Utilizing Treewidth for Quantitative Reasoning on Epistemic Logic Programs (Extended Abstract)*

Viktor Besin[1] , **Markus Hecher**[1,2] , **Stefan Woltran**[1]
[1]TU Wien, Vienna, Austria
[2]University of Potsdam, Potsdam, Germany
{vbesin, hecher, woltran}@dbai.tuwien.ac.at

Abstract

Extending the popular Answer Set Programming (ASP) paradigm by introspective reasoning capacities has received increasing interest within the last years. Particular attention is given to the formalism of epistemic logic programs (ELPs) where standard rules are equipped with modal operators which allow to express conditions on literals for being known or possible, i.e., contained in all or some answer sets, respectively. ELPs thus deliver multiple collections of answer sets, known as world views. Employing ELPs for reasoning problems so far has mainly been restricted to standard decision problems (complexity analysis) and enumeration (development of systems) of world views. In this paper, we first establish quantitative reasoning for ELPs, where the acceptance of a certain set of literals depends on the number (proportion) of world views that are compatible with the set. Second, we present a novel system capable of efficiently solving the underlying counting problems required for quantitative reasoning. Our system exploits the graph-based measure treewidth by iteratively finding (graph) abstractions of ELPs.

1 Introduction

Answer Set Programming (ASP) [Brewka *et al.*, 2011] is a well-studied problem modeling and solving framework that is particularly suited for solving problems related to knowledge representation and reasoning and artificial intelligence, see, e.g., the work [Brewka *et al.*, 2011]. In ASP, questions are modeled in the form of *logic programs (LPs)*, which can be seen as a rule-based language whose solutions are referred to by *answer sets* and which has been significantly extended over the time. The major driver in enabling logic programs for a broad use in both academia and industry was the development of efficient solvers. However, while the ASP framework is powerful, its limits in terms of expressiveness are visible when turning the attention to epistemic specifications.

The idea of these epistemic specifications, which dates back to the early 90s [Gelfond, 1991], allows to precisely describe the behavior of rational agents who are capable of reasoning over multiple worlds. There, depending on whether some objections are possible (true in some world) or known (i.e., true in all worlds) certain consequences have to be derived. This is often modeled by means of operators **K** or **M**, which represents that certain objections are *known to be true* or are *possibly true*, respectively. Internally these operators can be translated to *epistemic negation* **not**, which expresses that some objection is *not known*, i.e., not true in all worlds. Enhancing standard rules by such operators leads to the development of *epistemic logic programs (ELPs)*. Indeed, depending on the different semantics for ELPs, which have been developed and refined over the years [Truszczynski, 2011; Kahl *et al.*, 2015; Shen and Eiter, 2016; Cabalar *et al.*, 2019], usual reasoning problems like *world view existence* and certain extensions reach the third and fourth level of the polynomial hierarchy, respectively, and are thus considered significantly harder than standard ASP [Eiter and Gottlob, 1995].

In this work, we take a step further and initiate the study of *quantitative reasoning* for ELPs, where decisions concerning the acceptance of a given set of literals depend on the actual *number (proportion) of world views* compatible with the set. This allows us to reason about the acceptance of certain literals based on the likelyhood of being contained in an arbitrary world view. To the best of our knowledge, a few works on quantitiative reasoning in ASP exist [Fierens *et al.*, 2015], but it has not yet been studied for ELPs. As a second contribution we present a new system tailored for quantitiative reasoning in ELPs. Although there has been progress in developing ELP solvers (e.g., EP-ASP [Son *et al.*, 2017], selp [Bichler *et al.*, 2020] and a very recent extension of clingo for epistemic logic programs, called eclingo [Cabalar *et al.*, 2020]), these approaches basically rely on reducing ELP problems to standard ASP. Thus, these solvers typically materialize all world views, which is not necessary for quantitative reasoning. We take here a novel route by utilizing ideas from parameterized algorithmics which appear better suited for counting problems that underly the quantitative reasoning approach.

Our approach works on abstractions of the internal (graph) structure of ELPs; i.e., we take the *primal graph*[1] of an ELP and contract certain paths between nodes referring to epis-

*This is an extended abstract of [Besin *et al.*, 2021].

