TPLP **21** (5): 575–592, 2021. © The Author(s), 2021. Published by Cambridge University Press 575
doi:10.1017/S1471068421000399

Utilizing Treewidth for Quantitative Reasoning on Epistemic Logic Programs

VIKTOR BESIN
TU Wien, Vienna, Austria
(*e-mail:* vbesin@dbai.tuwien.ac.at)

MARKUS HECHER
TU Wien, Vienna, Austria
(*e-mail:* hecher@dbai.tuwien.ac.at)

STEFAN WOLTRAN
TU Wien, Vienna, Austria
(*e-mail:* woltran@dbai.tuwien.ac.at)

submitted 9 August 2021; revised 25 August 2021; accepted 27 August 2021

Abstract

Extending the popular answer set programming 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, that is, 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 take a next step and contribute to epistemic logic programming in two ways: First, we 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 that is capable of efficiently solving the underlying counting problems required to answer such quantitative reasoning problems. Our system exploits the graph-based measure treewidth and works by iteratively finding and refining (graph) abstractions of an ELP program. On top of these abstractions, we apply dynamic programming that is combined with utilizing existing search-based solvers like (e)clingo for hard combinatorial subproblems that appear during solving. It turns out that our approach is competitive with existing systems that were introduced recently.

KEYWORDS: epistemic logic programming, treewidth, tree decompositions, abstractions, hybrid solving, nested dynamic programming

1 Introduction

Answer set programming (ASP) 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, for example, the work of Brewka *et al.* 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 the use of LPs for a broad use in both academia and industry was the development of efficient solvers. However, while the ASP framework is quite 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*, that is, 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 thus are considered significantly harder than reasoning in 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 likelihood of being contained in an arbitrary world view. To the best of our knowledge, a few works on quantitative 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 quantitative 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 ELPs, 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 underlie the quantitative reasoning approach.

Our approach works on abstractions of the internal (graph) structure of ELPs; that is, we take the *primal graph*[1] of an ELP and contract certain paths between nodes referring to epistemic literals. On this graph we implicitly utilize the measure *treewidth*, which aims at measuring the tree-likeness of a given graph. The measure 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 accordingly. 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

[1] Basically, the primal graph of an (E)LP comprises of the atoms of the program, where two atoms are adjoined by an edge whenever these two atoms appear together in at least one rule.

such that still structural information in the form of TDs 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. Both additions combined, together with the guidance of abstract (implicit) structure of ELPs, allows us to efficiently evaluate ELPs.

Contributions. More concretely, we establish the following.

1. We motivate the problem of *world view counting*. This then leads to *probabilistic world view acceptance*, which accepts certain literals based on a quantitative argument concerning the proportion of world views agreeing with those literals.
2. Rooted in the theoretical investigation of Hecher *et al.* Hecher *et al.* (2020), we take up this idea and design an improved algorithm for evaluating ELPs by means of treewidth. Our algorithm lifts nested dynamic programming (Hecher *et al.* 020b) from satisfiability to logic programming, where treewidth is utilized on subsequently refined abstractions.
3. Finally, we present a system that implements this algorithm for quantitative reasoning. It turns out that the system is competitive and scales well for typical benchmarks.

Related Work. Treewidth was already utilized for the evaluation of standard LPs (Jakl *et al.* 2009; Hecher 2020). The concept of using abstractions was stipulated before as well, but in a different context (Hecher *et al.* 020b) or with the purpose of establishing theoretical results (Ganian *et al.* 2017). However, we improved an existing algorithm (Hecher *et al.* 2020) and to the best of our knowledge, our solver is the first implementation of solving ELPs that is guided by TDs. While the solver selp (Bichler *et al.* 2020) uses decompositions for breaking large rules into smaller ones, the solving itself is not guided by TDs. Also studies for measures different from treewidth have been conducted in the ASP domain (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). Let k, m, n be non-negative integers such that $k \leq m \leq n$ and $a_1, \ldots, a_n$ be distinct propositional atoms. Moreover, we refer by *literal* to an atom or the negation thereof. A *program* P is a set of *rules* of the form $a_1 \vee \cdots \vee a_k \leftarrow a_{k+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$.

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 sets of *atoms* occurring in a rule r or in a program P by $ats(r) := H_r \cup B^+_r \cup B^-_r$ and $ats(\mathsf{P}) := \bigcup_{r \in \mathsf{P}} ats(r)$. An *interpretation* $I \subseteq ats(\mathsf{P})$ is a set of atoms. I *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. We refer to the set of answer sets of a given

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

program P by $AS(\mathsf{P})$. The problem of deciding whether a program has an answer set, that is, whether $AS(\mathsf{P}) \neq \emptyset$, is Σ_2^P-complete (Eiter and Gottlob 1995).

Example 1
Consider the program $\mathsf{P} := \{\overbrace{\{a \vee b\}}^{r_1}, \overbrace{\{c \leftarrow \neg d\}}^{r_2}, \overbrace{\{d \leftarrow \neg c\}}^{r_3}\}$. The set $AS(\mathsf{P})$, denoting the answer sets for the LP P, consists of $\{a, c\}$, $\{a, d\}$, $\{b, c\}$ and $\{b, d\}$.

Tree Decompositions and Treewidth. We assume that graphs are undirected, simple, and free of self-loops. Let $G = (V, E)$ be a graph and $U \subseteq V$ be a set of vertices. Then, $G - U := (V \setminus U, \{e \in E \mid e \cap U = \emptyset\})$ is the graph obtained from removing U from G. Further, U is a *connected component* of a graph $G' = (V', E')$ if $U \subseteq V'$, U is *connected* and $U = \{u' \mid u \in U, \{u, u'\} \in E'\}$.

Let $G = (V, E)$ be a graph, T a rooted tree with *root node* $\mathrm{root}(T)$, 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 $\mathcal{T} = (T, \chi)$ is called a *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 $\chi(r) \cap \chi(t) \subseteq \chi(s)$. Intuitively, a TD allows to solve problems on a graph by analyzing parts of the graph and combining solutions to these accordingly. In order to simplify presentation, restricted node types and decompositions are oftentimes used, which are given as follows. For a node t of T, we say that $\mathrm{type}(t)$ is leaf if t has no children and $\chi(t) = \emptyset$; join if t has children t' and t'' with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; intr ("introduce") if t has a single child t', $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; rem ("removal") if t has a single child t', $\chi(t') \supseteq \chi(t)$ and $|\chi(t')| = |\chi(t)| + 1$. If for every node $t \in T$, $\mathrm{type}(t) \in \{\mathsf{leaf}, \mathsf{join}, \mathsf{intr}, \mathsf{rem}\}$, then (T, χ) is called *nice*. For every TD, one can compute a nice TD in polynomial time (Bodlaender and Kloks 1996) without increasing the width by adding intermediate (auxiliary) nodes accordingly. The *width* of a TD is defined as the cardinality of its largest bag minus one. The *treewidth* of a graph G, denoted by $tw(G)$, is the minimum width over all TDs of G. Note that if G is a tree, then $tw(G) = 1$.

