Treewidth in Non-Ground Answer Set Solving and Alliance Problems in Graphs*

Bernhard Bliem

TU Wien, Vienna, Austria
bliem@dbai.tuwien.ac.at

Abstract

To solve hard problems efficiently via answer set programming (ASP), a promising approach is to take advantage of the fact that real-world instances of many hard problems exhibit small treewidth. Algorithms that exploit this have already been proposed – however, they suffer from an enormous overhead. In the thesis, we present improvements in the algorithmic methodology for leveraging bounded treewidth that are especially targeted toward problems involving subset minimization. This can be useful for many problems at the second level of the polynomial hierarchy like solving disjunctive ground ASP. Moreover, we define classes of non-ground ASP programs such that grounding such a program together with input facts does not lead to an excessive increase in treewidth of the resulting ground program when compared to the treewidth of the input. This allows ASP users to take advantage of the fact that state-of-the-art ASP solvers perform better on ground programs of small treewidth. Finally, we resolve several open questions on the complexity of alliance problems in graphs. In particular, we settle the long-standing open questions of the complexity of the Secure Set problem and whether the Defensive Alliance problem is fixed-parameter tractable when parameterized by treewidth.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases answer set programming, treewidth, secure set, defensive alliance, parameterized complexity

Digital Object Identifier 10.4230/OASIcs.ICLP.2017.12

1 Introduction

The problem solving paradigm Answer Set Programming (ASP) [12, 30, 46, 45] has become quite popular for tackling computationally hard problems. It offers its users a very convenient declarative language that allows for succinct specifications, and there are highly efficient systems available [32, 31, 29, 2, 3, 44, 4, 51, 23].

Although ASP systems have made huge advances in performance, they still struggle with several tough problems. This is not always just an issue of computational complexity in the classical sense. Interestingly, ASP systems may perform quite well in practice on one problem whereas the performance on another problem of the same complexity can be significantly worse. Often classical complexity theory is thus only of limited help to explain ASP solving performance in practice. In such cases, it may be insightful to consider the *parameterized* complexity of the problems [20, 28, 17, 48]. This theoretical framework investigates the complexity of a problem not only in terms of the input size, but also of other parameters.

In this work, we are particularly interested in the effect of the structural parameter *treewidth* [50] on the performance of ASP solvers. Intuitively, the smaller the treewidth

* This work was supported by the Austrian Science Fund (FWF) projects P25607 and Y698.

Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017).
Editors: Ricardo Rocha, Tran Cao Son, Christopher Mears, and Neda Saeedloei; Article No. 12; pp. 12:1–12:12

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of a graph, the closer the graph resembles a tree. It is well-known that many graph problems become easy if we restrict the input to trees and it has turned out that for many important problems this even holds for the more general class of instances of bounded treewidth [5]. Luckily, it has been observed that real-world instances usually exhibit small treewidth [10, 52, 41]. Moreover, treewidth is not only relevant for graph problems. It can also be applied to instances of all kinds of problems by choosing a suitable representation of the instance as a graph. For instance, treewidth has also been considered for constraint satisfaction problems [19], where it is known under the name of "induced width" and is crucial for the performance of a technique called bucket elimination [18].

There have already been some investigations concerning treewidth and ground ASP (i.e., ASP programs without variables, also known as propositional programs) [34, 49, 26]. An important result is the algorithm from [39] for deciding whether a ground ASP program has a solution in linear time on instances of bounded treewidth. This algorithm employs a technique called *dynamic programming on tree decompositions*, which is very common for algorithms that exploit small treewidth. The algorithm from [39] has also been implemented and proposed as an alternative solver for ground ASP [47]. For certain problems, this dynamic-programming-based solver was able to outperform state-of-the-art ASP solvers if the instances had a very small treewidth and the sizes of the instances were very large.

Although the encouraging results from [47] confirmed that small treewidth can be successfully exploited for ASP solving in experimental settings, the restrictions on problems and instances that make this approach perform well were still too severe for most practical applications. The main obstacles that prevented this approach from being useful for a broad range of applications were the facts that, on the one hand, the naive dynamic programming approach involves an enormous overhead (especially in terms of memory) and, on the other hand, state-of-the-art ASP solvers often perform so well that the theoretical superiority of the dynamic programming algorithm only pays off for instances of tremendous size. In fact, experiments in [7] indicated that state-of-the-art ASP solvers are "sensitive" to the treewidth of their input in the sense that smaller treewidth strongly correlates with higher solving performance.