[1]The (E)LP's primal graph comprises the variables (atoms) with an edge between two atoms, whenever appearing together in a rule.

temic literals. On this graph we implicitly utilize the measure *treewidth*, measuring the tree-likeness of a given graph. Treewidth gives rise to a so-called *tree decomposition (TD)*, which allows to solve a problem by following a divide-and-conquer approach, where world views of ELPs are computed by solving subprograms and combining world views accordingy. Our solver adheres to this approach, where we approximate suitable abstractions of the primal graph structure of an ELP in order to evaluate the program in a way that is guided along a TD of the abstraction. So, the idea of these abstractions compared to the full primal graph is to decrease treewidth such that still structural information can be utilized. In addition to the abstractions and in order to efficiently apply our approach also to (practical) ELPs of high treewidth, we present the following additions: (i) We nest the computation of abstractions and (ii) for hard combinatorial subprograms of (E)LPs, we employ existing standard solvers like (e)clingo.

Related Work. Treewidth was utilized for the evaluation of standard LPs [Jakl *et al.*, 2009; Hecher, 2022; Fandinno and Hecher, 2021] and competitions [Fichte *et al.*, 2021a]. Abstractions were also stipulated before, but in a different context [Hecher *et al.*, 2020b; Saribatur and Eiter, 2021] or for establishing theoretical results [Ganian *et al.*, 2017]. However, we improve an existing algorithm [Hecher *et al.*, 2020a] and while the solver selp [Bichler *et al.*, 2020] uses TDs for breaking-up rules, our solver is the first implementation of TD-guided solving for ELPs. Studies of measures different from treewidth have been conducted for LPs [Lonc and Truszczynski, 2003; Bliem *et al.*, 2016; Fichte *et al.*, 2019].

2 Preliminaries

Answer Set Programming (ASP). We follow standard definitions of propositional ASP [Brewka *et al.*, 2011] and refer by *literal* to an atom or its negation. A *program (LP)* P is a set of *rules* of the form $a_1 \vee \cdots \vee a_k \leftarrow a_{k+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$. The right side of "$\leftarrow$" of a rule is called *body*; the left side is referred to by *head*. For a rule r, we let $H_r := \{a_1, \ldots, a_\ell\}$, $B_r^+ := \{a_{\ell+1}, \ldots, a_m\}$, and $B_r^- := \{a_{m+1}, \ldots, a_n\}$. We denote the *atoms* occurring in a rule r or program P by $\mathsf{at}(r)$ or $\mathsf{at}(\mathsf{P})$, respectively. An *interpretation $I \subseteq \mathsf{at}(\mathsf{P})$ satisfies* a rule r if $(H_r \cup B_r^-) \cap I \neq \emptyset$, or $B_r^+ \setminus I \neq \emptyset$, or both. I is a *model* of P if it satisfies all rules of P. The *Gelfond-Lifschitz (GL) reduct* of P under I is the program P^I obtained from P by first removing all rules r with $B_r^- \cap I \neq \emptyset$ and then removing all $\neg z$ where $z \in B_r^-$ from the remaining rules r. Then, I is an *answer set* of a program P if I is a minimal model of P^I; $AS(\mathsf{P})$ are the answer sets of P.

Example 1. *Consider the logic program* $\mathsf{P} = \{a \vee b \leftarrow;\ c \leftarrow \neg d;\ d \leftarrow \neg c\}$. *The answer sets of* P *are* $\{a,c\}$, $\{a,d\}$, $\{b,c\}$, *and* $\{b,d\}$.

Tree Decompositions (TDs). For preliminaries in graph theory, we refer to the literature, e.g., [Diestel, 2012]. Let $G = (V,E)$ be a graph, T a (rooted) tree, and χ a labeling function that maps every node t of T to a subset $\chi(t) \subseteq V$ called the *bag* of t. The pair $\mathscr{T} = (T, \chi)$ is called a *tree decomposition (TD)* [Bodlaender and Kloks, 1996] of G iff (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 $\chi(r) \cap \chi(t) \subseteq \chi(s)$. The *width* of a TD is the cardinality of its largest bag minus one. The *treewidth* of a graph G is the minimum width over all TDs of G.