3 Counting and reasoning for epistemic programs

Epistemic Logic Programming. An *epistemic literal* is a formula $\mathbf{not}\ell$, where ℓ is a literal and $\mathbf{not}$ is the epistemic negation operator. Let k, m, j, n be non-negative integers such that $k \leq m \leq j \leq n$ and $a_1, \ldots, a_n$ be distinct propositional atoms. An *ELP* is a set Π of *ELP rules* of the form $a_1 \vee \cdots \vee a_k \leftarrow \ell_{k+1}, \ldots, \ell_m, \xi_{m+1}, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n$, where each ℓ_i with $k + 1 \leq i \leq m$ is a literal over atom a_i, and each ξ_i with $m + 1 \leq i \leq n$ is an epistemic literal of the form $\mathbf{not}\ell_i$, where ℓ_i is a literal over atom a_i. Then, $ats(r) := \{a_1, \ldots, a_n\}$ denotes the set of atoms occurring in an ELP rule r, $\mathsf{e}\text{-}ats(r) := \{a_{m+1}, \ldots, a_n\}$ denotes the set of *epistemic atoms*, that is, those used in epistemic literals of r, and $\mathsf{a}\text{-}ats(r) := ats(r) \setminus \mathsf{e}\text{-}ats(r)$ refers to the *non-epistemic* atoms of r. We call r *purely epistemic* if $\mathsf{a}\text{-}ats(r) = \emptyset$. These notions naturally extend to programs. In a rule we sometimes write $\mathbf{K}\ell$ and $\mathbf{M}\ell$ for a literal ℓ, which refers to the expressions $\neg\mathbf{not}\ell$ and $\mathbf{not}\neg\ell$, respectively.

Given an ELP Π, a *world view interpretation (WVI)* I for Π is a consistent set I of literals over a set $A \subseteq ats(\Pi)$ of atoms, that is, $I \subseteq \{a, \neg a \mid a \in A\}$ such that there

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

is no $a \in A$ with $\{a, \neg a\} \subseteq I$. Intuitively, every $\ell \in I$ is considered as "known" and every $a \in A$ with $\{a, \neg a\} \cap I = \emptyset$ is treated as "possible". We denote the WVI over a set $X \subseteq ats(\Pi)$ of atoms obtained by restricting I to $Y = (A \cap X)$ by $I_{|X} := I \cap \{a, \neg a \mid a \in Y\}$. Next, we define compatibility with a set of interpretations.

Definition 1 (*WVI Compatibility*)
Let $\mathcal{I}$ be a set of interpretations over a set A of atoms. Then, a WVI I is *compatible* with $\mathcal{I}$ if:

1. $\mathcal{I} \neq \emptyset$;
2. for each atom $a \in I$, it holds that for each $J \in \mathcal{I}$, $a \in J$;
3. for each $\neg a \in I$, we have for each $J \in \mathcal{I}$, $a \notin J$;
4. for each atom $a \in A$ with $\{a, \neg a\} \cap I = \emptyset$, there are $J, J' \in \mathcal{I}$, such that $a \in J$, but $a \notin J'$.

While there are many different semantics (Gelfond 1991; Truszczynski 2011; Kahl *et al.* 2015; Shen and Eiter 2016), we follow the approach of Gelfond (Gelfond 1991), syntactically denoted according to recent work (Morak 2019). The *epistemic reduct* (Gelfond 1991) of program Π w.r.t. a WVI I over A, denoted Π^I, is defined as $\Pi^I = \{r^I \mid r \in \Pi\}$ where r^I denotes rule r where each epistemic literal $\mathbf{not}\ell$, whose atom is also in A, is replaced by $\bot$ if $\ell \in I$, and by $\top$ otherwise. Note that Π^I is a plain LP with all occurrences of epistemic negation removed. Now, a WVI I over $ats(\Pi)$ is a *world view (WV)* of Π iff I is compatible with the set $AS(\Pi^I)$. Without loss of generality we only consider ELPs Π, where every epistemic atom appears non-epistemically, that is, $\mathsf{e}\text{-}ats(\Pi) = \mathsf{a}\text{-}ats(\Pi^{\mathsf{e}\text{-}ats(\Pi)})$. We refer by $\Pi \sqcup I$ to the ELP $\Pi \cup \{\leftarrow \neg\mathbf{K}\ell \mid \ell \in I\} \cup \{\leftarrow \neg\mathbf{M}a; \leftarrow \neg\mathbf{M}\neg a \mid a \in A, a \notin I, \neg a \notin I\}$ used for verifying whether I can be extended to a WV. The set of WVs of an ELP Π is denoted $WVS(\Pi)$. One of the reasoning tasks for ELPs is *world view existence* deciding for an ELP Π whether $WVS(\Pi) \neq \emptyset$. This problem is known to be 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 defined as in Example 1, that is, the ELP Π depicts an epistemic extension of the plain LP P. For simplicity, let the rules be numbered equally from r_1 to r_{12}. When constructing a WVI I over $\mathsf{e}\text{-}ats(\Pi)$ one guesses for each atom $a \in \mathsf{e}\text{-}ats(\Pi)$ either (1) $a \in I$, (2) $\neg a \in I$ or (3) $\{a, \neg a\} \cap I = \emptyset$ as described earlier, that is, for the three atoms in $\mathsf{e}\text{-}ats(\Pi)$ we obtain 3^4 possibilities. Each WVI I can be checked with the corresponding epistemic reduct Π^I by verifying Definition 1 for $AS(\Pi^I)$.

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 indeed a plain LP, since by semantics of LPs, 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, that is, $WVS(\Pi) = \{I_1, I_2, I_3\}$.

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.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

Definition 2 (*World View Counting*)
Let Π be an ELP and Q be a WVI, called *query*, over atoms $ats(\Pi)$. Then, the problem $\#\mathrm{ELP}(\Pi, Q)$ asks to count the number of world views W with $Q \cap ats(\Pi) \subseteq W$ and $\{a \mid \neg a \in Q\} \cap W = \emptyset$.

As a special case, where $Q = \emptyset$, a problem instance $\#\mathrm{ELP}(\Pi, \emptyset)$ amounts to counting world views. Interestingly, the problem can be used to reason about the likelihood of an atom or a set of atoms being contained in an arbitrary world view, defined as follows.

Definition 3 (*Probability of World View Acceptance*)
Let Π be an ELP and Q be a WVI over $ats(\Pi)$. 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 $\#\mathrm{ELP}(\Pi, Q) = 2$ naturally 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.

For Definitions 2 and 3, we only consider WVIs over epistemic atoms to simplify presentation.[2]

4 Quantitative reasoning for ELPs via dynamic programming

Next, we discuss core ideas of dynamic programming for the evaluation of ELPs. We demonstrate this technique in Section 4.1 on a problem for ELPs that is much simpler than computing world views. Then, we extend this technique to nested dynamic programming in order to count world views in Section 4.2, which finally leads to probabilistic reasoning.

4.1 Basics of dynamic programming

Algorithms that utilize treewidth for solving a problem in linear time typically proceed by *dynamic programming (DP)* along the 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. A *table* τ_t is a set of rows, where a *row* $u \in \tau_t$ is a sequence or tuple of fixed length. These tables are derived by an algorithm, which we therefore call *table algorithm* $\mathbb{A}$. The actual length, content, and meaning of the rows depend on the algorithm $\mathbb{A}$ that derives tables.