These issues hint at interesting research challenges. In particular, two approaches seem promising for successfully exploiting small treewidth for ASP solving in practice:

- The first research challenge is to improve the dynamic-programming-based methodology in order to avoid some of its overhead and redundant computations.

  For solving ground ASP, these issues are especially severe compared to other problems because the corresponding computational problems are even harder than NP under standard complexity-theoretic assumptions. (In fact, deciding whether a ground ASP program with disjunctions has an answer set is at the second level of the polynomial hierarchy.) This high complexity of ground ASP is mirrored in the dynamic programming algorithm [39], which uses brute force to first of all find all models of all parts of the decomposed program, and it subsequently uses brute force again *for each such partial model* to find all potential counterexamples that may cause the candidate to be discarded.

  This pattern also frequently occurs in dynamic programming algorithms for other problems that search for solutions satisfying some form of subset minimality. Besides ground ASP, this is the case, for instance, for the problem of finding subset-minimal models of a propositional formula. In general, problems involving subset-minimization are quite common in AI, and dynamic programming algorithms have been proposed for, e.g., circumscription, abduction or abstract argumentation (see [38, 35, 21]). Such algorithms typically store a great number of redundant objects because the subsets that may invalidate

a solution candidate are themselves solution candidates. Moreover, the specifications of such algorithms themselves contain redundancies because the potential counterexamples are usually manipulated in almost the same way as the solution candidates.

- The second research challenge is to solve ASP by not doing dynamic programming at all but instead exploiting small treewidth *implicitly* by relying on the assumption that state-of-the-art solvers perform better when given ground programs of small treewidth (as indicated by the experiments in [7]).
  Since problems are usually encoded in non-ground ASP, here the research objective is to investigate which non-ground encoding techniques significantly blow up the treewidth of the grounding when compared to the treewidth of the input facts.

In addition to leveraging treewidth for ASP solving, we are interested in several variants of a graph problem called Secure Set [13]. It belongs to the class of so-called *alliance problems* [42, 24, 53], which are problems that ask for groups of vertices that help each other out in a certain way. Practical applications of alliance problems include finding groups of websites that form communities [27] or distributing resources in a computer network in such a way that simultaneous requests can be satisfied [36]. Intuitively, a set S of vertices in a graph is secure if every subset of S has as least as many neighbors in S as neighbors not in S. The Secure Set problem asks whether a given graph contains a secure set at most of a certain size.

The reason why we are concerned with Secure Set is that this problem has quite interesting properties, especially for ASP researchers: Attempts of encoding this problem in ASP have resulted in very involved specifications indicating that Secure Set may require the full expressive power of ASP [1]. However, it is unfortunately unclear whether this is really necessary because its complexity has still remained unresolved although the problem has been introduced already in 2007 [13].

One of the variants of Secure Set that we consider in the proposed thesis is the Defensive Alliance problem [42, 43]. This problem has received quite some attention in the literature [24]. It is known to be NP-complete, but its complexity when parameterized by treewidth has remained open.

2 Background

We assume some familiarity with ASP; introductions can be found in [12, 30, 46, 45]. In the thesis, we study both ground ASP programs, i.e., programs without variables, but also programs utilizing the full ASP language as described in the ASP-Core-2 specification [14]. In particular, we include weak constraints and aggregates.

We only outline the syntax of a simplified version of non-ground ASP. For details and semantics, we refer to the standard [14]. An ASP program consists of *rules* of the following form:

$$h_1 \mid \ldots \mid h_k :- p_1, \ldots, p_\ell, \text{not } n_1, \ldots, \text{not } n_m.$$