3 Probabilistic Reasoning for Epistemic LPs

An *epistemic logic program (ELP)* Π is an extension of a LP, where each rule body can contain *epistemic literals* of the form $\mathbf{not}\ell$ using literal ℓ and *epistemic negation* $\mathbf{not}$. Then, $\mathsf{at}(r)$ denotes the atoms ocurring in such a rule r, $\mathsf{e\text{-}at}(r)$ denotes the *epistemic atoms*, i.e., those used in epistemic literals of r, and $\mathsf{a\text{-}at}(r)$ refers to the atoms of r not under epistemic negation. We call r *purely-epistemic* if $\mathsf{a\text{-}at}(r) = \emptyset$. We write $\mathbf{K}\ell$ ("ℓ is *known*") for a literal ℓ, referring to $\neg\mathbf{not}\ell$.

While there are many different semantics [Gelfond, 1991; Truszczynski, 2011; Kahl *et al.*, 2015; Shen and Eiter, 2016], we follow the approach of [Gelfond, 1991], syntactically denoted according to [Morak, 2019]. A *world view interpretation (WVI)* I for Π is a consistent set I of literals over a set $A \subseteq \mathsf{at}(\Pi)$ of atoms. Intuitively, every $\ell \in I$ is considered "known" and every $a \in A$ with $\{a, \neg a\} \cap I = \emptyset$ is treated as "possible". The *epistemic reduct* Π^I [Gelfond, 1991] of program Π for a WVI I over A is defined by $\Pi^I := \{r^I \mid r \in \Pi\}$, where r^I denotes rule r where each epistemic literal $\mathbf{not}\ell$ over A is replaced by $\bot$ if $\ell \in I$, and by $\top$ otherwise.

Let $\mathscr{I}$ be a set of interpretations over a set A of atoms. Formally, a WVI I is *compatible* with $\mathscr{I}$ if: (1.) $\mathscr{I} \neq \emptyset$; (2.) for each atom $a \in I$, it holds that for each $J \in \mathscr{I}$, $a \in J$; (3.) for each $\neg a \in I$, we have for each $J \in \mathscr{I}$, $a \notin J$; and (4.) for each atom $a \in A$ with $\{a, \neg a\} \cap I = \emptyset$, there are $J, J' \in \mathscr{I}$, such that $a \in J$, but $a \notin J'$. Then, a WVI I over $\mathsf{at}(\Pi)$ is a *world view (WV)* of Π iff I is compatible with the set $AS(\Pi^I)$. Deciding for WV existence is Σ_3^P-complete [Truszczynski, 2011].

Example 2. *Consider program* $\Pi := \mathsf{P} \cup \{a \leftarrow \neg\mathbf{K}b;\ b \leftarrow \neg\mathbf{K}a;\ c \leftarrow \neg\mathbf{K}d;\ d \leftarrow \neg\mathbf{K}c;\ \leftarrow \neg\mathbf{K}a, \neg\mathbf{K}\neg a;\ \leftarrow \neg\mathbf{K}b, \neg\mathbf{K}\neg b;\ \leftarrow \neg\mathbf{K}a, \neg\mathbf{K}c;\ \leftarrow \neg\mathbf{K}a, \neg\mathbf{K}b, \mathbf{K}c;\ \leftarrow \mathbf{K}c, \mathbf{K}d\}$, *where* P *is given in Example 1. When constructing a WVI I over* $\mathsf{e\text{-}at}(\Pi)$ *one guesses for each atom* $a \in \mathsf{e\text{-}at}(\Pi)$ *either (1)* $a \in I$, *(2)* $\neg a \in I$ *or (3)* $\{a, \neg a\} \cap I = \emptyset$ *as described earlier, i.e., for the three atoms in* $\mathsf{e\text{-}at}(\Pi)$ *we obtain* 3^4 *possibilities. Consider* $I_1 = \{a, d, \neg b, \neg c\}$ *with its epistemic reduct* $\Pi^{I_1} = \mathsf{P} \cup \{a;\ d\}$. *Note that the epistemic reduct is an LP, since by semantics of logic programs, rules r with* $\bot \in B_r^+$ *or* $\top \in B_r^-$ *can obviously be dropped. Since* $AS(\Pi^{I_1}) = \{\{a,d\}\}$, *compatibility of* I_1 *can be checked trivially which validates* I_1 *as WV of* Π. *Similarly WVIs* $I_2 = \{a, c, \neg b, \neg d\}$ *and* $I_3 = \{b, c, \neg a, \neg d\}$ *can be constructed and correctly validated as WVs, i.e.,* $WVS(\Pi) = \{I_1, I_2, I_3\}$.

