Answer Set Solving exploiting Treewidth and its Limits[⋆]

Markus Hecher

Institute of Logic and Computation, TU Wien, Austria, hecher@dbai.tuwien.ac.at
Keywords: Tree Decompositions, Dynamic Programming, Fixed-Parameter Tractability, Answer-Set Programming, Parameterized Complexity Theory
Advisor: Stefan Woltran, TU Wien, Austria
Co-Advisor: Torsten Schaub, University of Potsdam, Germany

Abstract Parameterized algorithms have been subject to extensive research of recent years and allow to solve hard problems by exploiting a parameter of the corresponding problem instances. There, one goal is to devise algorithms where the runtime is exponential exclusively in this parameter. One particular well-studied structural parameter is treewidth. Typically, a parameterized algorithm utilizing treewidth takes or computes a tree decomposition, which is an arrangement of a graph into a tree, and evaluates the problem in parts by dynamic programming on the tree decomposition. In my thesis, the goal is to exploit treewidth in the context of Answer Set Programming (ASP), a declarative modeling and solving framework, which has been successfully applied in several application domains and industries for years. So far, we presented algorithms for ASP for the full ASP-Core-2 syntax, which is competitive especially when it comes to counting answer sets. Since dynamic programming on tree decomposition lends itself well to counting, we designed a framework for projected model counting, which applies to ASP, abstract argumentation and even to problems higher in the polynomial hierarchy. Given standard assumptions in computational complexity, we established a novel methodology for showing lower bounds, and we showed that most worst-case runtimes of our algorithms cannot be significantly improved.

1 Introduction

Parameterized algorithms [8] have attracted considerable interest in recent years and allow to solve hard combinatorial problems by utilizing a certain parameter of the problem instance. Of particular interest is to devise algorithms where the runtime is polynomial and additionally depends on some computable function in the parameter. One structural and extensively studied parameter is treewidth [2,29]. Intuitively, treewidth measures the closeness of a graph to a tree based on the observation that problems on trees are often easier than on arbitrary graphs. A (parameterized) algorithm utilizing treewidth typically solves problem instances

[⋆] This work has been supported by Austrian Science Fund (FWF): Y698. The author is also affiliated with the University of Potsdam, Germany.

by *dynamic programming (DP)*. Thereby it takes a *tree decomposition*, which is an arrangement of a graph (representation) of the given problem instance into a tree, and then evaluates the problem in parts, where each tree decomposition node forms such a part. These dynamic programming algorithms are then sensitive to treewidth, which provides an upper bound on the worst-case runtime needed for evaluating each problem part induced by a tree decomposition node of an (optimal) tree decomposition. However, in practice, solvers based on this idea can produce results for certain problems [5,24] up to treewidth 80. Further, there are observations that instances relevant for certain applications and different problems sometimes have small treewidth. One particular problem, for which dynamic programming algorithms on tree decompositions were investigated [22] is Answer Set Programming. *Answer Set Programming (ASP)* [3] is a logic-based declarative modeling language and problem solving framework where selected models, the *answer sets*, of a given ASP program directly represent the solutions to the modeled problem. ASP has been applied in several application domains, which accelerates the search for alternative solving methods. This raises then the question whether exploiting structural parameters, e.g., treewidth, improves the performance of evaluation of ASP programs, which forms the topic of the thesis. Jakl et al. [22] established a DP algorithm for disjunctive ASP that is linear in the size of the (ground) ASP program, but double exponential in the treewidth of a certain graph representing of the program. There is an implementation called *DynASP solver* [26] that adheres to this idea. The goal of my thesis is to continue this line of research. In our work [13,14], we presented an evaluation of extended ASP programs based on the full ASP-Core-2 [4] syntax. Further, we also showed that the double exponential runtime in the treewidth cannot be avoided for ASP in general, unless the widely believed exponential time hypothesis (ETH) fails. This result is based on our novel methodology [17] for showing such lower bounds for problems even higher in the polynomial hierarchy. Currently, we are investigating *hybrid parameterized solving* techniques for ASP, where we aim to improve solving by a combination of both monolithic and parameterized solvers.

