Answer Set Counting and its Applications to Network Reliability and System Biology*

Mohimenul Kabir[1,*]

[1]National University Singapore, Singapore

Abstract

We focus on Answer Set Programming (ASP), with an emphasis on the problem of counting answer sets. Our work explores both exact and approximate methodologies. We developed sharpASP, an exact counter that employs a compact encoding of propositional formulas, significantly improving efficiency over existing ASP counters. Empirical evaluations show that sharpASP outperforms state-of-the-art tools on a range of benchmarks. Additionally, we introduce ApproxASP, a hashing-based approximate counter that integrates Gauss-Jordan elimination into the ASP solver clingo. We demonstrate the practical utility of ApproxASP through its application to network reliability estimation, systems biology, minimal model reasoning, and MUS counting. To address these challenges, we propose efficient reductions to the answer set counting problem. Our experimental evaluation shows that the proposed techniques outperform traditional estimators, explicit enumerators, and BDD- or #SAT-based methods in both accuracy and efficiency.

Keywords

Answer Set Counting, Exact and Approximate Answer Set Counting, Network Reliability, System Reliability,

1. Introduction

Answer Set Programming (ASP) [1] has established itself as a powerful framework for knowledge representation and reasoning, particularly well-suited for expressing complex combinatorial problems in a declarative manner [2]. Leveraging progress in SAT solving, the ASP community has developed highly optimized systems for solving the satisfiability problem—i.e., computing models (or answer sets) of a given program. Beyond satisfiability, recent research has increasingly focused on counting answer sets, a problem known as *answer set counting*. This shift is driven by applications that require quantifying uncertainty or reliability, such as probabilistic inference, network analysis, and planning [3, 4, 5, 6, 7, 8, 9]. In this work, we focus on advancing both exact and approximate techniques for answer set counting and demonstrate their applicability to real-world problems (e.g., network reliability and systems biology). Compared to my last year submission to ICLP LPNMR 2024 Doctoral Consortium [10], this submission makes contribution in application sides (e.g., network reliability, system biology, minimal model reasoning, and minimal unsatisfiable subsets) of answer set counting problem.

Contributions We have contributions in two directions: (i) engineering *scalable answer set counting techniques* and (ii) addressing *real-world applications* exploiting efficient answer set counting techniques. The contributions of our work are as follows:

- *Counting technique* (also covered in [10]): We develop an exact answer set counting method based on an alternative characterization of answer sets, addressing key limitations of existing approaches.

ICLP DC 2025: 21st Doctoral Consortium on Logic Programming, September 2025, Rende, Italy.

*You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

*Corresponding author.

†These authors contributed equally.

mahibuet045@gmail.com (M. Kabir)

https://mahi045.github.io/ (M. Kabir)

0000-0001-7551-0337 (M. Kabir)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

- *Counting technique* (also covered in [10]): We introduce ApproxASP, a scalable approximate counter that leverages hashing-based partitioning of the search space.
- *Counting Application* (not covered in [10]): We demonstrate the practical utility of answer set counting for network reliability estimation, achieving improved accuracy and efficiency over prior methods.
- *Counting Application* (not covered in [10]): We showcase the applicability of our approach in systems biology by applying answer set counting to analyze Boolean network dynamics.
- *Counting Application* (not covered in [10]): We present the theory of answer set counting within the context of minimal model counting.
- *Counting Application* (not covered in [10]): We present a minimal unsatisfiable subsets counting tool based on ASP solving.

2. Background and Problem Statement

An *answer set program* P consists of a set of rules, each rule is structured as follows:

$$\text{Rule } r\text{: } a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \tag{1}$$

where, $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are propositional variables or atoms, and k, m, n are non-negative integers. The notation $\mathsf{atoms}(P)$ denotes atoms within the program P. In rule r, the operator "not" denotes *default negation* [11]. For each rule r (eq. (1)), we adopt the following notations: the atom set $\{a_1, \ldots, a_k\}$ constitutes the *head* of r, denoted by $\mathsf{Head}(r)$, the set $\{b_1, \ldots, b_m\}$ is referred to as the *positive body atoms* of r, denoted by $\mathsf{Body}(r)^+$, and the set $\{c_1, \ldots, c_n\}$ is referred to as the *negative body atoms* of r, denoted by $\mathsf{Body}(r)^-$. A rule r is called a *constraint* when $\mathsf{Head}(r)$ contains no atom. A program P is called a *disjunctive logic program* if there is a rule $r \in P$ such that $|\mathsf{Head}(r)| \geq 2$ [12].