Counting and Reasoning. In this work, we mainly cover the following counting problem, which can then be used as a basis to solve (quantitative) reasoning problems.

Definition 1 (World View Counting). *Let Π be an ELP and Q be a WVI, called* query, *over atoms* $\mathsf{at}(\Pi)$. *Then, the problem* $\#\mathrm{ELP}(\Pi, Q)$ *asks to count the number of world views W with* $Q \cap \mathsf{at}(\Pi) \subseteq W$ *and* $\{a \mid \neg a \in Q\} \cap W = \emptyset$.

As a special case, where $Q = \emptyset$, problem $\#\mathrm{ELP}(\Pi, \emptyset)$ amounts to counting world views. Interestingly, the problem

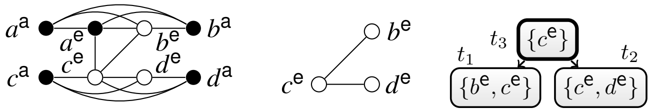

Figure 1: Primal graph G_Π (left) of Π, the nested primal graph G^A_Π for $A = \{b, c, d\}$ (middle), and a TD $\mathcal{T}$ of graph G^A_Π (right).

can be used to reason about the likelihood of an atom or a set of atoms being contained in an arbitrary world view:

Definition 2 (Probability of World View Acceptance)**.** *Let* Π *be an ELP and* Q *be a WVI over* $\mathsf{at}(\Pi)$*. We define the* probability *of* Q *being compatible with a world view by* $\mathsf{prob}(\Pi, Q) := \frac{\#\mathrm{ELP}(\Pi,Q)}{\#\mathrm{ELP}(\Pi,\emptyset)}$.

Consequently, counting allows us to reason about the degree of believing in literals being part of world views. This degree of belief can then be used for accepting literals depending on its probability exceeding a certain value, referred to by *probabilistic world view acceptance*.

Example 3. *Recall* Π *from Example 2. Given* $Q := \{a, \neg b\}$*, the number* $\#\mathrm{ELP}(\Pi, Q) = 2$ *agrees with the number of WVs including a, but not b. The probability* $\mathsf{prob}(\Pi, Q) = \frac{2}{3}$ *can be used to argue about the chance of a WV of* Π *containing a but not b, which renders a and* $\neg b$ *very likely.*

4 Abstractions for Dynamic Programming

Algorithms utilizing treewidth for solving a problem in linear time typically proceed by *dynamic programming (DP)* along a TD. Thereby, the tree is traversed in post-order and at each node t of the tree, information is gathered [Bodlaender and Kloks, 1996] in a table τ_t, such that the root table yields the final result. A *table* τ_t is a set of rows, which are sequences or tuples of fixed length. The actual length, content, and meaning of the rows depend on the algorithm that derives tables.

Before we sketch our algorithm, we need a *graph representation*. Let therefore Π be an epistemic logic program. Then, the *primal graph* G_Π uses atoms and epistemic atoms as vertices and it is defined by $G_\Pi := (\{a^\circ \mid a \in \circ\text{-}\mathsf{at}(\Pi), \circ \in \{\mathsf{a}, \mathsf{e}\}\}, E)$, where $E := \{\{a^\circ, b^\star\} \mid r \in \Pi, a \in \circ\text{-}\mathsf{at}(r), b \in \star\text{-}\mathsf{at}(r), \{\circ, \star\} \subseteq \{\mathsf{a}, \mathsf{e}\}\} \cup \{\{a^{\mathsf{a}}, a^{\mathsf{e}}\} \mid a \in \mathsf{e}\text{-}\mathsf{at}(\Pi)\}$.

Example 4. *Figure 1 (left) shows primal graph* G_Π *for our running example* Π*, assuming simple edges (no self-loops).*

For our purposes, we require suitable abstractions of G_Π. In the following, for $S \subseteq \mathsf{at}(\Pi)$ we denote $S^{\mathsf{e}} := \{s^{\mathsf{e}} \mid s \in S\}$.

