcal{T} = ((N, E_T, r), \chi)$ of primal graph G_Π.
2. $\mathtt{DP}_\mathbb{A}$: Run dynamic programming with algorithm $\mathbb{A}$ and traverse $\mathcal{T}$ in post-order. At each node $t \in N$ compute a new table by executing algorithm $\mathbb{A}$, which transforms tables computed for children of t, and enforces that only rows in the table are stored that satisfy the input program restricted to atoms that occur in the bag t and can be extended to an answer set of the program restricted to atoms that occur in bags below t. Then, in particular, at root r a row can be extended to an answer set of program Π.
3. Print the result by interpreting the table for the root r of $\mathcal{T}$.

Dynamic Programming for #PDA

In this paragraph, we lift a very recent approach to count projected models for propositional satisfiability (Fichte et al.

*The work has been supported by the Austrian Science Fund (FWF) Grant Y698 and the German Science Fund (DFG) Grant HO 1294/11-1. The authors are also affiliated with the University of Potsdam, Germany.

[1]Due to space limitations, proofs of statements marked with "$\star$" have been omitted.

2018) to answer set programming and the problem #PDA. Assume that we have given program Π and set P of projection atoms. First, we construct the primal graph G_Π and a TD $\mathcal{T}$ of G_Π. Then, a central idea is to traverse a given TD multiple times. We traverse $\mathcal{T}$ for the first time and run algorithm $\mathtt{DP}_\mathbb{A}$ with the local algorithm $\mathbb{A} = \mathbb{PRIM}$, which solves counting for ASP (Fichte et al. 2017; Jakl, Pichler, and Woltran 2009). After this step, the resulting tables may contain also rows, which cannot be extended to answer sets. We *purge* these rows in a second tree traversal, since they can also not be extended to a projected answer set. Actually, having only rows that can be extended to answer sets is a necessary assumption for the final traversal. In the final traversal, we produce the projected counts for the rows. In particular, to *bound the number* of these stored counts by the corresponding bag sizes, we exploit the combinatorial principle of inclusion and exclusion (PIE) (Graham, Grötschel, and Lovász 1995) twice and interleaved. To this end, we construct for each node t of the TD $\mathcal{T}$ equivalence classes of rows of tables at t of the previous traversal. The equivalence classes are built with respect to the set P of projection atoms restricted to the bag of the node t. Intuitively, when computing these counts for a node t, applying the PIE first transforms counts from tables of child nodes of t. Then, applying PIE for the second time adapts them by once again taking into account the equivalence classes of rows of tables at t. The actual algorithm is more evolved than in the propositional setting, however, due to space restrictions we omit details.

Theorem 1 ($\star$)**.** *Let Π be a disjunctive program and P be a set of projection atoms where the treewidth of G_Π is bounded by some integer k. Then, there is an algorithm that solves the problem* #PDA *in time $\mathcal{O}(2^{2^{2^{k+3}}} \cdot \|\Pi\| \cdot \gamma(\|\Pi\|))$*[2].

Idea. An algorithm that implements the first traversal above requires to store at most double exponentially many rows in the treewidth for each bag (Fichte et al. 2017). The second step vacuously runs in time double exponential. Finally, applying the inclusion exclusion principle increases the runtime by at most one exponent, since we have to store the counters for the potential equivalence classes, c.f., (Fichte et al. 2018).

A natural question is whether we can significantly improve this runtime. To this end, one would usually like to take the *exponential time hypothesis (ETH)* into account, which states that there is some real $s > 0$ such that we cannot decide satisfiability of a given 3-CNF formula F in time $2^{s \cdot |F|} \cdot \|F\|^{\mathcal{O}(1)}$. While we obtain lower bounds from the ETH for SAT (single-exponential) and for $\forall\exists$-SAT/$\exists\forall$-SAT (double-exponential), to our knowledge it is unproven whether this extends to $\forall\exists\forall$-SAT and $\exists\forall\exists$-SAT (triple-exponential). Since it was anticipated by Marx and Mitsou (2016) that it follows just by assuming ETH, we state this as hypothesis. In particular, they claimed that quantifier alternations are the reason for large dependence on treewidth. However, proofs can be quite involved, trading an additional alternation for exponential compression.

[2]We assume $\gamma(n)$ to be the number of operations that are required to multiply two n-bit integers, which can be done in time $n \cdot \log n \cdot \log\log n$ (Knuth 1998).

Hypothesis 1 (3ETH)**.** *The $\forall\exists\forall$-SAT problem for a QBF Q of treewidth k can not be decided in time $2^{2^{2^{o(k)}}} \cdot \|Q\|^{o(k)}$.*

Using this hypothesis, we obtain the following result.

Theorem 2 ($\star$)**.** *Unless 3ETH fails, the problem* #PDA *for disjunctive programs Π of treewidth k of G_Π cannot be solved in time $2^{2^{2^{o(k)}}} \cdot \|\Pi\|^{o(k)}$.*

Conclusion and Future Work

We considered counting the projected answer sets (#PDA) of disjunctive programs. Our approach employs dynamic programming in order to exploit small treewidth of the primal graph of the input program. We stated a hypothesis that one cannot solve 3-QBF in polynomial time in the instance size while being significantly better than triple exponential in the treewidth (3ETH). Finally, we showed that under 3ETH it is not possible to solve #PDA in time double exponential of the treewidth. We believe that our approach works for other hard combinatorial problems, such as circumscription (Durand, Hermann, and Kolaitis 2005), QBF (Charwat and Woltran 2016), or default logic (Fichte, Hecher, and Schindler 2018).

References

Aziz, R. A. 2015. *Answer Set Programming: Founded Bounds and Model Counting*. Ph.D. Dissertation, The University of Melbourne.

Bodlaender, H. L., and Kloks, T. 1996. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms* 21(2).

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Comm. of the ACM* 54(12).

Charwat, G., and Woltran, S. 2016. Dynamic programming-based QBF solving. In *QBF'16*.

Cygan, M.; Fomin, F. V.; Kowalik, Ł.; Lokshtanov, D.; Dániel Marx, M. P.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer Verlag.

Durand, A.; Hermann, M.; and Kolaitis, P. G. 2005. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science* 340(3).

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2017. Answer set solving with bounded treewidth revisited. LPNMR'17.

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2018. Exploiting treewidth for projected model counting and its limits. SAT'18.

Fichte, J. K.; Hecher, M.; and Schindler, I. 2018. Default Logic and Bounded Treewidth. LATA'18.

Graham, R. L.; Grötschel, M.; and Lovász, L. 1995. *Handbook of combinatorics*, volume I. Elsevier.

Jakl, M.; Pichler, R.; and Woltran, S. 2009. Answer-set programming with bounded treewidth. *IJCAI'09*.

Knuth, D. E. 1998. How fast can we multiply? In *The Art of Computer Programming*, volume 2.

Marx, D., and Mitsou, V. 2016. Double-Exponential and Triple-Exponential Bounds for Choosability Problems Parameterized by Treewidth. ICALP'16.