In ASP, an interpretation M over $\mathsf{atoms}(P)$ specifies which atoms are assigned true; that is, an atom a is true under M if and only if $a \in M$ (or false when $a \notin M$ resp.). An interpretation M satisfies a rule r, denoted by $M \models r$, if and only if $(\mathsf{Head}(r) \cup \mathsf{Body}(r)^-) \cap M \neq \emptyset$ or $\mathsf{Body}(r)^+ \setminus M \neq \emptyset$. An interpretation M is a *model* of P, denoted by $M \models P$, when $\forall_{r \in P} M \models r$. The *Gelfond-Lifschitz (GL) reduct* of a program P, with respect to an interpretation M, is defined as $P^M = \{\mathsf{Head}(r) \leftarrow \mathsf{Body}(r)^+ | r \in P, \mathsf{Body}(r)^- \cap M = \emptyset\}$ [13]. An interpretation M is an *answer set* of P if $M \models P$ and no $M' \subset M$ exists such that $M' \models P^M$. We denote the answer sets of program P using the notation $\mathsf{AS}(P)$.

Problem Formulation: Exact Answer Set Counting [14] Given an ASP program P, the exact answer set counting seeks to count the number of answer sets of P; more formally, the problem seeks to find $|\mathsf{AS}(P)|$.

Problem Formulation: Approximate Answer Set Counting [15] Given an ASP program P, tolerance parameter ϵ, and confidence parameter δ, the approximate answer set counting seeks to estimate the number of answer sets of P with a probabilistic guarantee; more formally, the approximate answer set counters returns a count c such that $\mathsf{Pr}[|\mathsf{AS}(P)|/1+\epsilon \leq c \leq (1+\epsilon) \times |\mathsf{AS}(P)|] \geq 1 - \delta$. Our approximate answer set counter invokes a polynomial number of calls to an ASP solver.

Clark's completion [11] or *program completion* is a technique to translate a normal program P into a propositional formula $\mathsf{Comp}(P)$ that is related but not semantically equivalent. Specifically, for each atom a in $\mathsf{atoms}(P)$, we perform the following steps:

1. Let $r_1, \ldots, r_k \in P$ such that $\mathsf{Head}(r_1) = \ldots = \mathsf{Head}(r_k) = a$, then we add the propositional formula $(a \leftrightarrow (\mathsf{Body}(r_1) \vee \ldots \vee \mathsf{Body}(r_k)))$ to $\mathsf{Comp}(P)$.
2. Otherwise, we add the literal $\neg a$ to $\mathsf{Comp}(P)$.

Finally, $\mathsf{Comp}(P)$ is derived by logically conjoining all the previously added constraints. Literature indicates that while every answer set of P satisfies $\mathsf{Comp}(P)$, the converse is not true [16].

Problem Formulation: Network Reliablity [6] The network reliability problem is defined on *probabilistic graphs*, where each edge is independently active with a specified probability. Given such a graph and a pair of nodes, the goal is to compute the probability that there exists a path of active edges connecting these two nodes—that is, the likelihood that these two nodes are reachable from one another. According to computational complexity, the network reliability problem is in #P [17].

Minimal trap spaces in System Biology [18] A Boolean Network (BN) is a useful formalism in system biology to model complex biological behaviour. A BN is defined over a set of variables or nodes, where each variable is associated with an *update* function, which identifies how the value of a variable updates over the time. A trap space is a *hypercube* in the state space of a BN that cannot be escaped once entered; that is, if the BN updates from a state s_1 to a state s_2, then both s_1 and s_2 lie within the trap space. The subset-minimal trap space has importance in systems biology due to its characterization in the dynamical landscape of BNs [19, 20].