Definition 3. *Assume an* abstraction $A \subseteq \mathsf{e}\text{-}\mathsf{at}(\Pi)$*. A* non-epistemic path *in* G_Π *is of the form* $a^{\mathsf{e}}, v_1, \ldots, v_l, b^{\mathsf{e}}$ *with* $\{v_1, \ldots, v_l\} \cap A^{\mathsf{e}} = \emptyset$*. The* nested primal graph G^A_Π *is defined by* $G^A_\Pi := (A^{\mathsf{e}}, E')$ *with* $E' := \{\{a^{\mathsf{e}}, b^{\mathsf{e}}\} \mid \{a, b\} \subseteq A$*, there is a non-epistemic path from* a^{e} *to* b^{e} *in* $G_\Pi\}$*.*

Example 5. *Given epistemic atoms* $A = \{b, c, d\}$*, the nested primal graph* G^A_Π *for our running example* Π *can be constructed with edges* $\{b^{\mathsf{e}}, c^{\mathsf{e}}\}$ *and* $\{c^{\mathsf{e}}, d^{\mathsf{e}}\}$ *through any non-epistemic paths between the two correlating vertices in* G_Π*, shown in Figure 1 (middle). Observe that* $G^{\mathsf{e}\text{-}\mathsf{at}(\Pi)}_\Pi = G_\Pi$*.*

Interestingly, these abstractions can be used to separate solving effort between DP and nested (standard) solving. If we used the primal graph, i.e., the trivial abstraction $G^{\mathsf{e}\text{-}\mathsf{at}(\Pi)}_\Pi$, then full effort goes into DP (no nested solving). If on the other hand we used the empty abstraction, i.e., $G^{\emptyset}_\Pi = (\emptyset, \emptyset)$, then no effort is left for DP (full effort into nested solving). Even further, the TD partitions the effort for nested solving.

Example 6. *Figure 1 (right) shows a corresponding TD* $\mathcal{T} = (T, \chi)$ *for the nested primal graph* G^A_Π*, using ELP* Π *and abstraction* $A = \{b, c, d\}$ *from Example 5. The purely-epistemic rules (constraints) comprising only atoms, whose vertices are in the corresponding bags, are called* bag programs*, which are evaluated via DP. For node* t_1*, this program is* $\{\leftarrow \neg\mathbf{K}b, \neg\mathbf{K}\neg b\}$*, since* $\chi(t_1) = \{b^{\mathsf{e}}, c^{\mathsf{e}}\}$*; for* t_2 *this is* $\{\leftarrow \mathbf{K}c, \mathbf{K}d\}$*, since* $\chi(t_2) = \{c^{\mathsf{e}}, d^{\mathsf{e}}\}$*. Intuitively, the remaining rules of* Π *are subject to nesting. When removing vertices* $\{b^{\mathsf{e}}, c^{\mathsf{e}}, d^{\mathsf{e}}\}$ *from* G_Π *one can identify two connected components; one over* $\{a^{\mathsf{a}}, b^{\mathsf{a}}, a^{\mathsf{e}}\}$ *and the other component over* $\{c^{\mathsf{a}}, d^{\mathsf{a}}\}$*. Observe that vertices* $a^{\mathsf{a}}, b^{\mathsf{a}}, a^{\mathsf{e}}$ *are only contained in non-epistemic paths involving epistemic atoms in* $\chi(t_1)$*; vertices* $c^{\mathsf{a}}, d^{\mathsf{a}}$ *are only contained in non-epistemic paths using* $\chi(t_2)$*. Consequently, our approach defines for every node a* nested bag program*, which for* t_1 *comprises* $\{a \vee b \leftarrow; a \leftarrow \neg\mathbf{K}b;\ b \leftarrow \neg\mathbf{K}a;\ \leftarrow \neg\mathbf{K}a, \neg\mathbf{K}\neg a;\ \leftarrow \neg\mathbf{K}a, \neg\mathbf{K}c;\ \leftarrow \neg\mathbf{K}a, \neg\mathbf{K}b, \mathbf{K}c;\}$*, i.e., rules of* Π *using* a, b, c*; for* t_2 *this program is* $\{c \leftarrow \neg d;\ d \leftarrow \neg c;\ c \leftarrow \neg\mathbf{K}d;\ d \leftarrow \neg\mathbf{K}c\}$*, comprising rules over atoms* c, d*. For precise definitions ensuring unique nested bag programs, we refer to [Besin* et al.*, 2021]. This example extends [Besin* et al.*, 2021, Ex. 7] and fixes a typo.*

Nested Dynamic Programming for ELPs. The concept of our algorithm is called *nested dynamic programming* and relies on computing abstractions in order to keep the width of the obtained TD manageable and to separate solving effort between DP and nested solving. Example 6 above shows the smaller (nested) bag programs obtained by using abstractions and nested primal graphs, which basically shows one step of our algorithm. For the outlined nested bag programs, our algorithm recursively computes abstractions and the nested primal graph, thereby providing a way to maintain tables needed for probabilistic reasoning via two counting problems of Definition 2. So in contrast to enumeration-based methods, our approach efficiently counts by materializing and combining solution parts (and not whole solutions).