The DP approach for solving problems of an ELP relies on a table algorithm $\mathbb{A}$ and consists of the following four steps:

[2] This is not a hard restriction that could be circumvented for a non-epistemic atom a, for example, via constraint $\leftarrow \neg\mathbf{K}a, \mathbf{K}a$.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

Prepare: Construct a *graph representation* G of the given ELP Π.

Decompose: Compute a tree decomposition (T, χ) of G, which can be obtained by using efficient heuristics (Abseher *et al.* 2017).

Compute: Execute table algorithm $\mathbb{A}$ for every node t of T in post-order, which returns the corresponding table for t. Algorithm $\mathbb{A}$ takes as input the corresponding bag $\chi(t)$, the assigned instance Π_t for node t, as well as the child tables previously computed during the post-order traversal for child nodes of t in T, and outputs a table τ_t.

Output: Print the solution by interpreting the table for root $n = \text{root}(T)$ of T.

For simplicity and the ease of presentation, the table algorithms presented in this work are *specified for nice TDs* due to clear case distinctions depending on type(t). However, the implemented architecture *does not* depend on certain normal forms of TDs. So, our approach works independently of whether such a TD is nice or not, since the different cases can be combined programmatically and TD nodes of any interleaved (combined) type can be processed.

Next, we briefly present a table algorithm for computing *plausible WVIs* of an ELP Π, which is a WVI I over $ats(\Pi)$ such that $AS(\{r \in \Pi \mid \mathsf{a}\text{-}ats(r) = \emptyset\}^I) \neq \emptyset$, denoted by $I \models_{\mathsf{p}} \Pi$. Observe that every WV of Π is always plausible as well. While counting plausible WVI serves the purpose of demonstrating and explaining dynamic programming, interestingly it is actually a #P-complete problem.

Proposition 1 (*Complexity of Counting Plausible WVIs*)
The problem of counting for a given ELP Π the number of plausible WVIs is #P-complete.

Proof (Sketch)
For membership, observe that one can guess a WVI I and then check whether $I \models_{\mathsf{p}} \Pi$ in polynomial time. Hardness is by reducing from #SAT, where one aims for counting the number of models of a 3-CNF formula $F = \{c_1, \ldots, c_l\}$. We construct an ELP Π that contains for every variable v of F a rule $\leftarrow \neg\mathbf{K}v, \neg\mathbf{K}\neg v$ and for every clause $c_i = \ell_1 \vee \ell_2 \vee \ell_3$ of F a rule $\leftarrow \neg\mathbf{K}\ell_1, \neg\mathbf{K}\ell_2, \neg\mathbf{K}\ell_3$. Then, the number of plausible WVIs of Π precisely captures the number of models of F. □

Before we discuss a table algorithm for counting plausible WVIs, we first require a *graph representation.* To this end, we employ the *epistemic primal graph* E_Π of an ELP Π, whose vertices stem only from the epistemic atoms $\mathsf{e}\text{-}ats(\Pi)$ and there is an edge between two vertices whenever the corresponding epistemic atoms appear together in a common purely epistemic rule of Π. Formally[3], we let $E_\Pi = (\mathsf{e}\text{-}ats(\Pi)^{\mathsf{e}}, E)$ with E being $\{\{a^{\mathsf{e}}, b^{\mathsf{e}}\} \mid r \in \Pi, \mathsf{a}\text{-}ats(r) = \emptyset, \{a, b\} \subseteq \mathsf{e}\text{-}ats(r)\}$. Now, let $\mathcal{T} = (T, \chi)$ be a TD of the epistemic primal graph E_Π and t be a node of T. Then, the *epistemic bag program* for t is given by $\Pi_t := \{r \in \Pi \mid \mathsf{a}\text{-}ats(r) = \emptyset, \mathsf{e}\text{-}ats(r)^{\mathsf{e}} \subseteq \chi(t)\}$. This allows us to refer to the *epistemic bag program up to* t by $\Pi_{\leq t} := \bigcup_{t' \text{ is a descendant node of } t \text{ in } T} \Pi_{t'} \cup \Pi_t$, which is the union over all epistemic bag programs for nodes below t in T Consequently, the epistemic bag program $\Pi_{\leq \text{root}(T)}$ up to the root corresponds to Π.

[3] For a set X of elements, we use the shortcuts $X^{\mathsf{e}} := \{x^{\mathsf{e}} \mid x \in X\}$.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

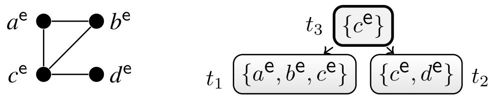


Fig. 1. Epistemic primal graph E_Π (left) of Π from Example 2 and a TD $\mathcal{T}$ (right) of E_Π.

Listing 1: Table algorithm $\#\mathbb{PWV}(\chi_t, \Pi_t, \langle \tau_1, \ldots, \tau_\ell \rangle)$ for Counting Plausible WVIs.

In: Node t, bag χ_t, epistemic bag program Π_t, and child tables $\langle \tau_1, \ldots, \tau_\ell \rangle$ of t.
Out: Table τ_t.

1 **if** $\text{type}(t) = \textit{leaf}$ **then**
2 | $\tau_t \leftarrow \{\langle \emptyset, 1 \rangle\}$
3 **else if** $\text{type}(t) = \textit{intr}$ *and* $a^{\mathsf{e}} \in \chi_t$ *is introduced* **then**
4 | $\tau_t \leftarrow \{\langle J, c \rangle \mid \langle I, c \rangle \in \tau_1, J \in \{I, I \cup \{a\}, I \cup \{\neg a\}\}, J \models_{\mathsf{p}} \Pi_t\}$
5 **else if** $\text{type}(t) = \textit{rem}$ *and* $a^{\mathsf{e}} \notin \chi_t$ *is removed* **then**
6 | $\tau_t \leftarrow \{\langle I', \sum_{\langle J, c' \rangle \in \tau_1 : I' \subseteq J} c' \rangle \mid \langle I, c \rangle \in \tau_1, I' = I \setminus \{a, \neg a\}\}$
7 **else if** $\text{type}(t) = \textit{join}$ **then**
8 | $\tau_t \leftarrow \{\langle I, c_1 \cdot c_2 \rangle \mid \langle I, c_1 \rangle \in \tau_1, \langle I, c_2 \rangle \in \tau_2\}$
9 **return** τ_t

Example 4

Figure 1 depicts the epistemic primal graph E_Π for Π as defined in Example 2 as well as one corresponding TD $\mathcal{T}$ of E_Π of width 2. Further, consider the epistemic bag programs $\Pi_{t_1} = \{r_8, r_9, r_{10}, r_{11}\}$, $\Pi_{t_2} = \{r_{12}\}$ and $\Pi_{t_3} = \emptyset$. Note that by definition of Π_t only rules solely built from $\mathsf{e}\text{-}ats(\Pi)$, that is, only purely epistemic rules are being considered. Observe that for the root node t_3 we have $\Pi_{\leq t_3} = \Pi$.