Minimal Models [8] We often present a model as a set of atoms that are assigned to *true* [21]. A *minimal model* of a propositional formula is a model that is minimal under *set inclusion* among all models of the formula — none of its proper subset is a model of the formuala. The minimal model reasoning is a crucial task in *circumscription* [22, 23], *default logic* [24], *diagnosis* [25], and others.

Minimal Unsatisfiable Subsets [7] Given an unsatisfiable Boolean formula F, the minimal unsatisfiable subset (MUS) $F' \subset F$ is unsatisfiable and every proper subset of F' is satisfiable. A MUS of an unsatisfiable formula provides a concise explanation for the formula's unsatisfiability [26].

3. Related Work on Answer Set Counting

The decision version of normal logic programs is NP-complete; therefore, the ASP counting for normal logic programs is #P-complete [17] via a polynomial reduction [27]. Given the #P-completeness, a prominent line of work focused on ASP counting relies on translations from the ASP program to a CNF formula [16, 27, 28, 29]. Such translations often result in a large number of CNF clauses and thereby limit practical scalability for *non-tight* ASP programs.

Eiter et al. [30] introduced $\mathrm{T_P}$-*unfolding* to break cycles and produce a tight program. They proposed an ASP counter called aspmc, that performs a *treewidth-aware Clark completion* from a cycle-free program to a CNF formula. Jakl, Pichler, and Woltran [31] extended the tree decomposition based approach for #SAT due to Samer and Szeider [32] to ASP and proposed a *fixed-parameter tractable* (FPT) algorithm for ASP counting. Fichte et al. [33, 34] revisited the FPT algorithm due to Jakl et al. [31] and developed an exact model counter, called DynASP, that performs well on instances with *low treewidth*. Aziz et al. [3] extended a propositional model counter to an answer set counter by integrating unfounded set detection. ASP solvers [35] can count answer set via enumeration, which is suitable for a sufficiently small number of answer sets.

Subtraction-based answer set counters have also been proposed [36, 37, 38]. The *subtraction-based* answer set counter iascar is specifically designed for normal programs. The approach first computes an overcount, and subsequently refines the overcount by enforcing *external support* for each loop and applying the *inclusion-exclusion principle*. Kabir et al. [15] focused on lifting hashing-based techniques to ASP counting, resulting in an approximate counter, called ApproxASP, with (ε, δ)-guarantees. Kabir et al. [14] introduced an ASP counter that utilizes a sophisticated Boolean formula, termed the copy formula, which features a compact encoding.

4. Answer Set Counting Techniques

We have already engineered two ASP counters: SharpASP [14] and ApproxASP [15]. SharpASP[1] is an exact answer set counter and ApproxASP[2] is an approximate answer set counter.

The principal contribution of SharpASP is to design a scalable answer set counter, without a substantial increase in the size of the transformed propositional formula, particularly when addressing circular dependencies. The key idea behind a substantial reduction in the size of the transformed formula is an alternative yet correlated perspective of defining answer sets. This alternative definition formulates the answer set counting problem on a pair of Boolean formulas (F, G), where the formula F over-approximates the search space of answer sets, while the formula G exploits *justifications* to identify answer sets correctly. We set $F = \mathsf{Comp}(P)$ since every answer set satisfies Clark completion. Note that $\mathsf{Comp}(P)$ overapproximates answers sets of P. We propose another formula, named *copy formula*, denoted as $\mathsf{Copy}(P)$, which comprises a set of (implicitly conjoined) implications defined as follows:

1. (type 1) for every $v \in \mathsf{LA}(P)$, the implication $v' \rightarrow v$ is in $\mathsf{Copy}(P)$.
2. (type 2) for every rule $x \leftarrow a_1, \ldots, a_k, b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_n$ in P, where $x \in \mathsf{LA}(P)$, $\{a_1, \ldots, a_k\} \subseteq \mathsf{LA}(P)$ and $\{b_1, \ldots, b_m\} \cap \mathsf{LA}(P) = \emptyset$, the implication $a_1{}' \wedge \ldots \wedge a_k{}' \wedge b_1 \wedge \ldots \wedge b_m \wedge \neg c_1 \wedge \ldots \wedge \neg c_n \rightarrow x'$ is in $\mathsf{Copy}(P)$.
3. No other implication is in $\mathsf{Copy}(P)$.