For hard combinatorial subproblems that turn up during nesting, where the counts are not needed or not many solutions are expected, we delegate the task to standard (CDCL-based) solvers. Similarly, if a certain nesting depth is reached, further nesting and decomposing might not be beneficial, so subproblems are also delegated to standard solvers, thereby efficiently combining DP and search-based solving. For details we refer to the long version [Besin *et al.*, 2021].

5 Implementation & Experiments

We implemented our algorithm, resulting in the solver `nestelp`[2], which is written in Python3. It is based on the system `nesthdb` that was presented for variants of model counting [Hecher *et al.*, 2020b]. For manipulating tables during DP, `nestelp` uses the open source database Postgres 12,

[2] `nestelp` is readily available at github.com/viktorbesin/nestelp.

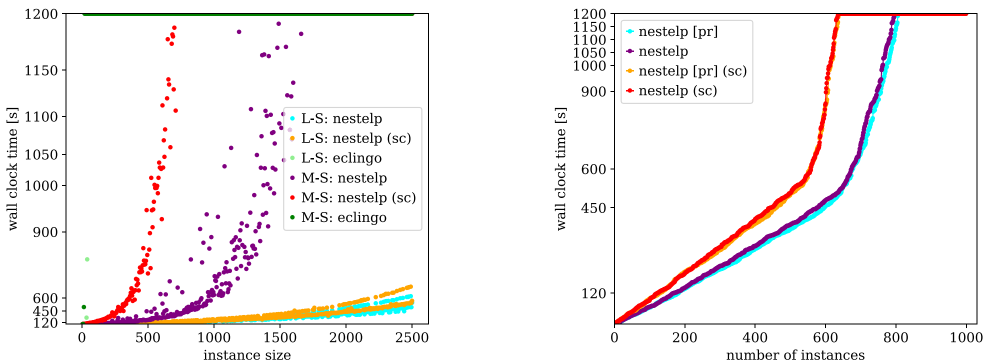

Figure 2: (Left): Line plot of runtimes for L-S and M-S; instances are ordered ascendingly according to instance size. (Right): Cactus plot for L-S and M-S (probabilistic reasoning); x-axis shows the number of instances; y-axis depicts runtime ascendingly sorted per solver.

which supports instant parallelization and was run on a tmpfs-ramdisk as intended by `nesthdb`. In order to compute TDs we use `htd` [Abseher *et al.*, 2017], which for every instance outputs TDs of decent widths in a runtime below some seconds. For solving decision problems of (epistemic) logic programs we utilize `clingo` 5.4 (`eclingo` 0.2). For finding good abstractions, we employ a logic program (available on github) using built-in optimization of `clingo`, where we take the best results after running at most 35 seconds. Our implementation supports both world view *counting* as given in Definition 1 as well as *probabilistic* world view acceptance of Definition 2.

In our benchmarks we compare wall clock runtime of `nestelp` and `eclingo` [Cabalar *et al.*, 2020], where a timeout is considered to occur after 1200 seconds and each solver was granted 16GB of main memory (RAM) per run. We restricted our solver to 12 physical cores. In *single core mode (sc)* of `nestelp`, only one physical core was used, allowing us to compare the performance with other single-core solvers.

We consider the following existing and generated instances.

Classic-Scholarship. As in previous works [Cabalar *et al.*, 2020], this is a set of 25 non-ground ELP programs encoding the Scholarship Eligibility problem [Gelfond, 1991] for 1–25 students, where all entities are independent from each other. If a students eligibility is not determined by the plain logic rules, an epistemic rule implies the interview of the student.