2 Background

Tree Decompositions (TDs). Tree decompositions are built for a given graph and are a tree-like representation of the graph. These tree decompositions are trees consisting of nodes, where each node contains certain vertices of the given graph. The so-called *width* of a given tree decomposition corresponds to the largest number of vertices that is contained in one node (minus one). The parameter *treewidth* captures the “tree-likeness” of the given graph and corresponds to the smallest width among all tree decompositions for the graph. The treewidth intuitively reveals, how hard the given graph is when solving a certain problem. The smaller the treewidth of a given graph, the more “tree-like” the graph and thus typically easier to solve problems on the graph. In particular, the treewidth of a graph that is a tree corresponds to one. Formally, let $G = (V, E)$ be an undirected graph, T a rooted tree, and χ a labeling function that maps every

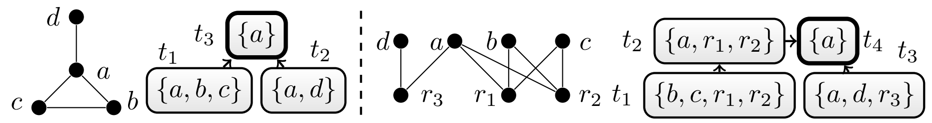

Figure 1. Graph G_1 with a TD of G_1 (left) and graph G_2 with a TD of G_2 (right).

node t of T to a subset $\chi(t) \subseteq V$ called the *bag* of t. Then the pair $\mathcal{T} = (T, \chi)$ is called a *tree decomposition (TD)* [28] of G if (i) for each $v \in V$, there exists a t in T, such that $v \in \chi(t)$; (ii) for each $\{v, w\} \in E$, there exists t in T, such that $\{v, w\} \subseteq \chi(t)$; and (iii) for each r, s, t of T, such that s lies on the unique path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$.

Example 1. Figure 1(left) shows a graph G_1 and a TD of G_1. The width of the TD of G_1 is 2, which coincides with the treewidth of G_1 (by a simple graph property [2]). Figure 1(right) presents a graph G_2 and a TD of G_2.

Dynamic Programming. Tree decompositions allow to tackle hard problems by evaluating a certain problem-dependent graph representation of a problem instance in parts, thereby being sensitive to the treewidth of the instance. This evaluation is done by means of dynamic programming (on the tree decomposition), where the tree decomposition is traversed in post-order, and each tree decomposition node reflects a certain part of the problem instance. During the traversal, one stores intermediate results in a table for each node. Thereby, each node uses and transforms intermediate results of its child nodes and computes solutions to the corresponding problem part for the node. The dynamic programming algorithm relies on properties of the tree decomposition in order to ensure that a non-empty table for the root node guarantees that a problem solution was found. Using this exact idea, one can also define algorithms to enumerate and count solutions. The runtime of these dynamic programming algorithms is polynomial in the instance size, but additionally depends on a function $f(k)$ for some computable function f, where k is the parameter treewidth. If the parameter k is reasonably small, such an algorithm could outperform classical ASP solvers, which require exponential runtime in the instance size in the worst-case. Since f might be exponential, we aim at classification of problems according to the function f that is required to solve the problem using parameter treewidth. These lower bounds are typically dependent on the widely believed *exponential time hypothesis (ETH)*, which implies that there is no algorithm for *Boolean satisfiability (SAT)* such that satisfiability of a given formula with n variables can be decided in time $2^{o(f(n))} \cdot n^{\mathcal{O}(1)}$.

Answer Set Programming (ASP). ASP is a rule-based modeling and problem solving framework, where rules can contain (first-order) variables, which are instantiated by the *grounder*. The grounder is responsible for producing a *(ground) ASP program*, which is a set of rules obtained by eliminating variables of a given *non-ground* program. Modern grounders parse non-ground programs that adhere to the *ASP-Core-2* [4] syntax and output (among others) ground programs in

SModels intermediate format [4,20]. The main interest of this research proposal concerns the solving, i.e., how to efficiently evaluate an ASP program.

3 Research

First, we discuss the goals of the thesis. Then, we provide a detailed description of achievements and the lessons learned so far. Finally, we give an outlook about ongoing and future work.

3.1 Goals

The thesis shall address and achieve the following main goals.

- Design ASP solving algorithms and techniques that utilize the structural parameter treewidth in order to efficiently solve ASP-Core-2 programs. These algorithms shall be extended to problem variants and extensions later.
- Implement these ideas in a solver and compete against other ASP solvers and the related DynASP system.
- Investigate the theoretical limitations that any of these solvers cannot evade given reasonable assumptions in computational complexity. The thereby obtained results might depend on certain fragments of ASP and could be crucial to further improve the solver.
- Generalize and apply these findings in a broader, general context. Especially applications in artificial intelligence might benefit from the outcome of this research. In this regard also other structural parameters could be considered.