For each satisfying assignment $M \models \mathsf{Comp}(P)$, we have the following observations:

- if $M \in \mathsf{AS}(P)$, then $\mathsf{Copy}(P)_{|M} = \emptyset$
- if $M \notin \mathsf{AS}(P)$, then $\mathsf{Copy}(P)_{|M} \neq \emptyset$

where $\mathsf{Copy}(P)_{|M}$ denotes the *unit propagation* of M on $\mathsf{Comp}(P)$. We integrate these observations into propositional model counters to engineer an answer set counter.

Within ApproxASP, we present a scalable approach to approximate the number of answer sets. Inspired by approximate model counter ApproxMC [39], our approach is based on systematically adding parity (XOR) constraints to ASP programs to divide the search space uniformly and independently. We prove that adding random XOR constraints partitions the answer sets of an ASP program. When a randomly chosen partition is *quite small*, we can approximate the number of answer sets by simple multiplication. The XOR semantic in answer set programs was initiated by Everardo et al. [40]. In practice, we use a *Gaussian elimination*-based approach by lifting ideas from SAT to ASP and integrating them into a state-of-the-art ASP solver.

5. Practical Implementations of Answer Set Counters

We implemented prototypes of both SharpASP, on top of the existing propositional model counter SharpSAT-TD (denoted as SharpASP (STD) and ApproxASP, on top of ASP solver Clingo. Finally, we empirically evaluate their performance against existing counting benchmarks used in answer set counting literature [3, 30, 34].

Exact ASP Counter: SharpASP Our extensive empirical analysis of 1470 benchmarks demonstrates significant performance gain over current state-of-the-art exact answer set counters. The result demonstrated is presented in Table 1 and the rightmost column presents the result of SharpASP. Specifically, by using SharpASP, we were able to solve 1023 benchmarks with a PAR2 score of 3373, whereas using prior state-of-the-art, we could only solve 869 benchmarks with a PAR2 score of 4285. A detailed experimental analysis revealed that the strength of SharpASP is that it spends less time in binary constraint propagation while making more decisions compared to off-the-shelf propositional model counters.

[1] https://github.com/meelgroup/SharpASP
[2] https://github.com/meelgroup/ApproxASP2

	clingo	ASProb	aspmc+STD	lp2sat+STD	SharpASP (STD)
#Solved	869	188	840	776	**1023**
PAR2	4285	8722	4572	5084	**3373**

Table 1
The performance comparison of SharpASP vis-a-vis other ASP counters in terms of the number of solved instances and PAR2 scores.

		Clingo	DynASP	Ganak	ApproxMC	ApproxASP
Normal	#Solved (1500)	738	47	973	**1325**	1323
	PAR-2	5172	9705	3606	**1200**	1218
Disjunc.	#Solved (200)	177	0	0	0	**185**
	PAR2	1372	10000	10000	10000	**795**

Table 2
A runtime performance comparison of Clingo, DynASP, Ganak, ApproxMC, and ApproxASP was conducted on 1500 normal and 200 disjunctive logic program instances.

Approximate ASP Counter: ApproxASP Table 2 presents the result of ApproxASP with state-of-the-art answer set counters. ApproxASP performs well in disjunctive logic programs. ApproxASP solved 185 instances among 200 instances, while the best ASP solver clingo solved a total of 177 instances. In addition, on normal logic programs, ApproxASP performs on par with state-of-the-art approximate model counter ApproxMC.

6. Answer Set Counting Applications

We evaluated our answer set counters on practical applications in network reliability, systems biology, and minimal model reasoning.