Listing 1 depicts a table algorithm $\#\mathbb{PWV}$ for counting plausible WVIs. Observe that it thereby suffices to compute WVIs over *epistemic atoms*, as such a WVI already uniquely identifies one WVI over all atoms. Then, algorithm $\#\mathbb{PWV}$ stores rows of the form $\langle I, c \rangle$, where I is a WVI over $\chi(t)$ and c is an integer (counter) referring to the number of plausible WVIs of the epistemic bag program up to t, that when restricted to $\chi(t)$ coincide with I. Consequently, for decompositions whose roots have empty bags, the counter of a stored row refers to the number of plausible world views of Π. As already mentioned above, for the ease of presentation, table algorithm $\#\mathbb{PWV}$ is given for nice TDs, that is, in Listing 1 we distinguish the four different cases of nice TDs. So, if node t is a leaf node, cf. Line 2, the only row matching these conditions is $\langle \emptyset, 0 \rangle$. Then, whenever a vertex a^{e} is introduced in a node t, Line 4 guesses all three possibilities for extending an existing WVI I by atom a and checks that the resulting WVI J ensures Π_t. For nodes t with $\text{type}(t) = \mathsf{rem}$, where we remove a^{e}, Line 6 removes the mapping of a in any existing WVI I and sums up the counters of collapsing WVIs, that is, where all atoms guessed in I' match, accordingly. Finally for a join node t, we intuitively keep only rows, whose WVIs are in all child nodes tables, and counters of those rows need to be multiplied. Note that the clear case distinction between node types of nice TDs simplifies the processing of child tables, for example, when processing a node of type join, since there are at most two child nodes.

Example 5

Considering program Π from Example 2, we obtain three world views as described earlier. Table algorithm $\#\mathbb{PWV}$ can be used to restrict the possible WVIs. Figure 2 shows a nice

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

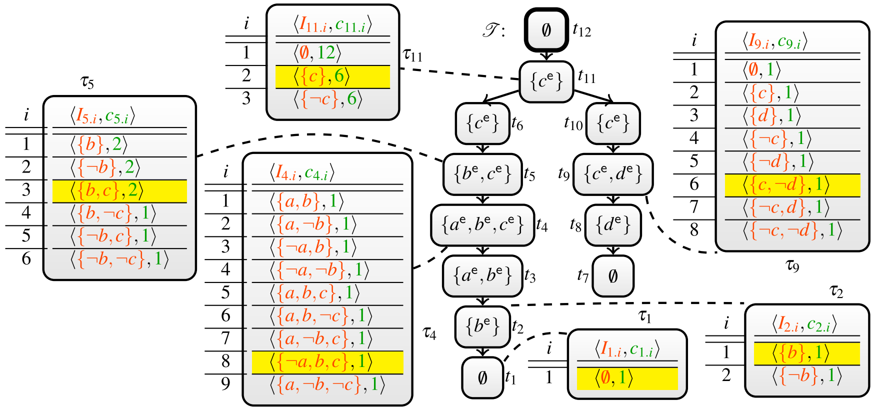


Fig. 2. A nice TD $\mathcal{T}$ of the epistemic primal graph E_Π of program Π from Example 2 as well as selected tables obtained by $\#\mathbb{PWV}$ on Π and $\mathcal{T}$.

TD $\mathcal{T} = (T, \chi)$ of E_Π and a selection of the tables $\tau_1, \ldots, \tau_{12}$, which illustrate computation results obtained during post-order traversal of $\mathcal{T}$ by $\#\mathbb{PWV}$.

Table $\tau_1 = \{\langle\emptyset, 1\rangle\}$ as per definition for $\mathrm{type}(t_1) = \mathsf{leaf}$. Since $\mathrm{type}(t_2) = \mathsf{intr}$, we construct table τ_2 from τ_1 by taking $I_{1.i}$, $I_{1.i} \cup \{b\}$ and $I_{1.i} \cup \{\neg b\}$ for each $\langle I_{1.i}, c_{1.i}\rangle \in \tau_1$ (corresponding to a guess on b). Since $\mathsf{e}\text{-}ats(r_9) \subseteq \chi(t_2)$ we have $\Pi_{t_2} = \{r_9\}$ for t_2 as described in Example 4. In consequence, for each $I_{2.i}$ of table τ_2, we have $I_{2.i} \models \{r_9\}$ since $\mathbb{PWV}$ enforces satisfiability of Π_t in node t. Then, t_3 introduces a^{e} and t_4 introduces c^{e} in similar fashion while satisfying the appropriate epistemic bag programs $\Pi_{t_3} = \{r_8\}$ and $\Pi_{t_4} = \{r_{10}, r_{11}\}$. We derive tables τ_7 to τ_9 similarly. Since $\mathrm{type}(t_5) = \mathsf{rem}$, we remove atom a from all elements in τ_4 to construct τ_5. As described earlier, this is accomplished by summing up the counters for matching WVIs when removing the atom a, for example, since the remaining, guessed atoms b^e and c^e are matching, counters for line 2 and 4 in table τ_4 are summed up, resulting in line 2 in table τ_5. Note that we have already seen all rules where a^{e} occurs and hence a^{e} can no longer affect witnesses during the remaining traversal. We similarly construct $\tau_6 = \{\langle\emptyset, 4\rangle, \langle\{c\}, 3\rangle, \langle\{\neg c\}, 2\rangle\}$ and $\tau_{10} = \{\langle\emptyset, 3\rangle, \langle\{c\}, 2\rangle, \langle\{\neg c\}, 3\rangle\}$. Since $\mathrm{type}(t_{11}) = \mathsf{join}$, we construct table τ_{11} by taking the intersection $\tau_6 \cap \tau_{10}$. Intuitively, this combines witnesses agreeing on c while multiplying the counters for matching guesses. Node t_{12} is again of type rem. By definition (primal graph and TDs) for every $r \in \Pi$, atoms $\mathsf{a}\text{-}ats(r)$ occur together in at least one common bag. Hence, $\Pi = \Pi_{t_{12}}$ and since $\tau_{12} = \{\langle\emptyset, 24\rangle\}$, we end up with 24 plausible WVIs of Π which we can construct from the tables. For example, we obtain the interpretation $\{\neg a, b, c, \neg d\} = I_{11.2} \cup I_{4.8} \cup I_{9.6}$, as highlighted in yellow.