3.2 Achieved Results

First of all, we established dynamic programming algorithms [13] for decision and counting problems related to ASP[1] using the full ASP-Core-2 syntax. Thereby we adapted the existing DynASP solver [26] resulting in the DynASP2 system. This covers interoperation with state-of-the-art grounders, and handling of the SModels intermediate format including also optimization statements. The new system DynASP2 conceptually follows the approach of the original implementation, where tree decomposition(s) of a certain graph representation of a given ground program are prepared[2]. The resulting tree decomposition is then traversed in a bottom-up manner (using post-order traversals) to evaluate the program. We implemented several variants of such algorithms, based on the input program's graph representation. One such graph representation is the *primal graph representation*, where atoms are vertices, and atoms appearing together in a rule have an edge between them. Further, we also investigated algorithms based on the *incidence graph representation*, where atoms and rules are vertices, and when an atom

[1] These algorithms were later also generalized to the formalism of default logic [18].

[2] Later we also investigated the more general approach of fractional hypertree decompositions [11], which could be relevant for ASP as well.

appears in a rule there is an edge between them. Then, we improved the whole approach, where we presented dynamic programming algorithms that rely on multiple passes (where the tree decomposition is traversed multiple times). This new approach of traversing in multiple passes, which extends an existing work on two passes [1], allows then to improve existing data structures of DynASP2. In particular, one can benefit from early pruning of data after each traversal, which then allows for efficient solving in practice [14], given dedicated data structures that heavily rely on preventing redundancy using pointers. Given ASP programs of small treewidth, DynASP2 proved to be competitive in the setting of *model counting*, a central problem in areas like machine learning, statistics, probabilistic reasoning and combinatorics. Examples of such applications are identifying the reliability of energy infrastructure [9], recognizing spam [25,30], learning preference distributions [7], or carrying out patient case simulations [27]. When counting answer sets, our approach conceptually has a big advantage: it does not require to materialize the full answer sets in order to count them. This ensures huge speed-ups against classical ASP systems like clasp [20]. However, our implementation was also able to beat existing SAT model counters, i.e., solvers that count the number of models of a Boolean formula. Benchmarks indicate that the performance of model counting using a popular fragment of quantified Boolean formulas with quantifier depth two (2-QBFs) is competitive as well.

Model counting proved to be a rather successful application of dynamic programming on tree decompositions, since it lends itself well to parallelization. Indeed, using a novel approach on modern graphics computing units (GPUs), we were able to compete existing model counters in a comprehensive benchmark evaluation using all known state-of-the-art systems for model counting of Boolean formulas and weighted variants. We recently improved the resulting system gpusat [19] by a new architecture involving data compression, dedicated data structures, optimized counters and customized tree decompositions.

Later, we generalized our algorithms to *projected model counting*, where models that are identical with respect to a given set of projection variables, i.e., when "projected" to these projection variables, count as one (projected) model. Formally, given a Boolean formula F and a set P of projection variables, which is a subset of the variables appearing in F, the task of projected model counting corresponds to computing $|\{M \cap P \mid M \text{ is a model of } F\}|$, where a model of F is also a subset of the variables of F. To this end, we designed an algorithm [15] that works in multiple passes for SAT. The idea was to propose one dynamic programming algorithm that relies on previous passes that solve the base (decision) problem according to certain conditions. Then, this algorithm takes the results of the previous passes and performs projected counting on top of it. Unfortunately, the algorithm might result in an exponentially increased runtime (in the treewidth) of the previous pass in the worst case. As an example, although SAT can be solved in single-exponential runtime in the treewidth (while still being polynomial in the instance size), projected model counting on SAT requires double-exponential runtime in the treewidth using our algorithm. However, we also considered the exponential time hypothesis (ETH), a standard assumption in computational