Counting Application 1: Network Relability We introduced RelNet-ASP[3] [6], a network reliability estimator that integrates answer set counting with weighted model counting techniques. Within this framework, we reduced network reliability to weighted answer set counting problem. To simulate the probabilistic nature of edges, RelNet-ASP transforms a *weighted graph* into an *unweighted graph*, as proposed in the literature on weighted model counting [41]. The related transformation is known as *chain* graphs — if an edge is active with a probability of $\frac{k}{2^m}$, then the probability can be simulated by a *series-parallel* graph of size m; i.e., the graph has m edges. We baselined RelNet-ASP with off-the-shelf exact and approximate network reliability estimators. Our empirical evaluation on large real-world instances reveals that RelNet-ASP demonstrates superior performance, achieving the best trade-off between accuracy and efficiency among existing reliability estimators. In particular, RelNet-ASP achieved a TAP score[4] of 491, while state-of-the-art estimators can achieve a TAP score of 690.

Counting Application 2: System Biology We demonstrated the effectiveness of ApproxASP in addressing the problem of counting minimal trap spaces[5], a key task in quantifying the *emergence* and *robustness* of a *phenotype* in BNs [43]. We formulated biologically meaningful variants of the minimal trap space counting problems [44], including: (i) straightforward counting, (ii) counting subject to specific *properties* (or phenotype), and (iii) counting *perturbations* upon selected variables. We introduced novel and efficient reductions of these problems to answer set counting. Our reductions leverage the concept

[3] https://github.com/meelgroup/RelNet-ASP
[4] The TAP score [42] is a performance metric assessing both accuracy and efficiency; the lower the better.
[5] https://github.com/meelgroup/bn-counting

of *facets* [45] to encode structural properties. To model perturbations, we construct a new BN where assignments to designated variables represent interventions in the original network. Experimentally, we observed that ApproxASP scales to larger Boolean Network (BN) instances, successfully counting minimal trap spaces in networks with up to 4000 variables. In contrast, existing counting techniques are limited to instances with at most 321 variables.

Counting Application 3: Minimal Model Reasoning We proposed minLB[6] [8], a method for computing lower bounds on the number of minimal models of a propositional formula. Our approach reduces the problem of minimal model counting to answer set counting by encoding minimal models as answer sets of a suitably constructed disjunctive ASP program. We propose two techniques for computing lower bounds on the minimal model count: (i) a method based on *knowledge compilation*, and (ii) a *hashing-based* counting approach. The resulting tool, minLB, builds on established answer set counting techniques to compute these lower bounds efficiently. Our experimental evaluation revealed that minLB computed better lower bound than existing minimal model reasoning systems.

Counting Application 4: MUS Counting We proposed MUS-ASP [7], ASP solving-based MUS enumeration framework At a high level, given an unsatisfiable Boolean formula F, we formulate a disjunctive ASP program P such that the answer sets of P correspond to the unsatisfiable subsets of F. To compute MUSes, we focus on *subset minimal* answer sets with respect to a *target* set of atoms. This reduction allows us to harness the efficiency of subset minimal answer set solvers to compute MUSes. While the complete MUS enumeration is intractable in theory, MUS-ASP significantly outperforms state-of-the-art MUS counters in practice. In our experiments, MUS-ASP successfully counted 925 instances, compared to 887 instances counted by off-the-shelf MUS counters.

7. Open Issues and Expected Achievements

Model counting is a computationally intractable problem, falling into the complexity class #P for normal programs and #·co-NP for disjunctive logic programs [34], posing substantial challenges for building scalable answer set counters. Our empirical analysis reveals that while our engineered counters perform well on specific classes of problems, they struggle with others. The wide range of application domains further complicates the development of specialized ASP counters tailored to particular use cases. Additionally, we observed cases where existing tools outperform our sharpASP system. Bridging this performance gap by incorporating the strengths of these tools into sharpASP remains an open and challenging research direction.

Trusting the output of answer set counters requires confidence in their underlying implementations. However, to the best of our knowledge, there is limited work on the certification of answer set counting. Our goal is to bridge this gap by developing techniques that combine certification with answer set counting, thereby enhancing the reliability of the results.