The *head* of a rule r is the set denoted by $H(r) = \{h_1, \ldots, h_k\}$, the *positive body* of r is the set $B^+(r) = \{p_1, \ldots, p_\ell\}$, and the *negative body* of r is the set $B^-(r) = \{n_1, \ldots, n_m\}$. All elements of these sets are called *atoms*. An atom is either a *predicate atom* or an *aggregate atom*. Predicate atoms have the form $p(t_1, \ldots, t_j)$, where p is a *predicate* and $t_1, \ldots, t_j$ are *terms*, that is, constants or variables. A predicate is called *extensional* in a program Π if it only occurs in rule bodies of Π. An atom is called *extensional* in Π if it is a predicate atom over an extensional predicate. We omit a definition of aggregate atoms but only mention

that they allow us to, e.g., compute a sum of integers or count the cardinality of a set. For details, we refer to [14]. In addition to rules as defined above, we allow for weak constraints, which are special kinds of rules and have the following form:

$$:\sim p_1, \ldots, p_\ell, \text{not } n_1, \ldots, \text{not } n_m.\ [w, t_1, \ldots, t_q]$$

The intuition is that if an answer set violates a ground instantiation of such a weak constraint, then this incurs a penalty of w to the cost of the solution. (Without the terms $t_1, \ldots, t_k$, each weight w would in fact only be counted once in the cost of a solution, hence $t_1, \ldots, t_k$ can be specified for counting the same weight multiple times.) Again, we omit details and refer to [14] instead.

To solve a non-ground ASP program, ASP systems usually first invoke a *grounder* that transforms a program into a set of ground rules. The answer sets of the original program are the *stable models* (as defined in [33]) of the resulting ground program.

A "naive" grounder blindly instantiates variables by all possible ground terms. Grounders in practice, on the other hand, employ sophisticated techniques in order to keep the resulting ground program as small as possible. As these techniques differ between systems, we define a simplified notion of grounding that is easier to study. For a meaningful investigation of the relationship between the treewidth of the input and the treewidth of the grounding, we need to assume that the grounder still performs some basic simplifications. These simplifications are so basic that they can be assumed to be implemented by all reasonable grounders. The intuition is that a rule from the "naive" grounding is omitted in our grounding whenever its positive body contains an atom that cannot possibly be derived.

▶ **Definition 1.** Let Π be a non-ground ASP program, let Π^+ denote the positive program obtained from Π by removing all negated atoms and replacing disjunctions with conjunctions (i.e., splitting disjunctive into normal rules), and let M^+ be the unique minimal model of Π^+. The *grounding* of Π, denoted by $\mathrm{gr}(\Pi)$, is such that, for every substitution s from variables to constants, $s(r) \in \mathrm{gr}(\Pi)$ iff $s(B^+(r)) \subseteq M^+$.

In our work, we are interested in the treewidth of ground ASP programs. For this, we represent programs as graphs as follows.

▶ **Definition 2.** The *primal graph* of a ground ASP program Π is an undirected graph whose vertices are the atoms in Π and there is an edge between two atoms if they appear together in a rule in Π. The *treewidth* of a ground ASP program Π is the treewidth of its primal graph.

Deciding whether a disjunctive ground ASP program has a stable model is Σ_2^{P}-complete in general [22], and it can be done in linear time for ground programs of bounded treewidth [34]. Treewidth is an important parameter studied in the context of *parameterized complexity theory*. Here, decision problems consist not only of an instance and a yes-no question, but additionally of a parameter of the instance. For introductions, we refer to [20, 28, 17, 48]. The central notion of tractability is called *fixed-parameter tractability*.

▶ **Definition 3.** A problem is *fixed-parameter tractable* (FPT) w.r.t. a parameter k of the instances if it admits an algorithm that runs in time $\mathcal{O}(f(k) \cdot n^c)$, where f is an arbitrary computable function that only depends on k, n is the input size and c is an arbitrary constant. We call such an algorithm an *FPT algorithm*.

Note that the factor $f(k)$ in this running time may be exponential in the parameter k, but if k is bounded by a constant, then the algorithm runs in polynomial time. Importantly, the degree c of the polynomial must be a constant and may not depend on the parameter, otherwise the algorithm is not considered FPT.