4.2 Counting world views via nested dynamic programming

In order to extend DP for solving #ELP, we require a suitable graph representation that still allows for simple table algorithms. Let therefore Π be an ELP. Then, the *primal graph* G_Π uses atoms and epistemic atoms as vertices and it is defined by $G_\Pi := (\{a^\circ \mid a \in \circ\text{-}ats(\Pi), \circ \in \{\mathsf{a}, \mathsf{e}\}\}, E)$, where $E := \{\{a^\circ, b^\star\} \mid r \in \Pi, a \in \circ\text{-}ats(r), b \in \star\text{-}ats(r), \{\circ, \star\} \subseteq \{\mathsf{a}, \mathsf{e}\}\} \cup \{\{a^{\mathsf{a}}, a^{\mathsf{e}}\} \mid a \in \mathsf{e}\text{-}ats(\Pi)\}$. For our purposes, we require suitable abstractions of G_Π, given as follows. A *non-epistemic path* in G_Π is a

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

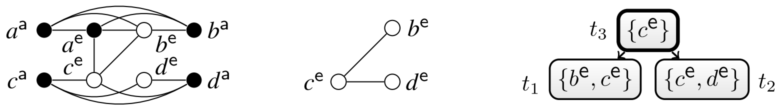


Fig. 3. Primal graph G_Π (left) of Π, the nested primal graph G_Π^A for $A = \{b, c, d\}$ (middle) and a TD $\mathcal{T}$ for the nested primal graph G_Π^A (right).

path of the form $a^e, v_1^a, \ldots, v_l^a, b^e$ with $l \geq 0$. The *nested primal graph* G_Π^A over a given set $A \subseteq \textsf{e}\text{-}ats(\Pi)$ of epistemic atoms is given by $G_\Pi^A := (A^e, E')$ with $E' := \{\{a^e, b^e\} \mid \{a, b\} \subseteq A,$ there is a non-epistemic path from a^e to b^e in $G_\Pi\}$.

Example 6

Recall program Π of Example 2. Figure 3 shows the primal graph G_Π for program Π. Given epistemic atoms $A = \{b, c, d\}$ the nested primal graph G_Π^A can be constructed with edges $\{b^e, c^e\}$ and $\{c^e, d^e\}$ through any of the non-epistemic paths between the two correlating vertices in G_Π.

Indeed, in this section we use the nested primal graph G_Π^A for applying DP in a nested fashion. There, the nested primal graph provides sufficient abstractions of the primal graph, where we count plausible WVIs over A, similar to Listing 1. These plausible WVIs over A are then subsequently extended and refined (to obtain world views), since in each node of a TD, one chooses again an abstraction A' that decides on remaining epistemic atoms until all epistemic atoms are considered. So, if in the beginning we decide that $A{=}\textsf{e}\text{-}ats(\Pi)$, we end up with full DP and zero nesting, whereas setting $A{=}\emptyset$ results in full nesting, that is, no DP. Before we discuss how to choose such a set A somewhere between these two extreme cases, we define how the ELP that is subject to nesting looks like. To formalize this, we assume a TD $\mathcal{T} = (T, \chi)$ of G_Π^A and say a set $U \subseteq ats(\Pi)$ of atoms is *compatible* with a node t of T, and vice versa, if

(I) there is a connected component C of graph $G_\Pi - A^e$ such that $U = \{a \mid \{a^e, a^a\} \cap C \neq \emptyset\}$;

(II) all neighbor vertices of C in G_Π that are in A^e, are contained in $\chi(t)$, that is, $\{a^e \mid a \in A, u \in U,$ there is a non-epistemic path from u^e to a^e in $G_\Pi\} \subseteq \chi(t)$.

If such a set $U \subseteq ats(\Pi)$ of atoms is compatible with a node of T, we say that U is a *compatible set*. By construction of the nested primal graph, any atom not in A is in at least one compatible set, but a compatible set could be compatible with several nodes of T. Hence, to enable nested evaluation, we ensure that each nesting atom is evaluated in one unique node t.

As a result, we formalize for every compatible set U a *unique* node t of T that is compatible with U, denoted by $\text{comp}(U) := t$. We denote the union of all compatible sets U with $\text{comp}(U) = t$, by *nested bag atoms* $A_t := \bigcup_{U:\text{comp}(U)=t} U$. Finally, the *nested bag program* Π_t^A for a node t of T, that is, the ELP subject to nesting, equals $\Pi_t^A := \{r \in \Pi \mid \textsf{a}\text{-}ats(r) \subseteq A_t, \textsf{e}\text{-}ats(r) \subseteq A_t \cup \{a \mid a^e \in \chi(t)\}\} \setminus \Pi_t$. Observe that the definition of nested bag programs ensures that any connected component U of $G_\Pi - A^e$ "appears" among nested bag atoms of some unique node of T. Consequently, for each atom $a \in ats(\Pi) \setminus A$ there is a *unique* node t such that $a \in ats(\Pi_t^A)$.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

Example 7

Considering program Π from Example 2 and the nested primal graph G^A_Π for $A = \{b, c, d\}$, Figure 3 shows a corresponding TD $\mathcal{T}$ for the nested primal graph G^A_Π. When removing vertices A^e from G_Π one can identify the two connected components $\{a^\mathsf{a}, b^\mathsf{a}, a^\mathsf{e}\}$ and $\{c^\mathsf{a}, d^\mathsf{a}\}$ each of which building a compatible set in the form of $U_1 := \{a, b\}$ uniquely compatible with node t_1 and $U_2 := \{c, d\}$ uniquely compatible with node t_2, that is, $\mathrm{comp}(U_1) = t_1$ and $\mathrm{comp}(U_2) = t_2$. Then nested bag programs $\Pi^A_{t_1} = \{r_1, r_4, r_5, r_8, r_9, r_{10}, r_{11}\}$ and $\Pi^A_{t_2} = \{r_2, r_3, r_6, r_7, r_{12}\}$ emerge from $A_{t_1} = \{a, b\}$ and $A_{t_2} = \{c, d\}$, respectively. Note that $\Pi^A_{t_3} = \emptyset$ because of $A_{t_3} = \emptyset$.

Nested dynamic programming for ELPs

Next, we discuss *nested dynamic programming (nested DP)* in order to count world views of an ELP Π. Thereby we aim at solving the more elaborated problem $\#\mathrm{ELP}(\Pi \sqcup W, \emptyset)$ for a WVI W over a set $X \subseteq \mathsf{a}\text{-}ats(\Pi)$ of atoms of Π. This problem amounts to counting the number of world views of Π that agree with W over atoms X. Hence, we consider a more fine-grained variant of counting world views that for the special case of $X = \emptyset$ actually coincides with $\#\mathrm{ELP}(\Pi, \emptyset)$ as stated in Definition 2.

Our algorithm for nested dynamic programming, called `NestELP`, is presented in Listing 2 and relies on the nested primal graph that is utilized in a nested fashion. Therefore, Algorithm `NestELP` takes as first argument an integer for the nesting depth, the ELP Π and the WVI W. Listing 2 consists of four separated blocks. The first block (Lines 1–4) comprises solving the base case where Π has no epistemic atoms, that is, no epistemic "decisions" are left for solving Π. There, if all atoms of X appear positively or negatively in W, we use two ASP solver calls to check Conditions (1) or (2)+(3) of Defnition 1, respectively. Otherwise all four conditions of Definition 1 are verified via one ELP solver call. The next block consists of Lines 5–Lines 7, which computes a TD $\mathcal{T}$ of the primal graph of Π (nested primal graph with $A = \mathsf{e}\text{-}ats(\Pi)$). Then this block utilizes standard ELP solvers in case $\mathrm{width}(\mathcal{T})$ is out of reach ($\mathrm{threshold}_{\mathrm{hybrid}}$) or nesting is already too deep ($\mathrm{threshold}_{\mathrm{depth}}$). If this is not the case and $\mathrm{width}(\mathcal{T})$ is insufficient for DP ($\mathrm{threshold}_{\mathrm{abstr}}$), the third block consisting of Lines 8–10 chooses a suitable abstraction A and computes a TD $\mathcal{T}$ of the nested primal graph G^A_Π. Finally, the last block comprises of the remaining lines of Listing 2, which performs DP on the TD $\mathcal{T}$ that is obtained either in Block 2 or Block 3 and returns the solution in Line 14. The actual recursion (nesting) is via table algorithm $\#\mathbb{ELP}$ that is used during DP in Line 13, discussed next.