Declaration on Generative AI Usage

We used generative AI tools only for minor language editing after the paper was fully drafted by the author. Specifically, ChatGPT (OpenAI) was used to improve grammar and phrasing; no text, figures, code, ideas, or experimental content were generated or altered beyond language edits. Importantly, the author reviewed and verified all changes.

[6] https://github.com/meelgroup/MinLB

References

[1] V. W. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, The Logic Programming Paradigm: a 25-Year Perspective (1999) 375–398.

[2] A. Brik, J. Remmel, Diagnosing automatic whitelisting for dynamic remarketing ads using hybrid asp, in: LPNMR, Springer, 2015, pp. 173–185.

[3] R. A. Aziz, G. Chu, C. Muise, P. J. Stuckey, Stable model counting and its application in probabilistic logic programming, in: AAAI, 2015.

[4] J. K. Fichte, S. A. Gaggl, D. Rusovac, Rushing and strolling among answer sets–navigation made easy, in: AAAI, volume 36, 2022, pp. 5651–5659.

[5] S. Hahn, T. Janhunen, R. Kaminski, J. Romero, N. Rühling, T. Schaub, Plingo: a system for probabilistic reasoning in clingo based on lp mln, in: RULEML+RR, Springer, 2022, pp. 54–62.

[6] M. Kabir, K. S. Meel, A fast and accurate ASP counting based network reliability estimator, in: LPAR, volume 94, 2023, pp. 270–287.

[7] M. Kabir, K. S. Meel, An ASP-based framework for muses, arXiv preprint arXiv:2507.03929 (to appear in ICLP 2025) (2025).

[8] M. Kabir, K. S. Meel, On lower bounding minimal model count, TPLP 24 (2024) 586–605.

[9] M. Kabir, K. S. Meel, A simple and effective ASP-based tool for enumerating minimal hitting sets, arXiv preprint arXiv:2507.09194 (2025).

[10] M. Kabir, Answer set counting and its applications, arXiv preprint arXiv:2502.09231 (2025).

[11] K. L. Clark, Negation as failure, in: Logic and data bases, Springer, 1978, pp. 293–322.

[12] R. Ben-Eliyahu, R. Dechter, Propositional semantics for disjunctive logic programs, Annals of Mathematics and Artificial intelligence 12 (1994) 53–87.

[13] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming., in: ICLP/SLP, volume 88, 1988, pp. 1070–1080.

[14] M. Kabir, S. Chakraborty, K. S. Meel, Exact ASP counting with compact encodings, in: AAAI, volume 38, 2024, pp. 10571–10580.

[15] M. Kabir, F. O. Everardo, A. K. Shukla, M. Hecher, J. K. Fichte, K. S. Meel, ApproxASP–a scalable approximate answer set counter, in: AAAI, volume 36, 2022, pp. 5755–5764.

[16] F. Lin, Y. Zhao, Assat: computing answer sets of a logic program by sat solvers, Artificial Intelligence 157 (2004) 115 – 137. URL: http://www.sciencedirect.com/science/article/pii/S0004370204000578. doi:https://doi.org/10.1016/j.artint.2004.04.004.

[17] L. G. Valiant, The complexity of enumeration and reliability problems, SIAM Journal on Computing 8 (1979) 410–421.

[18] J. D. Schwab, S. D. Kühlwein, N. Ikonomi, M. Kühl, H. A. Kestler, Concepts in boolean network modeling: What do they all mean?, Computational and structural biotechnology journal 18 (2020) 571–582.

[19] S. Chevalier, C. Froidevaux, L. Paulevé, A. Zinovyev, Synthesis of boolean networks from biological dynamical constraints using answer-set programming, in: ICTAI, IEEE, 2019, pp. 34–41.

[20] L. Paulevé, J. Kolčák, T. Chatain, S. Haar, Reconciling qualitative, abstract, and scalable modeling of biological networks, Nature communications 11 (2020) 4256.