Dynamic programming on tree decompositions is perhaps the most common technique for obtaining FPT algorithms when the parameter is treewidth. It is employed in the algorithm for solving ground ASP in [39], for instance. The basic idea is the following: Given a graph G, a tree decomposition of G is a tree whose nodes correspond to subgraphs of G according to certain conditions. If the treewidth of a graph is bounded by a constant, then we can find (in linear time) a tree decomposition whose nodes correspond to subgraphs of constant size [11]. We can then solve many problems by first applying brute force at each subgraph in order to solve a subproblem corresponding to this subgraph and then trying to combine the obtained partial solutions. Due to the bound on the treewidth, we can afford this brute force approach because each of the considered subgraphs has bounded size. Formal definitions and examples of this technique can be found in, e.g., [48].

We now define secure sets in graphs [13]. For this, we use the notation $N_G[S]$ to denote the closed neighborhood of a subset S of the vertices of a graph G; that is, $N_G[S]$ contains the vertices in S itself and the vertices that are adjacent to an element of S.

▶ **Definition 4.** Let G be a graph and S be a subset of its vertices. We call S *secure in* G if $|N_G[X] \cap S| \geqslant |N_G[X] \setminus S|$ holds for every $X \subseteq S$.

Intuitively, we can regard a neighbor of X as a "good" neighbor if it is also in S, and as a "bad" neighbor otherwise. Now X is a counterexample to S being secure if X has more bad neighbors than good ones.

3 Contributions

Our contributions can be arranged in three groups: First, we present improvements in the dynamic programming methodology; second, we define non-ground ASP classes that can be shown to preserve bounded treewidth of the input in grounding; third, we provide complexity results and algorithms for alliance problems in graphs.

3.1 Improvements in the Dynamic Programming Methodology

We present an improved dynamic programming methodology for problems that involve subset minimization. Specifically, for any problem P whose solutions are exactly the subset-minimal solutions of some base problem B, we formalize how a dynamic programming algorithm for B can automatically be transformed into a dynamic programming algorithm for P. We prove that the resulting algorithm runs in linear time on instances of bounded treewidth if the base algorithm does. Moreover, we prove that the resulting algorithm is correct if the base algorithm is correct and, intuitively, it only computes partial solutions that do not "revoke decisions" made by associated partial solutions further down in the tree decomposition. The resulting algorithm has two advantages compared to solving P directly in a naive way: first, it is usually easier to specify because we only need to design an algorithm for the base problem and need not care about subset minimization; second, it is potentially more efficient because it stores fewer redundant items.

Indeed, this methodology has been empirically shown to lead to significant performance benefits for several problems [6]. An improved version of the classical dynamic programming algorithm for ground ASP has been implemented using these ideas [25] and proved to be significantly more efficient than the algorithm from [39]. Our result formalizes the common scheme that underlies these algorithms. We thus provide a formal framework that makes it possible to transfer the mentioned optimizations easily to other problems. Thereby we make the impressive performance benefits that have been reported in [6, 25] accessible to

Listing 1 A guarded ASP encoding for checking whether a given set S (declared using predicate `s`) is secure in a given graph (declared using predicates `v` and `e`). The guards of rules are underlined.

```
% Guess a subset X of S.
x(S) | nx(S) :- s(S).
% Neighbors of X are "good" if they are in S, otherwise they are "bad".
neighbor(V) :- x(X), e(X,V).
neighbor(X) :- x(X), v(X).
good(V)     :- neighbor(V), s(V).
bad(V)      :- neighbor(V), v(V), not s(V).
% If X has more bad neighbors than good ones, S is not secure.
% We use the following weak constraints to determine this by summing up.
:~ v(V), good(V). [1,V]  % Add 1 for each good neighbor.
:~ v(V), bad(V). [-1,V]  % Subtract 1 for each bad neighbor.
```

algorithm designers working on related problems. This is primarily useful for problems on the second level of the polynomial hierarchy as subset minimization is a recurring theme in many such problems.

3.2 Non-Ground ASP Classes that Preserve Bounded Treewidth

We define non-ground ASP classes for which grounding, according to Definition 1, preserves bounded treewidth of the input. By restricting the syntax of non-ground ASP, we define two classes of programs called *guarded* and *connection-guarded* programs [7]. Guarded programs guarantee that the treewidth of any fixed program after grounding stays small whenever the treewidth of the input facts is small. We formally prove this property and show that, despite their restrictions, guarded programs can still express problems that are complete for the second level of the polynomial hierarchy.

