Structure-Driven Answer-Set Solving*

Markus Hecher

TU Wien, Austria; University of Potsdam, Germany
hecher@dbai.tuwien.ac.at

Abstract Parameterized algorithms are a way to solve hard problems efficiently, given that a specific parameter of the input is small. A well-studied parameter is treewidth, which roughly measures to which extent the structure of a graph resembles a tree. In our research, we want to exploit this parameter in the context of Answer Set Programming. In the literature, algorithms have been proposed that are linear in the program size assuming bounded treewidth. However, this approach works only in case of small treewidth. In consequence, we aim at new methods "between" traditional techniques and algorithms exploiting structural parameters.

Keywords: Tree Decompositions, Dynamic Programming, Fixed-Parameter Tractability, Answer-Set Programming, Parameterized Complexity

1 Introduction

Parameterized algorithms [14,8] have attracted considerable interest in recent years and allow to tackle hard problems by directly exploiting a small parameter of the input problem. One particular goal in this field is to find guarantees that the runtime is exponential exclusively in the parameter. A structure-based parameter that has been researched extensively is treewidth [15,5]. Generally speaking, 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 exploiting small treewidth typically takes a *tree decomposition* (TD), which is an arrangement of a graph into a tree, and evaluates the problem in parts by *dynamic programming* (DP) on the TD.

Answer Set Programming (ASP) [6] is a logic-based declarative modelling language and problem solving framework where solutions, so called answer sets, of a given logic program directly represent the solutions of the modelled problem. ASP has been successfully applied in several application domains and industrial needs, which, in particular, accelerates the search for alternative solving methods exploiting major aspects of relevant instances, as for example the structure of these instances. This generally brings up the question whether structure-based parameters as for instance treewidth aids in evaluating logic programs. Jakl et al. [12] give a DP algorithm that is linear in the input size of the program, double exponential in the treewidth of a certain graph representing the program

* Research was supported by the Austrian Science Fund (FWF), Grants Y698, P25607.

structure, and restricted to disjunctive rules. In our work [10], we presented an extension on how to evaluate extended logic programs based on the full ASP-Core-2 syntax. However, this paradigm seems to be only of use in case of small, bounded treewidth. In consequence, the idea is to study alternative evaluation techniques somewhere "between" traditional algorithms for ASP and algorithms exploiting structure-based parameters of logic programs. This research topic enables a broad range of specialization, ranging from the study of parameterized algorithms to CDCL-based algorithms exploiting structure of logic programs.

2 Preliminaries

Tree Decompositions (TDs) & Parameterized Algorithms. TDs form a tool to capture essential parts of the structure of a given graph instance. These decompositions are trees consisting of nodes, which contain (parts of) the given graph instances. The so-called "width" of such a TD is essential for the definition of the parameter "treewidth", which captures in a sense, how hard the given graph instance 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. TDs provide a way on how to tackle hard problems by evaluating a certain input problem instance (represented as a graph) in parts, thereby being sensitive to the treewidth of the instance. Of particular interest are so-called *fixed-parameter tractable* (FPT) algorithms with respect to a certain parameter k, which are capable of evaluating instances such that the runtime depends polynomially on the instance size and on $f(k)$ for some computable function f. If the parameter k is reasonably small, such an algorithm seems preferable compared to an approach, which requires exponential runtime in the worst-case. In consequence, such FPT algorithms with respect to treewidth perform well for certain instances and are typically based on *dynamic programming* (DP) on TDs, which computes parts of the problem obtained by iterating a TD of the problem instance, and combines them accordingly. There are also open-source systems like *D-FLAT* [3], which provide a general DP framework. Several techniques incorporated in our *DynASP* solver [10,9] for ASP, stem from D-FLAT.

Answer Set Programming (ASP). ASP is typically evaluated by two involved components, namely (i) the grounder and (ii) the solver. The grounder is responsible for producing a ground program during the *grounding*, a process which eliminates variables in an originally *non-ground* program containing variables. One can think of such non-ground programs as general sets of rule schemes, whereas ground programs are obtained by instantiating these general rule schemes into actual ASP rules. The main interest of this research proposal concerns part (ii), i.e., how to efficiently evaluate a set of ASP rules forming a *logic program*. However, note that in practice the performance of ASP techniques not only require efficient solving techniques, but also rely on smart grounding tools. Hence, especially the bridge between grounder and solver is of interest and might lead to further investigations concerning the structure of programs. As already mentioned, there also exist FPT algorithms for solving logic programs [10,12] by means of TDs.

3 Proposed Research

In this section, we first discuss the aim covered in this proposal and then give a detailed description of the results obtained so far and planned research tasks.