[21] F. Angiulli, R. Ben-Eliyahu, F. Fassetti, L. Palopoli, On the tractability of minimal model computation for some CNF theories, Artificial Intelligence 210 (2014) 56–77.

[22] V. Lifschitz, Computing circumscription., in: IJCAI, volume 85, 1985, pp. 121–127.

[23] J. McCarthy, Circumscription—a form of non-monotonic reasoning, Artificial intelligence 13 (1980) 27–39.

[24] R. Reiter, A logic for default reasoning, Artificial intelligence 13 (1980) 81–132.

[25] J. De Kleer, A. K. Mackworth, R. Reiter, Characterizing diagnoses and systems, Artificial intelligence 56 (1992) 197–222.

[26] M. H. Liffiton, A. Previti, A. Malik, J. Marques-Silva, Fast, flexible MUS enumeration, Constraints 21 (2016) 223–250.

[27] T. Janhunen, I. Niemelä, Compact Translations of Non-disjunctive Answer Set Programs to Propo-

sitional Clauses, 2011, pp. 111–130. doi:10.1007/978-3-642-20832-4_8.
[28] T. Janhunen, Representing normal programs with clauses, in: ECAI, volume 16, 2004, p. 358.
[29] T. Janhunen, Some (in) translatability results for normal logic programs and propositional theories, Journal of Applied Non-Classical Logics 16 (2006) 35–86.
[30] T. Eiter, M. Hecher, R. Kiesel, aspmc: New frontiers of algebraic answer set counting, Artificial Intelligence 330 (2024) 104109.
[31] M. Jakl, R. Pichler, S. Woltran, Answer-set programming with bounded treewidth., in: IJCAI, volume 9, 2009, pp. 816–822.
[32] M. Samer, S. Szeider, Algorithms for propositional model counting, Journal of Discrete Algorithms 8 (2010) 50–64.
[33] J. K. Fichte, M. Hecher, Treewidth and counting projected answer sets, in: LPNMR, Springer, 2019, pp. 105–119.
[34] J. K. Fichte, M. Hecher, M. Morak, S. Woltran, Answer set solving with bounded treewidth revisited, in: LPNMR, 2017, pp. 132–145.
[35] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, Artificial Intelligence 187 (2012) 52–89.
[36] J. K. Fichte, S. A. Gaggl, M. Hecher, D. Rusovac, IASCAR: Incremental answer set counting by anytime refinement, TPLP 24 (2024) 505–532.
[37] M. Hecher, R. Kiesel, The impact of structure in answer set counting: fighting cycles and its limits, in: KR, 2023, pp. 344–354.
[38] M. Kabir, S. Chakraborty, K. S. Meel, Counting answer sets of disjunctive answer set programs, volume XX, Cambridge University Press, 2025, p. XXX.
[39] S. Chakraborty, K. S. Meel, M. Y. Vardi, A scalable approximate model counter, in: CP, Springer, 2013, pp. 200–216.
[40] F. Everardo, T. Janhunen, R. Kaminski, T. Schaub, The return of xorro, in: LPNMR, Springer, 2019, pp. 284–297.
[41] S. Chakraborty, D. Fried, K. S. Meel, M. Y. Vardi, From weighted to unweighted model counting., in: IJCAI, 2015, pp. 689–695.
[42] D. Agrawal, Y. Pote, K. S. Meel, Partition function estimation: A quantitative study, in: IJCAI, 2021.
[43] N. Beneš, L. Brim, S. Pastva, D. Šafránek, E. Šmijáková, Phenotype control of partially specified boolean networks, in: CMSB, Springer, 2023, pp. 18–35.
[44] M. Kabir, V.-G. Trinh, S. Pastva, K. S. Meel, Scalable counting of minimal trap spaces and fixed points in boolean networks, arXiv preprint arXiv:2506.06013 (to appear in CP 2025) (2025).
[45] C. Alrabbaa, S. Rudolph, L. Schweizer, Faceted answer-set navigation, in: RuleML+RR, Springer, 2018, pp. 211–225.