Problem	Restriction	Upper Bound	Lower Bound (ETH)
SAT, #SAT	-	$2^{\mathcal{O}(k)} \cdot poly(\lvert I \rvert)$ [31]	$2^{\Omega(k)} \cdot poly(\lvert I \rvert)$ [21]
ASP, #ASP	tight	$2^{\mathcal{O}(k)} \cdot poly(\lvert I \rvert)$ [16]	$2^{\Omega(k)} \cdot poly(\lvert I \rvert)$ [21]
ASP, #ASP	normal, HCF	$2^{\mathcal{O}(k \cdot log(k))} \cdot poly(\lvert I \rvert)$ [16]	$2^{\Omega(k)} \cdot poly(\lvert I \rvert)$ [21]
ASP, #ASP	disjunctive	$2^{2^{\mathcal{O}(k)}} \cdot poly(\lvert I \rvert)$ [22]	$2^{2^{\Omega(k)}} \cdot poly(\lvert I \rvert)$ [13]
Proj.#SAT	-	$2^{2^{\mathcal{O}(k)}} \cdot poly(\lvert I \rvert)$ [15]	$2^{2^{\Omega(k)}} \cdot poly(\lvert I \rvert)$ [15]
Proj.#ASP	tight	$2^{2^{\mathcal{O}(k)}} \cdot poly(\lvert I \rvert)$[16]	$2^{2^{\Omega(k)}} \cdot poly(\lvert I \rvert)$ [16]
Proj.#ASP	normal, HCF	$2^{2^{\mathcal{O}(k \cdot log(k))}} \cdot poly(\lvert I \rvert)$ [16]	$2^{2^{\Omega(k)}} \cdot poly(\lvert I \rvert)$ [16]
Proj.#ASP	disjunctive	$2^{2^{2^{\mathcal{O}(k)}}} \cdot poly(\lvert I \rvert)$ [16]	$2^{2^{2^{\Omega(k)}}} \cdot poly(\lvert I \rvert)$ [16]
QSAT, #QSAT	3-QBF	$2^{2^{2^{\mathcal{O}(k)}}} \cdot poly(\lvert I \rvert)$ [6]	$2^{2^{2^{\Omega(k)}}} \cdot poly(\lvert I \rvert)$ [17]
Proj.#QSAT	2-QBF	$2^{2^{2^{\mathcal{O}(k)}}} \cdot poly(\lvert I \rvert)$ [15]	$2^{2^{2^{\Omega(k)}}} \cdot poly(\lvert I \rvert)$ [17]
QSAT, #QSAT	ℓ-QBF	$\mathrm{t}(\ell, \mathcal{O}(k)) \cdot poly(\lvert I \rvert)$ [15]	$\mathrm{t}(\ell, \Omega(k)) \cdot poly(\lvert I \rvert)$ [17]
Proj.#QSAT	$(\ell - 1)$-QBF	$\mathrm{t}(\ell, \mathcal{O}(k)) \cdot poly(\lvert I \rvert)$ [15]	$\mathrm{t}(\ell, \Omega(k)) \cdot poly(\lvert I \rvert)$ [17]

Table 1. Overview of upper and lower bounds using treewidth k of the primal graph of instance I. We assume ℓ to be an integer with $\ell \geq 2$. The column "Restriction" refers to certain fragments of the problem. Tower $\mathrm{t}(\ell, k)$ refers to a tower of iterated exponentials of 2 of height ℓ with k on the top. "SAT", "ASP" and "QSAT" refer to the corresponding decision problems, whereas "#SAT", "#ASP" and "#QSAT" indicate counting problems. "Proj. #SAT" refers to projected model counting and "Proj. #ASP" as well as "Proj. #QSAT" refer to the counting problems involving projection for ASP as well as QSAT, respectively.

complexity. Our results reveal that if one assumes ETH, one cannot significantly improve the worst-case runtime of this problem. We also generalized our results of projected model counting to abstract argumentation [12] and ASP [10,16], where we provided algorithms and lower bounds based on ETH for fragments of ASP (for decision, counting and projected model counting problems). Some of these lower bounds were achieved by relying on our recent generalization of a result for deciding validity (QSAT) of 2-QBFs with quantifier depth 2 to arbitrary ℓ-QBFs of quantifier depth ℓ, which provides a novel methodology for lower bounds of problems located higher in the polynomial hierarchy [17].

Table 1 shows upper and lower bounds obtained so far. For more details on the results and algorithms, sometimes including implementations and experimental results, we refer to the literature as mentioned in the table.

3.3 Current and Future Work

Given our recent dynamic programming algorithms [16] that utilize treewidth for certain fragments of ASP, we consider an implementation, which will be added to DynASP. Especially in the context of *hybrid parameterized solving*, which aims at the interplay between existing monolithic solvers (e.g., clasp) and approaches based on exploiting (structural) parameters, there is still room for improvement. There are already existing systems of this kind, e.g., *dynqbf* [5], which is capable of deciding validity of QBFs up to treewidth 80 with this idea. Since DynASP can be used up to treewidth 14, and existing reductions from ASP to QBF seem to fail for dynqbf, one might consider a new implementation DynASP3, which provides the best of monolithic solving (e.g., clasp) and parameterized solving (e.g., DynASP2).

At the moment I am focusing on an efficient implementation of our algorithm for projected model counting, since it seems that the mentioned worst-case result does not reflect the average case. In particular, I am evaluating the prospect of a potential implementation targeted on alternative hardware, e.g., GPUs, and potential strategies towards projected model counting solvers for ASP. This new type of hardware raises interesting questions concerning alternative data structures and methods to exploit parallelism efficiently.

