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 1 January 2003; revised 1 January 2003; accepted 1 January 2003

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 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. This work is under consideration for acceptance in TPLP.

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, e.g., (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 logic programs 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 $\text{not}$, 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, e.g., (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, e.g., (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 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\footnote{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.} 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, 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 tree decomposition of the abstraction. So, the idea of these abstractions compared to the full primal graph is to decrease treewidth such that still structural information in the form of tree decompositions 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. 2020], we take up this idea and design an improved algorithm for evaluating epistemic logic programs by means of treewidth. Our algorithm lifts nested dynamic programming [Hecher et al. 2020b] 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, e.g., [Jakl et al. 2009] [Hecher 2020]. The concept of using abstractions was stipulated before as well, but in a different context [Hecher et al. 2020b] or with the purpose of establishing theoretical results [Ganian 2009; Hecher 2020]. The concept of using abstractions was stipulated before as well, but in a different context (Hecher et al. 2020) 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 tree decompositions. While the solver selp [Bichler et al. 2020] uses decompositions for breaking large rules into smaller ones, the solving itself is not guided by tree decompositions. Also studies for measures different from treewidth have been conducted in the ASP domain, e.g., [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 \( a_1 \lor \cdots \lor a_k \leftarrow a_{k+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \).

For a rule \( r \), we let \( H_r := \{ a_1, \ldots, a_t \} \), \( B_r^+ := \{ a_{t+1}, \ldots, a_m \} \), 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 \( \text{ats}(r) \) and \( \text{ats}(P) := \bigcup_{r \in P} \text{ats}(r) \). An interpretation \( I \subseteq \text{ats}(P) \) is a set of atoms. \( I \) satisfies a rule \( r \) if \( (H_r \cup B_r^+) \cap I \neq \emptyset \), or \( B_r^- \cap I \neq \emptyset \), 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 program \( P \) by \( AS(P) \). The problem of deciding whether a program has an answer set, i.e., whether \( AS(P) \neq \emptyset \), is \( \Sigma^p_2 \)-complete [Eiter and Gottlob 1995].

Example 1

Consider the program \( P := \{ \{ a \lor b \}, \{ c \leftarrow \neg d \}, \{ d \leftarrow \neg e \} \} \). The set \( AS(P) \), denoting the answer sets for the logic program \( P \), 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 \( \text{root}(T) \), and \( \chi \) 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 tree decomposition (TD) \cite{BodlaenderK96} 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 tree decomposition 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 type \( \chi(t) \) is leaf if \( t \) has no children and \( \chi(t) = \emptyset \); join if \( t \) has children \( r' \) and \( r'' \) with \( r' \neq r'' \) and \( \chi(t) = \chi(r') = \chi(r'') \); intr (“introduce”) if \( t \) has a single child \( r' \), \( \chi(t) \subseteq \chi(r) \) and \( |\chi(r)| = |\chi(r')| + 1 \); rem (“removal”) if \( t \) has a single child \( r' \), \( \chi(t) \supseteq \chi(r) \) and \( |\chi(r')| = |\chi(t)| + 1 \). If for every node \( t \in T \), type \( (t) \in \{\text{leaf}, \text{join}, \text{intr}, \text{rem}\} \), then \( (T, \chi) \) is called nice. For every TD, one can compute a nice tree decomposition in polynomial time \cite{BodlaenderK96} 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 \( \text{not}_\ell \), where \( \ell \) is a literal and \( \text{not} \) is the epistemic negation operator. Let \( k, m, j, n \) be non-negative integers such that \( k \leq m \leq j \leq n \) and \( a_1, \ldots, a_k \) be distinct propositional atoms. An epistemic logic program (ELP) is a set \( \Pi \) 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 \( \ell_i \) with \( k + 1 \leq i \leq m \) is a literal over atom \( a_i \), and each \( \xi_i \) with \( m + 1 \leq i \leq n \) is an epistemic literal of the form \( \text{not}_\ell \), where \( \ell \) is a literal over atom \( a_i \). Then, \( \text{ats}(r) := \{a_1, \ldots, a_k\} \) denotes the set of atoms occurring in an ELP rule \( r \), \( \text{e-ats}(r) := \{a_{m+1}, \ldots, a_n\} \) denotes the set of epistemic atoms, i.e., those used in epistemic literals of \( r \), and \( \text{a-ats}(r) := \text{ats}(r) \setminus \text{e-ats}(r) \) refers to the non-epistemic atoms of \( r \). We call \( r \) purely-epistemic if \( \text{a-ats}(r) = \emptyset \). These notions naturally extend to programs. In a rule we sometimes write \( K \ell \) and \( M \ell \) for a literal \( \ell \), which refers to the expressions \( \neg \text{not}_\ell \) and \( \neg \text{not}_\ell \), respectively.