Connection-guarded programs are even more expressive than guarded programs. We show that the treewidth of any fixed connection-guarded program after grounding is small whenever the treewidth *and the maximum degree* of (a graph representation of) the input facts is small.

These results bring us closer to the goal of implicitly taking advantage of the apparent sensitivity to treewidth exhibited by modern ASP solvers because they give us insight into what happens to the treewidth of the input during grounding. Thus, by writing a program in guarded ASP, we can be sure that the grounder does not destroy the property of bounded treewidth. In the case of connection-guarded ASP, the same holds for the combination of treewidth and maximum degree.

▶ **Example 5.** The ASP encoding in Listing 1 can be used for deciding whether a given set S of vertices is secure in a given graph G. It guesses a subset X of S and uses weak constraints in such a way that the cost of each answer set is exactly $|N_G[X] \cap S| - |N_G[X] \setminus S|$. If there is a subset of S that has more "bad" neighbors than "good" ones, then the program has an answer set with negative cost. The solutions of a program with weak constraints are those answer sets that minimize the cost incurred by violated weak constraints. We can thus decide whether S is secure by checking if this minimum value is negative.

The program in Listing 1 is guarded, which means that, for each rule r, all variables of r occur together in a single extensional atom of $B^+(r)$ that we call the *guard* of r. Note that, alternatively, it is also possible to check whether a set of vertices is secure without using weak constraints. For instance, we can replace the weak constraints by the "hard" constraint

`:- #sum{ 1,G : good(G); -1,B : bad(B) } >= 0.` The resulting program has an answer set if and only if S is not secure. However, the new constraint is not guarded. This means that the original program in Listing 1 generally leads to groundings of much lower treewidth and can thus be expected to perform better. Indeed, the ground instantiation of the new hard constraint would contain a linear number of atoms. Thus, the primal graph of the grounding would contain a clique of linear size and thus have linear treewidth even if the treewidth of input graphs is bounded by a constant.

In the thesis, we also present a complexity analysis of computational problems corresponding to these classes when the parameter is the treewidth of the input, the maximum degree of the input, or the combination of both. The results of this analysis show that, for any fixed guarded ASP program, answer set solving is FPT when parameterized by the treewidth of the input; moreover, for any fixed connection-guarded ASP program, answer set solving is FPT when parameterized by the combination of treewidth and maximum degree. This is not obvious because our ASP classes support weak constraints and aggregates, which are not accounted for in the FPT algorithms [39, 25] for ground ASP. Furthermore, we prove hardness results showing that for connection-guarded ASP programs *both* the treewidth and the maximum degree must be bounded for obtaining fixed-parameter tractability. We do this by presenting a connection-guarded ASP encoding of a problem that is NP-hard even if the treewidth of the instances is fixed and by presenting a guarded encoding of a problem that is Σ_2^P-hard even if the degree of the instances is fixed.

As a side-product of these investigations, we obtain *metatheorems* for proving FPT results. That is, our results on guarded ASP allow us to prove that a problem is FPT when parameterized by treewidth by simply expressing the problem in guarded ASP. We compare this metatheorem to the common approach of proving fixed-parameter tractability by expressing a problem in monadic second-order logic and invoking the well-known theorem by Courcelle [15, 16]. Similarly, we can prove that a problem is FPT when parameterized by the combination of treewidth and maximum degree by expressing the problem in connection-guarded ASP. This result is appealing because we are not aware of any metatheorems that allow us to obtain FPT results for the combination of treewidth and degree as the parameter.

3.3 Alliance Problems in Graphs

We perform a complexity analysis of alliance problems in graphs, both in the classical setting and when parameterized by treewidth. First, we settle the complexity of the Secure Set problem by proving that the problem, along with several variants, is Σ_2^P-complete (that is, at the second level of the polynomial hierarchy).

Next we turn to the complexity of Secure Set and Defensive Alliance when the problems are parameterized by treewidth. We illustrate the use of our ASP classes as FPT classification tools by presenting simple encodings for alliance problems in graphs. By encoding the NP-complete Defensive Alliance problem in connection-guarded ASP, we easily obtain the already known result that the problem is FPT when parameterized by the combination of treewidth and maximum degree. More importantly, we obtain the new result that the co-NP-complete problem of deciding whether a given set is secure in a graph is FPT for the parameter treewidth by encoding the problem in guarded ASP.