The table algorithm $\#\mathbb{ELP}$ is given in Listing 3. Compared to Listing 1, we have two additional parameters, namely the nested bag program and WVI W. The main difference is in Line 5 of Listing 3, where an additional recursive call to `NestELP` is performed. This recursive call increases the depth and concerns about the nested bag program that is simplified by the current WVI J and aims at verifying WVI $W \cup J$ restricted to those atoms that appear also in non-epistemic atoms of a rule of the nested bag program. The other atoms not appearing in such a rule will be checked in the context of an other bag. Intuitively, the resulting count c' of the recursive call needs to be multiplied as it concerns different epistemic atoms, cf. Line 4 of Listing 3.

Listing 2: Algorithm $\texttt{NestELP}(\text{depth}, \Pi, W)$ for world view counting by means of nested DP.

In: Nesting depth ≥ 0, epistemic logic program Π, and a WVI W over a set $X \subseteq \mathsf{a}\text{-}ats(\Pi)$ of atoms.
Out: The number $\#\mathrm{ELP}(\Pi \sqcup W, \emptyset)$ of world views.

$A \leftarrow \mathsf{e}\text{-}ats(\Pi)$
if $A = \emptyset$ `/* No Epistemic Decisions left; Verify Decisions */` **then**
if $\{a \in X \mid a \notin W, \neg a \notin W\} = \emptyset$ **then return** $|AS(\Pi)| = 1$ *and* $|AS(\Pi \cup \{\leftarrow W\})| = 0$ `/* ASP */`
  else return $WVS(\Pi \sqcup W) \neq \emptyset$ `/* Verify via Standard ELP Solver */`
$\mathcal{T} = (T, \chi) \leftarrow \text{Decompose}(G_\Pi)$ `/* Decompose via Heuristics */`
if $\text{width}(\mathcal{T}) \geq \mathit{threshold}_{hybrid}$ *or* $\text{depth} \geq \mathit{threshold}_{depth}$ `/* Standard ELP Solver */` **then**
  return $\#\mathrm{ELP}(\Pi \sqcup W, \emptyset)$

if $\text{width}(\mathcal{T}) \geq \mathit{threshold}_{abstr}$ `/* Abstract & Decompose via Heuristics */` **then**
  $A \leftarrow \text{Choose-Abstraction}(A, \Pi)$
  $\mathcal{T} = (T, \chi) \leftarrow \text{Decompose}(G_\Pi^A)$

for iterate t *in* post-order(T) `/* Dynamic Programming */` **do**
  $\text{Child-Tabs} \leftarrow \langle \tau_{t_1}, \dots, \tau_{t_\ell} \rangle$ where $\text{children}(t) = \langle t_1, \dots, t_\ell \rangle$
  $\tau_t \leftarrow \#\mathbb{ELP}(\text{depth}, \chi(t), \Pi_t, \Pi_t^A, W, \text{Child-Tabs})$
return $\sum_{\langle I, c \rangle \in \tau_{\text{root}(T)}} c$ `/* Return Total Count */`

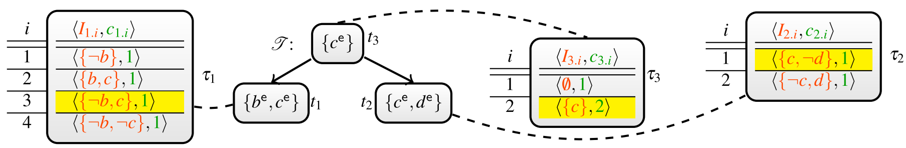


Fig. 4. A TD $\mathcal{T}$ of the nested primal graph G_Π^A of program Π from Example 2 for $A = \{b, c, d\}$ as well as selected tables obtained by $\#\mathbb{ELP}$ on Π and $\mathcal{T}$.

Example 8
Recall program Π, set A of epistemic atoms, TD $\mathcal{T}$ of nested primal graph G_Π^A and nested bag programs given in Example 7. Figure 4 illustrates computation results obtained during post-order traversal of $\mathcal{T}$ by $\#\mathbb{ELP}$. Notice that similar to $\#\mathbb{PWV}$ the algorithms enforces the entailment of Π_t for each guess, reducing the number of rules for the actual nested call, for example the nested call for node t_1 will only include rules $\{r_1, r_4, r_5, r_8, r_{10}, r_{11}\}$, c.f. Example 7. Further observe that while guessing introduced epistemic atoms as in node t_1 and t_2, the epistemic reduct is built over all guessed atoms, but the guess of c is only checked actively in node t_2 using epistemic constraints. Since joining the nodes naturally enforces agreeing assignments of c this is indirectly checked for t_1. Similar to Example 5, one can identify that epistemic program Π has three world views which can be reconstructing joining agreeing assignments of the tables in-order. For example, we obtain the (incomplete) world view $\{b, c, \neg d\} = I_{3.2} \cup I_{1.3} \cup I_{2.1}$, as highlighted in yellow.

Having established an algorithm for counting, we only briefly discuss how to extend the table algorithm of Listing 3 for *probabilistic world view acceptance* of a WVI (query) Q via Definition 3. To this end, instead of storing only a WVI and a counter, the rows of the tables of the obtained table algorithm $\mathbb{PELP}$ are of the form $\langle I, c, q \rangle$, where I is a WVI and c as well as q are counters. Thereby, I and c are maintained as before

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

Listing 3: Table algorithm $\#\mathbb{ELP}(\text{depth}, \chi_t, \Pi_t, \Pi_t^A, W, \langle \tau_1, \ldots, \tau_\ell \rangle)$ for Counting WVIs.

In: Nesting depth ≥ 0, bag χ_t, epistemic bag program Π_t, nested bag program Π_t^A, world view interpretation W, and sequence $\langle \tau_1, \ldots, \tau_\ell \rangle$ of child tables of t.

Out: Table τ_t.