Yale-Shooting. This is a set of 12 non-ground ELP programs [Cabalar *et al.*, 2020] for the Yale Shooting problem [Hanks and Mcdermott, 1986]. For each instance the initial state, i.e., if a gun is initially loaded or not, is unknown.

Large-Scholarship (L-S). While classic-scholarship is limited to 25 instances, large-scholarship can be configured for an arbitrary number of students. We implement a generator and use existing instances to initialize more students. This set consists of 500 instances ranging from 5 to 2500 students.

Many-Scholarship (M-S). In contrast to classic-scholarship, where all students are in one unique WV, here a students eligibility is ranked (low or high chances), yielding many WVs per student. This set has 500 instances.

5.1 Experimental Results

For the classic-scholarship and yale-shooting problems, we observed `nestelp` keeping up with a traditional solver like `eclingo`, but, as expected, `nestelp` introduces additional overhead by the creation of tables and the general build-up for dynamic programming. Small instances do not benefit from that process, that is why we expected such results. However, the number of solved instances is the same for both systems.

The line plot in Figure 2 (left) shows an outstanding performance of `nestelp` for instances L-S and even M-S. Both instance sets allow their instances to be arranged into decompositions with low treewidth, representing instances where `nestelp` can exploit all its features. Further it can be seen that parallelism of `nestelp` has better performance than the single-core experiments (`nestelp` (sc)), indicating that there are enough independent nodes such that parallelism is beneficial. Even with the fair comparison to `eclingo`, the solver `nestelp` proves to handle large-scale instances well.

As it can be seen in the cactus plot in Figure 2 (right), the effort for probabilistic reasoning is very small in comparison to world view counting. Since `nestelp` intuitively only processes sub-calls where they are justified, i.e., only when there are any world views, there is little to no difference in the plot. We are certain that the visible differences are due to scattering factors like query optimization and CPU clocking. To summarize, the systems performance can be described quite competitively with a higher number of solved instances in similar or even shorter runtimes. Furthermore, consider that `nestelp` uses `eclingo` for sub-calls, so every revision of `eclingo` will improve our system as well.

6 Conclusion

In this work, we count world views of epistemic logic programs (ELPs), which is extended to probabilistic reasoning. We took up ideas of a theoretical algorithm that utilizes treewidth and turned this into an efficient solver. Our solver `nestelp` works on iteratively computing and refining (graph) abstractions of the ELP and counting world views over epistemic atoms of the abstract program. For counting and probabilistic reasoning, `nestelp` seems to scale well. We plan on further optimizing `nestelp`, which automatically improves with the availability of faster (E)LP solvers. We expect "tightness" notions, e.g., [Fandinno and Hecher, 2021; Hecher, 2022], to aid in improving performance. Recent complexity insights for treewidth [Fichte *et al.*, 2020; Fichte *et al.*, 2021b] renders our approach useful for other formalisms.

Acknowledgments

This research has been supported by the Vienna Science and Technology Fund (WWTF) grant ICT19-065, and by the Austrian Science Fund (FWF) grants P32830 and Y698.

References

[Abseher *et al.*, 2017] Michael Abseher, Nysret Musliu, and Stefan Woltran. htd – a free, open-source framework for (customized) tree decompositions and beyond. In *CPAIOR'17*, volume 10335 of *LNCS*, pages 376–386. Springer Verlag, 2017.

[Besin *et al.*, 2021] Viktor Besin, Markus Hecher, and Stefan Woltran. Utilizing treewidth for quantitative reasoning on epistemic logic programs. *Theory Pract. Log. Program.*, 21(5):575–592, 2021. Winner of the best paper award and best student paper award at ICLP'21.

[Bichler *et al.*, 2020] Manuel Bichler, Michael Morak, and Stefan Woltran. selp: A single-shot epistemic logic program solver. *Theory Pract. Log. Program.*, 20(4):435–455, 2020.

[Bliem *et al.*, 2016] Bernhard Bliem, Sebastian Ordyniak, and Stefan Woltran. Clique-width and directed width measures for answer-set programming. In *ECAI*, pages 1105–1113, 2016.

[Bodlaender and Kloks, 1996] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[Brewka *et al.*, 2011] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[Cabalar *et al.*, 2019] Pedro Cabalar, Jorge Fandinno, and Luis Fariñas del Cerro. Splitting epistemic logic programs. In *LPNMR*, pages 120–133, 2019.