3.1 Research Aim

Gutin emphasized the necessity of establishing [1] parameterized algorithmics. As it turned out (not unexpectedly), implementing theoretical ideas in a straightforward way does not immediately yield practically successful systems. Several obstacles for properly handling real-world instances need to be taken into account, as we will discuss also in Section 3.2. We therefore identify the following aims:

- Incorporate concepts exploiting structure into existing solvers (and probably grounders) for ASP in order to make these tools more aware of the structure present in both data and rules, and to *improve solving performance*.
- Develop methods for building a novel, *competitive* ASP system consisting of (i) a solver that improves the DynASP system, and (ii) further methods exploiting and capturing structure of logic programs.
- Apply our findings in a broader, general context (for instance default logic).

3.2 Lessons Learned: Unexpected Results and Obstacles

The insights gained so far indicate that a general way to turn structure-driven answer set solving into a practical success is indeed challenging. First, we have observed that for ASP, structure can be fruitfully exploited. In other words, the treewidth of a ground program can be kept small when the instance has small treewidth as well (*structure in data*), but this depends on the actual (problem) encoding. Second, concerning the design of actual DP algorithms, we collected the following insights of the literature, which will also be the basis for my research.

- The shape of the TD on which DP is performed is crucial [1].
- A quite surprising result is that alternative space-efficient storage techniques via *Binary Decision Diagrams* (BDDs) can be extremely beneficial [7]. In fact, DP over TDs with widths up to 50 can be handled with such an approach.
- If it is enough to compute one solution, the classical, naive DP approach of computing all tables in their entirety before being able to print a solution can be substantially improved by a lazy-evaluation technique [4].

At the same time, it seems to be hard to compete with standard ASP solvers on the consistency problem (i.e., whether there is an answer set). In fact, we put much effort in the development of competitive systems, which excel mainly at counting answer sets, as we propose in [9]. We will thus aim (see Section 3.4) at ASP extensions that allow for reasoning over the sets of answer sets (similar to a result for monadic second-order logic [2]) and also develop new DP algorithms.

[1] G. Gutin: Should We Care about Huge Imbalance in Parameterized Algorithmics? *The Parameterized Compl. Newsletter* 11(2), `http://fpt.wdfiles.com/local--files/fpt-news:the-parameterized-complexity-newsletter/2015Dec.pdf`.

3.3 Obtained Results

The results obtained cover how structural properties can be exploited with focus on TDs and DP. First, we extended our DynASP solver [13] to the full ASP syntax as specified in the ASP-Core-2 standard. This includes interoperation with state-of-the-art grounders, and handling of all language constructs produced by such grounders such as weight constraints, optimization statements, choice rules and disjunctive rules. The implementation [10] works as follows: Ground ASP rules are represented as a graph and a TD is prepared, which is traversed in a bottom-up manner to evaluate the input program. We implemented several versions of such algorithms, based on the input program's graph representation, e.g., primal graphs, incidence graphs or a variant of incidence graphs where there is, in addition, a clique between weighted atoms (i.e., atoms in weight constraints or choice heads). Second, we improved the data structures in DynASP by using pointers to avoid duplicate data, as done in the D-FLAT^2 [3] system. Our technical proposal [9] also includes a thorough experimental evaluation.

Given logic programs of small treewidth, our new ASP solver proved to be very competitive in the setting of model counting (#SAT), a central problem in areas like machine learning, statistics, probabilistic reasoning and combinatorics. When counting answer sets (#ASP), our approach has a big advantage: It does not need to materialize the full answer sets in order to count them. This provided huge speed-ups against classical ASP systems like Clasp. However, our implementation was also able to beat SAT model counters like sharpSAT or Cachet, even though it was not specifically optimized for classical model counting. Benchmarks show that QBF model counting performance (comp. depQBF) is competitive as well.

3.4 Future Goals

Exploiting Treewidth in Existing Solvers. Our tasks concerning this goal cover extending CDCL-based ASP solvers like Clasp by exploiting structure of the input instances using suitable *heuristic parameterization*. Further, we aim at *improving branch-and-bound* algorithms used to solve an extended logic program capable of modelling cost optimization, by incorporating the instance structure.

A way to improve existing ASP technology is to push ASP solvers into the direction of DP on TDs by *modifying solver heuristics*. Heuristic modifications in ASP solvers have recently received increasing attention to solve problems in domains where dedicated solving strategies would have required rewriting problem encodings entirely. A main reason is due to the observation that solving problem instances efficiently by means of ASP solvers regularly relies on either a dedicated encoding, which integrates a certain heuristic or exploits certain structural properties of the problem, or controlling the solver heuristic explicitly. Our ultimate goal here is to develop methods for combining the two worlds of (i) state-of-the-art ASP solvers like Clasp and (ii) local approaches based on DP on TDs. We expect that integrating aspects of DP (either in terms of heuristics or by extending ASP solvers) can speed up CDCL-based ASP solvers on structured instances. A corresponding, very limited proof of concept is available.[2]