1 **if** type(t) = *leaf* **then**
2 $\quad \tau_t \leftarrow \{\langle \emptyset, 1 \rangle\}$
3 **else if** type(t) = *intr and* $a^e \in \chi_t$ *is introduced* **then**
4 $\quad \tau_t \leftarrow \{\langle J, c' \rangle \mid \langle I, c \rangle \in \tau_1, J \in \{I, I \cup \{a\}, I \cup \{\neg a\}\}, J \models_{\mathsf{p}} \Pi_t,$
5 $\quad \mathsf{P} = (\Pi_t^A)^J, c' = c \cdot \mathtt{NestELP}(\text{depth}+1, \mathsf{P}, (W \cup J)_{|\mathsf{a}\text{-}ats(\mathsf{P})}), c' > 0\}$
6 **else if** type(t) = *rem and* $a^e \notin \chi_t$ *is removed* **then**
7 $\quad \tau_t \leftarrow \{\langle I', \sum_{\langle J, c' \rangle \in \tau_1 : I' \subseteq J} c' \rangle \mid \langle I, c \rangle \in \tau_1, I' = I \setminus \{a, \neg a\}\}$
8 **else if** type(t) = *join* **then**
9 $\quad \tau_t \leftarrow \{\langle I, c_1 \cdot c_2 \rangle \mid \langle I, c_1 \rangle \in \tau_1, \langle I, c_2 \rangle \in \tau_2\}$
10 **return** τ_t

and q is computed similarly to c, but in Line 5 the recursive call for obtaining q' involves the nested bag program extended by Q, that is, $\Pi_t^A \sqcup Q$. Then, instead of summing up counters c in Line 14 of Listing 2, these adapted tables computed by $\mathbb{PELP}$ explained above are used to sum up fractions $\frac{q}{c}$, which leads the desired result. Detailed algorithms for $\mathbb{PELP}$ and $\mathtt{NestELP}_{\mathbb{PELP}}$ are depicted in the supplementary material of this paper, cf. Listings 4 and 5.

5 Implementation & preliminary experiments

We implemented the algorithm `NestELP`, resulting in the solver `nestelp`[4], which is written in Python3. It is based on the system `nesthdb` that was presented for variants of model counting (Hecher *et al.* 020b). For manipulating tables during DP, `nestelp` uses the open source database Postgres 12, which supports instant parallelization and was run on a tmpfs-ramdisk as intended by `nesthdb`. In order to compute TDs (Lines 5 and 10 of Listing 2), 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 LPs in Line 3 we used `clingo` 5.4. For solving ELP problems in Lines 4 and 7, we utilized `eclingo` 0.2. Internally, we set $\text{threshold}_{\text{hybrid}} = 45$, $\text{threshold}_{\text{abstract}} = 8$ and allowed nesting once, which overall seemed to produce good results. However, these parameters are not the result of extensive performance tuning, but were chosen as initial values with the goal of balancing abstractions and hybrid (standard) solving. For finding good abstractions in Line 9, that is, searching for epistemic atoms when constructing the nested primal graph, we employ a L similar to `nesthdb`. Intuitively, we thereby aim for a preferably large set A of epistemic atoms such that the resulting graph N_Π^A is reasonably sparse. This is achieved heuristically by minimizing the number of edges of N_Π^A. To this end, we use built-in optimization of `clingo`, where we take the best results after running at most 35 seconds. For the concrete encodings, we refer to the online repository of `nestelp` as given above. Our implementation supports both world view *counting* as given in Definition 2 as well as *probabilistic* world view acceptance of Definition 3.

[4] The solver `nestelp` is open source and available at github.com/viktorbesin/nestelp.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

Benchmark setting

In order to draw conclusions about the efficiency of our implementation, we conducted a series of benchmarks. All our used benchmark instances, raw results and detailed data are available online at tinyurl.com/iclp21-nestelp. 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, which allows us to compare the performance with other single-core solvers. Benchmarks were conducted on a cluster consisting of 12 nodes. Each node of the cluster is equipped with two Intel Xeon E5-2650 CPUs and each of these 12 physical cores runs at 2.2 GHz clock speed that has access to 256 GB shared RAM. Results are gathered on Ubuntu 16.04.1 LTS OS that is powered on kernel 4.4.0-139. We disabled hyperthreading and used Python 3.7.6.

Benchmark instances

The following instances are considered from the literature and extended accordingly.

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 one to twenty-five 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) encoding the Yale Shooting problem (Hanks and Mcdermott 1986). With each instance the knowledge of the initial state, that is, if the gun is initially loaded or not, is incomplete.

Large-Scholarship (L-S). While classic-scholarship is limited to 25 instances, large-scholarship can be configured to a number of students, that is, a student-wise extension to classic-scholarship. As part of our testing, we implemented a generator for such instances, using existing instances to initialize more students. This set consists of 500 instances ranging from 5 to 2500 students.

Many-Scholarship (M-S). In comparison to classic-scholarship, where all students are part of one unique world view, many-scholarship extends the situation and aims for a more relaxed situation, where additionally a students eligibility is ranked with low or high chances. This often results in many world views per student. Our generator is implemented in a way such that both introduced instance sets are supported. Also this set consists of 500 instances.

Benchmark scenarios

We considered the following three scenarios in order to test the efficiency of `nestelp`.

- S1 Counting world views for the classical-scholarship as well as Yale-shooting instances.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press

S2 Counting world views for large-scale instances, thereby using large-scholarship and many-scholarship instances. For a fair comparison, we allow `eclingo` to decide WV existence.

S3 Probabilistic reasoning *[pr]* for large-scale instances. This scenario concerns probabilistic WV acceptance using also large-scholarship and many-scholarship instances.

Based on these scenarios, we state corresponding hypothesis that shall be verified in this section.

H1 `nestelp` is competitive for counting, although monolithic solvers like `eclingo` are faster.

H2 Our implementation `nestelp` is rather competitive for large-scale instances.

H3 Probabilistic reasoning comes almost for the same cost as counting in the solver `nestelp`.

Experimental results

The results for Scenario S1 in comparison with `eclingo` are summarized in the table of Figure 5. Overall it can be seen that `nestelp` can keep 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, as for S1, do not benefit from that process, that is why we expected such results. The number of solved instances is the same for both systems, overall agreeing with our Hypothesis H1. The line plot in Figure 5 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 its ability to handle large-scale instances well, as proposed in Hypothesis H2. As it can be seen in the cactus plot in Figure 6, the effort needed for probabilistic reasoning is very small in comparison to world view counting. Since

solver	max(width)	#solved (width range) 0-5	5-20	>20	Σ	time[h]
Classic-Scholarship: 25 instances						
eclingo	1.0	25	0	0	25	0.01
nestelp	1.0	25	0	0	25	0.01
nestelp (sc)	1.0	25	0	0	25	0.02
Yale-Shooting: 12 instances						
eclingo	61.0	2	3	3	8	1.34
nestelp	61.0	2	3	3	8	1.37
nestelp (sc)	61.0	2	3	3	8	1.40
Σ: 37 instances						
eclingo	61.0	27	3	3	33	1.35
nestelp	61.0	27	3	3	33	1.39
nestelp (sc)	61.0	27	3	3	33	1.42

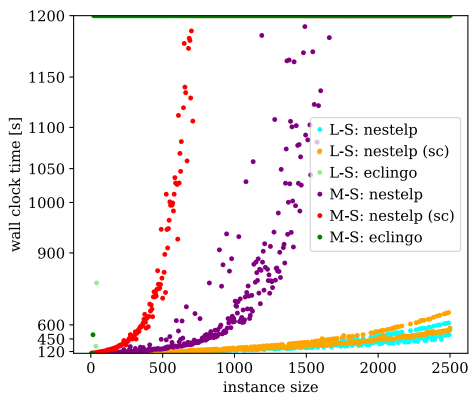
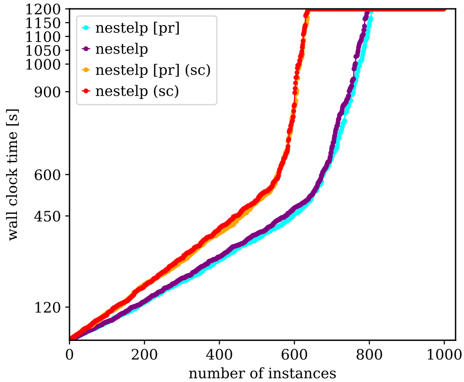