We also give several negative results. We prove that both Defensive Alliance and Secure Set, as well as several problem variants, are not FPT when parameterized by treewidth (under commonly held complexity-theoretic assumptions). These questions have been open since the problems have been introduced in 2002 and 2007, respectively. They have explicitly been stated as open problems in [40] (for Defensive Alliance) and in [37] (for Secure Set).

Despite the parameterized hardness of Secure Set, we can give at least a slightly positive result: We show that the Secure Set problem can still be solved in polynomial time for instances of bounded treewidth although the degree of the polynomial depends on the treewidth.

4 Current Status

The largest part of the research for the proposed thesis has already been done and is in the process of being integrated and written down. Most of the results have been published in conference proceedings and journals:

- The work on improving the dynamic programming methodology for problems involving subset minimization has been published in [6].
- The class of connection-guarded ASP programs, which preserves bounded treewidth of the input in grounding whenever the maximum degree is also bounded has been published in [7]. That paper neither contained the thorough complexity analysis performed in the proposed thesis nor the work on the class of guarded programs, which may be attractive because this class does not require the degree of input graphs to be bounded.
- The Σ_2^P-completeness result of the Secure Set problem has been published in [9]. An extended version [8], which is currently under review for a journal, additionally contains the parameterized complexity results. The proposed thesis extends this by results on the parameterized complexity of the Defensive Alliance problem as well.

5 Open Issues

The proposed thesis opens up several possibilities for future research:

- The class of connection-guarded ASP programs may be of interest for algorithmic purposes because it allows us to classify a problem as FPT when parameterized by treewidth plus degree. A common technique for classifying problems parameterized by treewidth as FPT is expressing them in monadic second-order logic (MSO). Our result may lead to an extension of MSO that can be used for classifying problems as FPT when the parameter is treewidth + degree.
- From a more practical perspective, it is promising to look closely into what ASP solvers and in particular their heuristics are doing when they are presented with a grounding of small treewidth. This could provide us insight into why state-of-the-art ASP solvers perform better on instances of small treewidth even though they do not "consciously" exploit this fact. With the gained understanding, we may be able to improve their performance by explicitly taking information from a tree decomposition into account during solving. This could perhaps lead to a hybrid ASP solving approach that uses classical conflict-driven clause learning in combination with techniques based on tree decompositions.
- We showed that Secure Set is not FPT when parameterized by treewidth (unless the class W[1] is equal to FPT). It would be interesting to study which additional restrictions beside bounded treewidth need to be imposed on Secure Set instances to achieve fixed-parameter tractability. In particular, we do not know whether it becomes FPT when additionally the degree is bounded.
- Regarding our polynomial-time algorithm for Secure Set on instances of bounded treewidth, it would be interesting to study if this result can be extended to instances of bounded clique-width, a parameter related to treewidth.

- Moreover, we have not considered a problem that is closely related to Defensive Alliance, namely Offensive Alliance. Possibly some of our techniques can also be applied to obtain complexity results for this problem.
- Defensive Alliance differs from Secure Set in the size of the subsets of solution candidates that need to be checked. For future work it would be interesting to study the complexity of a problem that generalizes both of them, where the size of the subsets is a parameter.
- Finally, for the parameterized hardness results that we obtained we do not have corresponding membership results. This is an obvious task for future work.

Acknowledgements. The proposed thesis was supervised by Stefan Woltran and is based on publications with significant contributions by Günther Charwat, Markus Hecher, Marius Moldovan and Michael Morak.

References

1 Michael Abseher, Bernhard Bliem, Günther Charwat, Frederico Dusberger, and Stefan Woltran. Computing secure sets in graphs using answer set programming. *J. Logic Comput.*, 2015. Accepted for publication. doi:10.1093/logcom/exv060.

2 Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Pedro Cabalar and Tran Cao Son, editors, *Proceedings of LPNMR 2013*, volume 8148 of *LNCS*, pages 54–66. Springer, 2013. doi:10.1007/978-3-642-40564-8_6.