[2] `https://github.com/hmarkus/dynclasp`

ASP has been extended to also allow cost optimization, where solutions to such an extended logic program can be seen as "cost-minimal" answer sets. There exist algorithms to compute these solutions, which are based on branch-and-bound, and also other approaches. Branch-and-bound basically is a technique with the goal of searching for optimal answer sets in a systematic way by cutting off parts of the search space. However, there is no guarantee that the algorithm does not hit parts of an answer set, which might not be cost-optimal, several times during the search. One possible idea to overcome this limitation is to exploit the structure of these logic programs by *caching parts of answer sets* together with the corresponding cost found during the search in such a way, that the atoms of these parts reflect the content of some nodes of the TDs. In other words, whenever a new answer set is found, we store the answer set in parts such that each TD node remembers a table of parts of answer sets. After some potential solutions are proposed by the branch-and-bound algorithm, we obtain a tree of tables of answer set parts with the knowledge how these parts can be combined (by TD properties). In the end, the TD could enable us to combine these stored parts such that we might still obtain a valid answer set of lowest cost not found by the algorithm so far. Ultimately, the goal is to not derive answer sets, which contain *non-optimal* parts already contained in an answer set proposed before.

Building a Novel System. The goal is to develop a novel ASP system which makes direct use of the structure provided by the data and the program. Driven by recent success stories (see, e.g. [7,4]), the strategy is to *directly improve* DynASP's performance, and to *enhance* the system by certain features on the other hand.

The current implementation of DynASP provides (compared to Clasp) good results for solving ASP counting and enumeration problems in case of problem instances of sufficiently low treewidth, and is not yet competitive when solving the consistency problem. The reason is that DynASP does not apply preprocessing and constraint handling techniques and does not have a counterpart for conflict learning yet. Some of these disadvantages can be handled by integrating one or several *lightweight CDCL-solvers* for SAT or ASP into DynASP, which then solve the sub-problems induced by the TD nodes (during bottom-up solving), as implemented in our system D-FLAT [4]. Since the computation in each TD node works on an autonomic basis and only pushes new findings to the successor node(s), this leaves room for *improving the interplay* between these nodes. We believe that with the knowledge on how the (local) sub-programs evolve during the solving process (bottom-up traversal), there is a smart way to combine these lightweight solvers and improve their interplay by taking certain shortcuts. Especially *learning failing solution paths* could greatly improve the resulting solving performance. Learning might be promising in combination with lazy-evaluation [4], which proved valuable for optimization problems and aims at deriving answer sets without completely evaluating all the TD nodes.

Since DynASP turned out to be competitive for counting answer sets [10] (#ASP), our plan is to further improve by significantly reducing time spent on invalidating $\subset$-optimality of answer set candidates (see [9] for more details). One can easily identify instances with an exponential number (w.r.t. the treewidth)

of $\subset$-smaller model candidates and although this still suffices for the algorithm to be FPT, it is a performance bottleneck in practice. Recent approaches suggest *Binary Decision Diagrams* [7] with the benefits of compact representation.

Given the already explored correspondence between model counting and advanced reasoning tasks (e.g., Bayesian inference [16]), our future plan for DynASP includes a *query language for Bayesian reasoning* on top of it. This new language should be able to reduce queries to #ASP problems including, but not limited to "How plausible is it to assume that a given atom belongs to an answer set?", "If an atom is believed, what is the probability of another atom belonging to an answer set?" or "Is it safe to assume that one atom is more reasonable than an other atom?". Considerations also include solving Problog [11] programs.

References

1. M. Abseher, F. Dusberger, N. Musliu, and S. Woltran. Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning. In *IJCAI*, pages 275–282. AAAI, 2015.
2. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *JAlg*, 12(2):308–340, 1991.
3. B. Bliem, G. Charwat, M. Hecher, and S. Woltran. D-FLATˆ2: Subset minimization in dynamic programming on tree decompositions made easy. *FI*, 147(1):27–61, 2016.
4. B. Bliem, B. Kaufmann, T. Schaub, and S. Woltran. ASP for Anytime Dynamic Programming on Tree Decompositions. In *IJCAI*, pages 979–986. AAAI, 2016.
5. H. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *TCJ*, 51(3):255–269, 2008.
6. G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
7. G. Charwat and S. Woltran. Dynamic programming-based QBF solving. In *QBF*, pages 27–40, 2016.
8. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
9. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer Set Solving using Tree Decompositions and Dynamic Programming - The DynASP2 System -. Technical Report DBAI-TR-2016-101, TU Wien, 2016.
10. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer set solving with bounded treewidth revisited. In *LPNMR*, 2017. To appear.
11. D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *TPLP*, 15(3):358–401, 2015.
12. M. Jakl, R. Pichler, and S. Woltran. Answer-set programming with bounded treewidth. In *IJCAI*. AAAI, 2009.
13. M. Morak, N. Musliu, R. Pichler, S. Rümmele, and S. Woltran. A New Tree-Decomposition Based Algorithm for Answer Set Programming. In *ICTAI*, pages 916–918, 2011.
14. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. OUP, 2006.
15. N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *JAlg*, 7(3):309–322, 1986.
16. T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian Inference by Weighted Model Counting. In *AAAI*, pages 475–482. AAAI, 2005.