Fig. 5. Detailed results (left) over Scenario S1 showing maximal width of the primal graph among solved instances, solved instances over certain width ranges, as well as total runtime in hours, where timeouts count as 1200s. Line plot (right) of instances L-S and M-S for Scenario S2, where instances are ordered ascendingly according to instance size.

https://doi.org/10.1017/S1471068421000399 Published online by Cambridge University Press



solver	max_width	#fastest	#unique	#solved	time[h]
Large-Scholarship (L-S): 500 instances					
nestelp [pr]	1.0	275	0	500	35.72
nestelp	1.0	225	0	500	36.50
nestelp [pr] (sc)	1.0	4	0	500	39.46
nestelp (sc)	1.0	4	0	500	40.08
eclingo	1.0	5	0	8	164.32
Many-Scholarship (M-S): 500 instances					
nestelp [pr]	2.0	183	18	306	106.14
nestelp	2.0	132	9	296	109.19
nestelp [pr] (sc)	2.0	0	0	138	142.61
nestelp (sc)	2.0	0	0	135	143.31
eclingo	2.0	1	0	3	165.81
Σ: 1000 instances					
nestelp [pr]	2.0	458	18	806	141.87
nestelp	2.0	357	9	796	145.70
nestelp [pr] (sc)	2.0	4	0	638	182.07
nestelp (sc)	2.0	4	0	635	183.39
eclingo	2.0	6	0	11	330.13

Fig. 6. Scenario S3: Cactus plot (left), whose x-axis shows the number of instances; the y-axis depicts runtime sorted ascendingly for each solver individually. Detailed results (right).

`nestelp` intuitively only processes sub-calls where they are justified, that is, only when there are any world views, there is little to no difference in the plot. While agreeing with Hypothesis H3, we even believe 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, leading to the assumption that every revision of the base solver will improve our system too.

6 Conclusion

In this work we studied counting world views of ELPs and extended this further to probabilistic reasoning. We took up ideas of a theoretical algorithm that utilizes treewidth and progressively 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. Then, the count is subsequently improved by refining the abstraction in a nested fashion, for which we use our algorithm or existing (E)LP solvers. Specifically for counting and probabilistic reasoning, `nestelp` seems to scale well. For future work we plan on further optimizing this technique, which however automatically improves with the availability of faster solvers as those are the core engines in `nestelp`. Further, given recent insights on complexity results for treewidth (Fichte *et al.* 2020; 2021), the techniques developed and applied in this work could be also carried out for other formalisms like abstract argumentation or description logics.

Acknowledgements

This work has been supported by the Austrian Science Fund (FWF), Grants P32830 and Y698, as well as the Vienna Science and Technology Fund, Grant WWTF ICT19-065. We would like to thank the reviewers for their detailed and valuable comments. Part of the research was carried out while Hecher was visiting the Simons Institute for the Theory of Computing.

Supplementary material

To view supplementary material for this article, please visit http://dx.doi.org/10.1017/S1471068421000399.

References

Abseher, M., Musliu, N. and Woltran, S. 2017. htd – A free, open-source framework for (customized) tree decompositions and beyond. In *CPAIOR'17*. LNCS, vol. 10335. Springer Verlag, 376–386.

Bichler, M., Morak, M. and Woltran, S. 2020. selp: A single-shot epistemic logic program solver. *Theory and Practice of Logic Programming 20,* 4, 435–455.

Bliem, B., Ordyniak, S. and Woltran, S. 2016. Clique-width and directed width measures for answer-set programming. In *Proc. ECAI*. 1105–1113.

Bodlaender, H. L. and Kloks, T. 1996. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms 21,* 2, 358–402.

Brewka, G., Eiter, T. and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

Cabalar, P., Fandinno, J. and Fariñas del Cerro, L. 2019. Splitting epistemic logic programs. In *LPNMR*. 120–133.

Cabalar, P., Fandinno, J., Garea, J., Romero, J. and Schaub, T. 2020. eclingo : A solver for epistemic logic programs. *Theory and Practice of Logic Programming 20,* 6, 834–847.

Eiter, T. and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence 15,* 3–4, 289–323.

Fichte, J. K., Hecher, M. and Meier, A. 2021. Knowledge-base degrees of inconsistency: Complexity and counting. In *AAAI*. AAAI Press, 6349–6357.

Fichte, J. K., Hecher, M., and Pfandler, A. 2020. Lower bounds for QBFs of bounded treewidth. In *LICS*. ACM, 410–424.

Fichte, J. K., Kronegger, M. and Woltran, S. 2019. A multiparametric view on answer set programming. *Annals of Mathematics and Artificial Intelligence 86,* 1–3, 121–147.

Fierens, D., den Broeck, G. V., Renkens, J., Shterionov, D. S., Gutmann, B., Thon, I., Janssens, G. and Raedt, L. D. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming 15,* 3, 358–401.

Ganian, R., Ramanujan, M. S., and Szeider, S. 2017. Combining treewidth and backdoors for CSP. In *STACS'17*. 36:1–36:17.

Gelfond, M. 1991. Strong introspection. In *Proc. AAAI*. AAAI Press/The MIT Press, 386–391.

Hanks, S. and Mcdermott, D. 1986. Default reasoning, nonmonotonic logics, and the frame problem. In *AAAI'86: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence,* 328–333.

Hecher, M. 2020. Treewidth-aware reductions of normal ASP to SAT - is normal ASP harder than SAT after all? In *KR 2020*. 485–495.

Hecher, M., Morak, M. and Woltran, S. 2020. Structural decompositions of epistemic logic programs. In *AAAI 2020*. AAAI Press, 2830–2837.

Hecher, M., Thier, P. and Woltran, S. 2020b. Taming high treewidth with abstraction, nested dynamic programming, and database technology. In *SAT 2020*. LNCS, vol. 12178. Springer, 343–360.

Jakl, M., Pichler, R. and Woltran, S. 2009. Answer-set programming with bounded treewidth. In *IJCAI*. 816–822.

Kahl, P. T., Watson, R., Balai, E., Gelfond, M. and Zhang, Y. 2015. The language of epistemic specifications (refined) including a prototype solver. *Journal of Logic and Computation 25*.

Lonc, Z. and Truszczynski, M. 2003. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Log. 4,* 1, 91–119.

Morak, M. 2019. Epistemic logic programs: A different world view. In *Proc. ICLP*, 52–64.

Shen, Y. and Eiter, T. 2016. Evaluating epistemic negation in answer set programming. *Artificial Intelligence 237*, 115–135.

Son, T. C., Le, T., Kahl, P. T. and Leclerc, A. P. 2017. On computing world views of epistemic logic programs. In *IJCAI*, 1269–1275.

Truszczynski, M. 2011. Revisiting epistemic specifications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*. LNCS, vol. 6565. Springer Verlag, 315–333.