[Cabalar *et al.*, 2020] Pedro Cabalar, Jorge Fandinno, Javier Garea, Javier Romero, and Torsten Schaub. eclingo : A solver for epistemic logic programs. *Theory Pract. Log. Program.*, 20(6):834–847, 2020.

[Diestel, 2012] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, 2012.

[Eiter and Gottlob, 1995] 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.

[Fandinno and Hecher, 2021] Jorge Fandinno and Markus Hecher. Treewidth-Aware Complexity in ASP: Not all Positive Cycles are Equally Hard. In *AAAI*, pages 6312–6320. AAAI Press, 2021.

[Fichte *et al.*, 2019] Johannes Klaus Fichte, Martin Kronegger, and Stefan Woltran. A multiparametric view on answer set programming. *Ann. Math. Artif. Intell.*, 86(1-3):121–147, 2019.

[Fichte *et al.*, 2020] Johannes Klaus Fichte, Markus Hecher, and Andreas Pfandler. Lower bounds for QBFs of bounded treewidth. In *LICS*, pages 410–424. ACM, 2020.

[Fichte *et al.*, 2021a] Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The Model Counting Competition 2020. *ACM Journal of Experimental Algorithmics*, 26, 2021.

[Fichte *et al.*, 2021b] Johannes Klaus Fichte, Markus Hecher, and Arne Meier. Knowledge-base degrees of inconsistency: Complexity and counting. In *AAAI*, pages 6349–6357. AAAI Press, 2021.

[Fierens *et al.*, 2015] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory Pract. Log. Program.*, 15(3):358–401, 2015.

[Ganian *et al.*, 2017] Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In *STACS*, pages 36:1–36:17, 2017.

[Gelfond, 1991] Michael Gelfond. Strong introspection. In *Proc. AAAI*, pages 386–391. AAAI Press / The MIT Press, 1991.

[Hanks and Mcdermott, 1986] Steve Hanks and Drew Mcdermott. Default reasoning, nonmonotonic logics, and the frame problem. In *AAAI*, pages 328–333. Morgan Kaufmann, 1986.

[Hecher *et al.*, 2020a] Markus Hecher, Michael Morak, and Stefan Woltran. Structural decompositions of epistemic logic programs. In *AAAI*, pages 2830–2837. AAAI Press, 2020.

[Hecher *et al.*, 2020b] Markus Hecher, Patrick Thier, and Stefan Woltran. Taming high treewidth with abstraction, nested dynamic programming, and database technology. In *SAT 2020*, volume 12178 of *LNCS*, pages 343–360. Springer, 2020.

[Hecher, 2022] Markus Hecher. Treewidth-aware reductions of normal ASP to SAT - Is normal ASP harder than SAT after all? *Artif. Intell.*, 304:103651, 2022.

[Jakl *et al.*, 2009] Michael Jakl, Reinhard Pichler, and Stefan Woltran. Answer-set programming with bounded treewidth. In *IJCAI*, pages 816–822, 2009.

[Kahl *et al.*, 2015] Patrick Thor Kahl, Richard Watson, Evgenii Balai, Michael Gelfond, and Yuanlin Zhang. The language of epistemic specifications (refined) including a prototype solver. *J. Log. Comput.*, 25, 2015.

[Lonc and Truszczynski, 2003] Zbigniew Lonc and Miroslaw Truszczynski. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Log.*, 4(1):91–119, 2003.

[Morak, 2019] Michael Morak. Epistemic logic programs: A different world view. In *ICLP*, pages 52–64, 2019.

[Saribatur and Eiter, 2021] Zeynep Gözen Saribatur and Thomas Eiter. Omission-based abstraction for answer set programs. *Theory Pract. Log. Program.*, 21(2):145–195, 2021.

[Shen and Eiter, 2016] Yi-Dong Shen and Thomas Eiter. Evaluating epistemic negation in answer set programming. *Artif. Intell.*, 237:115–135, 2016.

[Son *et al.*, 2017] Tran Cao Son, Tiep Le, Patrick Thor Kahl, and Anthony P. Leclerc. On computing world views of epistemic logic programs. In *IJCAI*, pages 1269–1275, 2017.

[Truszczynski, 2011] Miroslaw Truszczynski. Revisiting epistemic specifications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *LNCS*, pages 315–333. Springer Verlag, 2011.