Further, we are permanently improving on our methodology [17] for lower bounds, and the idea is to provide a catalog of simple, alternative proofs based on our methodology for existing lower bounds for treewidth under ETH. It is for example still open, whether our methodology can be applied to easily show lower bounds that are called slightly superexponential [23] (in the treewidth), which is between single- and double-exponential.

References

1. B. Bliem, G. Charwat, M. Hecher, and S. Woltran. D-FLAT^2: Subset Minimization in Dynamic Programming on Tree Decompositions Made Easy. *Fundam. Inform.*, 147(1):27–61, 2016.
2. H. L. Bodlaender and A. M. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.
3. G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.
4. F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-core-2 input language format, 2013.
5. G. Charwat and S. Woltran. Expansion-based QBF solving on tree decompositions. Accepted for publication in Fundamenta Informaticae. To appear, 2019.
6. H. Chen. Quantified constraint satisfaction and bounded treewidth. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, volume IOS Press, pages 161–170, 2004.
7. A. Choi, G. Van den Broeck, and A. Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In Q. Yang, editor, *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI'15).* The AAAI Press, 2015.
8. M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms.* Springer, 2015.
9. L. Dueñas-Osorio, K. S. Meel, R. Paredes, and M. Y. Vardi. Counting-based reliability estimation for power-transmission grids. In S. P. Singh and S. Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)*, pages 4488–4494, San Francisco, CA, USA, Feb. 2017. The AAAI Press.
10. J. K. Fichte and M. Hecher. Exploiting treewidth for counting projected answer sets. In *KR'18*, pages 643–644. The AAAI Press, 2018. Extended abstract.
11. J. K. Fichte, M. Hecher, N. Lodha, and S. Szeider. An SMT approach to fractional hypertree width. In *CP'18*, volume 11008 of *LNCS*, pages 109–127. Springer, 2018.

12. J. K. Fichte, M. Hecher, and A. Meier. Counting complexity for reasoning in abstract argumentation. In *AAAI*, pages 2827–2834. AAAI Press, 2019.
13. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer set solving with bounded treewidth revisited. In *LPNMR'17*, volume 10377 of *LNCS*, pages 132–145. Springer, 2017.
14. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. DynASP2.5: Dynamic programming on tree decompositions in action. In *IPEC'17*, volume 89 of *LIPIcs*, pages 17:1–17:13. Dagstuhl Publishing, 2017.
15. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Exploiting treewidth for projected model counting and its limits. In *SAT'18*, volume 10929 of *LNCS*, pages 165–184. Springer, 2018.
16. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer set solving with bounded treewidth revisited. In *LPNMR*, volume 11481 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2019.
17. J. K. Fichte, M. Hecher, and A. Pfandler. TE-ETH: Lower Bounds for QBFs of Bounded Treewidth. 2019. Under review. Preliminary version available at `https://tinyurl.com/y7wnvu6w`.
18. J. K. Fichte, M. Hecher, and I. Schindler. Default Logic and Bounded Treewidth. In *LATA'18*, volume 10792 of *LNCS*, pages 130–142. Springer, 2018.
19. J. K. Fichte, M. Hecher, S. Woltran, and M. Zisser. Weighted Model Counting on the GPU by Exploiting Small Treewidth. In *ESA'18*, volume 112 of *LIPIcs*, pages 28:1–28:16. Dagstuhl Publishing, 2018.
20. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.
21. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
22. M. Jakl, R. Pichler, and S. Woltran. Answer-set programming with bounded treewidth. In *IJCAI'09*, volume 2, pages 816–822. The AAAI Press, 2009.
23. D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *SODA'11*, pages 760–776. SIAM, 2011.
24. F. Lonsing and U. Egly. Evaluating QBF solvers: Quantifier alternations matter. In *CP'18*, volume 11008 of *LNCS*, pages 276–294. Springer, 2018.
25. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
26. M. Morak, N. Musliu, R. Pichler, S. Rümmele, and S. Woltran. A new tree-decomposition based algorithm for answer set programming. In *ICTAI'11*, pages 916–918. IEEE Computer Society, 2011.
27. O. Pourret, P. Naim, and M. Bruce. *Bayesian Networks - A Practical Guide to Applications*. John Wiley & Sons, 2008.
28. N. Robertson and P. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.
29. N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
30. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In T. Joachims, editor, *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, volume 62, pages 98–105, 1998.
31. M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.