3 Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In Francesco Calimeri, Giovambattista Ianni, and Mirosław Truszczyński, editors, *Proceedings of LPNMR 2015*, volume 9345 of *LNCS*, pages 40–54. Springer, 2015. doi:10.1007/978-3-319-23264-5_5.

4 Mario Alviano, Wolfgang Faber, Nicola Leone, Simona Perri, Gerald Pfeifer, and Giorgio Terracina. The disjunctive datalog system DLV. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers, editors, *Revised Selected Papers of Datalog 2010*, volume 6702 of *LNCS*, pages 282–301. Springer, 2011. doi:10.1007/978-3-642-24206-9_17.

5 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

6 Bernhard Bliem, Günther Charwat, Markus Hecher, and Stefan Woltran. D-FLATˆ2: Subset minimization in dynamic programming on tree decompositions made easy. *Fund. Inform.*, 147(1):27–61, 2016. doi:10.3233/FI-2016-1397.

7 Bernhard Bliem, Marius Moldovan, Michael Morak, and Stefan Woltran. The impact of treewidth on ASP grounding and solving. In Carles Sierra and Fahiem Bacchus, editors, *Proceedings of IJCAI 2017*. The AAAI Press, 2017. Accepted for publication.

8 Bernhard Bliem and Stefan Woltran. Complexity of secure sets. *CoRR*, abs/1411.6549, 2014. Updated to version 3 on July 11, 2017. URL: http://arxiv.org/abs/1411.6549.

9 Bernhard Bliem and Stefan Woltran. Complexity of secure sets. In Ernst W. Mayr, editor, *Revised Papers of WG 2015*, volume 9224 of *LNCS*, pages 64–77. Springer, 2016. doi:10.1007/978-3-662-53174-7_5.

10 Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, 11(1-2):1–21, 1993.

11 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

12 Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011. doi:10.1145/2043174.2043195.

13 Robert C. Brigham, Ronald D. Dutton, and Stephen T. Hedetniemi. Security in graphs. *Discrete Appl. Math.*, 155(13):1708–1714, 2007. doi:10.1016/j.dam.2007.03.009.

14 Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. https://www.mat.unical.it/aspcomp2013/ASPStandardization, 2015. Version: 2.03c.

15 Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

16 Bruno Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *RAIRO Theor. Inform. Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571.

17 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, Cham, Switzerland, 2015. doi:10.1007/978-3-319-21275-3.

18 Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999. doi:10.1016/S0004-3702(99)00059-4.

19 Rina Dechter. *Constraint Processing*. Elsevier Morgan Kaufmann, Amsterdam, The Netherlands, 2003. doi:10.1016/b978-1-55860-890-0.x5000-2.

20 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, New York, NY, USA, 1999. doi:10.1007/978-1-4612-0515-9.

21 Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence*, 186:1–37, 2012. doi:10.1016/j.artint.2012.03.005.

22 Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995. doi:10.1007/BF01536399.

23 Islam Elkabani, Enrico Pontelli, and Tran Cao Son. Smodels^{A} - A system for computing answer sets of logic programs with aggregates. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Proceedings of LPNMR 2005*, volume 3662 of *LNCS*, pages 427–431. Springer, 2005. doi:10.1007/11546207_40.

24 Henning Fernau and Juan A. Rodríguez-Velázquez. A survey on alliances and related parameters in graphs. *Electron. J. Graph Theory Appl. (EJGTA)*, 2(1):70–86, 2014. doi:10.5614/ejgta.2014.2.1.7.

25 Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Proceedings of LPNMR 2017*, volume 10377 of *LNCS*, pages 132–145. Springer, 2017. doi:10.1007/978-3-319-61660-5_13.

26 Johannes Klaus Fichte and Stefan Szeider. Backdoors to tractable answer set programming. *Artificial Intelligence*, 220:64–103, 2015. doi:10.1016/j.artint.2014.12.001.

27 Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3):66–71, 2002. doi:10.1109/2.989932.

28 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006. doi:10.1007/3-540-29953-X.

29 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub. Progress in clasp series 3. In Francesco Calimeri, Giovambattista Ianni, and

Mirosław Truszczyński, editors, *Proceedings of LPNMR 2015*, volume 9345 of *LNCS*, pages 368–383. Springer, 2015. doi:10.1007/978-3-319-23264-5_31.

30 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, Williston, VT, USA, 2012. doi:10.2200/S00457ED1V01Y201211AIM019.

31 Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp*: A conflict-driven answer set solver. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Proceedings of LPNMR 2007*, volume 4483 of *LNCS*, pages 260–265. Springer, 2007. doi:10.1007/978-3-540-72200-7_23.

32 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012. doi:10.1016/j.artint.2012.04.001.

33 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of JICSLP 1988*, volume 2, pages 1070–1080. The MIT Press, 1988.

34 Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artificial Intelligence*, 174(1):105–132, 2010. doi:10.1016/j.artint.2009.10.003.

35 Georg Gottlob, Reinhard Pichler, and Fang Wei. Tractable database design and datalog abduction through bounded treewidth. *Inf. Syst.*, 35(3):278–298, 2010. doi:10.1016/j.is.2009.09.003.

36 Teresa W. Haynes, Stephen T. Hedetniemi, and Michael A. Henning. Global defensive alliances in graphs. *Electron. J. Combin.*, 10, 2003. URL: http://www.combinatorics.org/Volume_10/Abstracts/v10i1r47.html.

37 Yiu Yu Ho and Ronald D. Dutton. Rooted secure sets of trees. *AKCE Int. J. Graphs Comb.*, 6(3):373–392, 2009.

38 Michael Jakl, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Fast counting with bounded treewidth. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Proceedings of LPAR 2008*, volume 5330 of *LNCS*, pages 436–450. Springer, 2008. doi:10.1007/978-3-540-89439-1_31.

39 Michael Jakl, Reinhard Pichler, and Stefan Woltran. Answer-set programming with bounded treewidth. In Craig Boutilier, editor, *Proceedings of IJCAI 2009*, pages 816–822. The AAAI Press, 2009.

40 Masashi Kiyomi and Yota Otachi. Alliances in graphs of bounded clique-width. *Discrete Appl. Math.*, 223:91–97, 2017. doi:10.1016/j.dam.2017.02.004.

41 András Kornai and Zsolt Tuza. Narrowness, pathwidth, and their application in natural language processing. *Discrete Appl. Math.*, 36(1):87–92, 1992. doi:10.1016/0166-218X(92)90208-R.

42 Petter Kristiansen, Sandra M. Hedetniemi, and Stephen T. Hedetniemi. Introduction to alliances in graphs. In Ilyas Cicekli, Nihan Kesim Cicekli, and Erol Gelenbe, editors, *Proceedings of ISCIS 2002*, pages 308–312. CRC Press, 2002.

43 Petter Kristiansen, Sandra M. Hedetniemi, and Stephen T. Hedetniemi. Alliances in graphs. *J. Combin. Math. Combin. Comput.*, 48:157–178, 2004.

44 Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006. doi:10.1145/1149114.1149117.

45 Vladimir Lifschitz. What is answer set programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of AAAI 2008)*, pages 1594–1597. The AAAI Press, 2008.

46 Victor W. Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof Apt, Victor W. Marek, Mirosław Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pages 375–398. Springer, New York, NY, USA, 2011. doi:10.1007/978-3-642-60085-2.

47 Michael Morak, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. A dynamic-programming based ASP-solver. In Tomi Janhunen and Ilkka Niemelä, editors, *Proceedings of JELIA 2010*, volume 6341 of *LNCS*, pages 369–372. Springer, 2010. doi:10.1007/978-3-642-15675-5_34.

48 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, United Kingdom, 2006. doi:10.1093/acprof:oso/9780198566076.001.0001.

49 Reinhard Pichler, Stefan Rümmele, Stefan Szeider, and Stefan Woltran. Tractable answer-set programming with weight constraints: Bounded treewidth is not enough. *Theory Pract. Log. Program.*, 14(2):141–164, 2014. doi:10.1017/S1471068412000099.

50 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Combin. Theory Ser. B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

51 Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002. doi:10.1016/S0004-3702(02)00187-X.

52 Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inform. and Comput.*, 142(2):159–181, 1998. doi:10.1006/inco.1997.2697.

53 Ismael González Yero and Juan A. Rodríguez-Velázquez. Defensive alliances in graphs: A survey. *CoRR*, abs/1308.2096, 2013. URL: http://arxiv.org/abs/1308.2096.