Given an ELP \( \Pi \), a world view interpretation (WVI) \( I \) for \( \Pi \) is a consistent set \( I \) of literals over a set \( A \subseteq \text{ats}(\Pi) \) of atoms, i.e., \( I \subseteq \{a, \neg a \mid a \in A\} \) such that there 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 \( X \subseteq \text{ats}(\Pi) \) of atoms obtained by restricting \( I \) to \( Y = \langle A \cap X \rangle \) 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, e.g., \cite{Gelfond91, Truszczynski11, Kahl15, ShenE16}, we follow the approach of \cite{Gelfond91}, syntactically denoted according to recent work \cite{Morak19}. The epistemic reduct \cite{Gelfond91} of program \( \Pi \) w.r.t. a WVI \( I \) over \( A \), denoted \( \Pi' \), is defined as \( \Pi' = \{r' \mid r \in \Pi\} \) where \( r' \) denotes rule \( r \) where each epistemic literal \( \text{not}_\ell \), whose atom is also in \( A \), is replaced by \( \bot \) if \( \ell \in I \), and by
Utilizing Treewidth for Quantitative Reasoning on Epistemic Logic Programs

with the corresponding epistemic reduct. Without loss of generality we only consider ELPs \( \Pi \), where every epistemic atom appears non-epistemically, i.e., \( e\text{-}ats(\Pi) \equiv a\text{-}ats(\Pi^{\text{epist}}) \). We refer by \( \Pi \cup I \) to the ELP \( \Pi \cup \{ \leftarrow \neg K \ell \mid \ell \in I \} \cup \{ \leftarrow \neg M a; \leftarrow \neg M \leftarrow a \mid a \in A, a \notin I, \neg a \notin I \} \) used for verifying whether \( I \) can be extended to a WV. The set of WVs of an ELP \( \Pi \) is denoted \( W(VS(\Pi)) \). One of the reasoning tasks for ELPs is \textit{world view existence} deciding for an ELP \( \Pi \) whether \( W(VS(\Pi)) \neq \emptyset \). This problem is known to be \( \Sigma^P_3 \)-complete [Truszczynski 2011].

Example 2
Consider program \( \Pi : = P \cup \{ a \leftarrow \neg Kb; b \leftarrow \neg Ka ; c \leftarrow \neg K d; d \leftarrow \neg K c ; \leftarrow \neg Ka, \neg K \leftarrow a ; \leftarrow \neg Kb, \neg K \leftarrow b ; \leftarrow \neg Ka, \neg Kc; \leftarrow \neg Ka, \neg Kb, Kc; \leftarrow Kc, Kd \} \), where \( P \) is defined as in Example 1 i.e., the ELP \( \Pi \) depicts an epistemic extension of the plain logic program \( P \). For simplicity, let the rules be numbered equally from \( r_1 \) to \( r_12 \). When constructing a WV I over \( e\text{-}ats(\Pi) \) one guesses for each atom \( a \in e\text{-}ats(\Pi) \) either (1) \( a \in I \), (2) \( \neg a \in I \) or (3) \( \{ a, \neg a \} \cap I = \emptyset \) as described earlier, i.e., for the three atoms in \( e\text{-}ats(\Pi) \) we obtain 3\(^4\) possibilities. Each WV I can be checked with the corresponding epistemic reduct \( \Pi^I \) by verifying Definition 1 for \( AS(\Pi^I) \).

Consider \( I_1 = \{ a, d, \neg b, \neg c \} \) with its epistemic reduct \( \Pi^I_1 : = P \cup \{ a ; d \} \). Note that the epistemic reduct is indeed a plain logic program, 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 \( \Pi \). Similarly WVs \( I_2 = \{ a, c, \neg b, \neg d \} \) and \( I_3 = \{ b, c, \neg a, \neg d \} \) can be constructed and correctly validated as WVs, i.e., \( W(VS(\Pi)) = \{ I_1, I_2, I_3 \} \).

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

As a special case, where \( Q = \emptyset \), a problem instance \#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 \( \Pi \) be an ELP and \( Q \) be a WV over \textit{ats}(\Pi). We define the \textit{probability of} \( Q \) being compatible with a world view by \( \text{prob}(\Pi, Q) := \frac{\#\text{ELP}(\Pi, Q)}{\#\text{ELP}(\Pi, \emptyset)} \).

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 \textit{probabilistic world view acceptance}.

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

For Definitions 2 and 3, we only consider WVs over epistemic atoms to simplify presentation.\(^2\)

\(^2\) This is not a hard restriction that could be circumvented for a non-epistemic atom \( a \), e.g., via constraint \( \leftarrow \neg Ka.Ka \).
4 Quantitative Reasoning for ELPs via Dynamic Programming

Next, we discuss core ideas of dynamic programming for the evaluation of epistemic logic programs. 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 tree decomposition. 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 $\tau_t$. A table $\tau_t$ 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 $A$. The actual length, content, and meaning of the rows depend on the algorithm $A$ that derives tables.

The DP approach for solving problems of an epistemic logic program relies on a table algorithm $A$ and consists of the following four steps:

- **Prepare**: Construct a graph representation $G$ of the given ELP $\Pi$.
- **Decompose**: Compute a tree decomposition $(T, \chi)$ of $G$, which can be obtained by using efficient heuristics (Abseher et al. 2017).
- **Compute**: Execute table algorithm $A$ for every node $t$ of $T$ in post-order, which returns the corresponding table for $t$. Algorithm $A$ takes as input the corresponding bag $\chi(t)$, the assigned instance $\Pi_t$ 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 $\tau_t$.
- **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 tree decompositions due to clear case distinctions depending on type $(t)$. However, the implemented architecture does not depend on certain normal forms of tree decompositions. 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 world view interpretations of an ELP $\Pi$, which is a WVI $I$ over $\text{ats}(\Pi)$ such that $AS(\{r \in \Pi \mid a\text{-ats}(r) = \emptyset\}) \neq \emptyset$, denoted by $I \models_p \Pi$. Observe that every WV of $\Pi$ 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 WVI)**
The problem of counting for a given ELP $\Pi$ the number of plausible WVI is #P-complete.

**Proof (Sketch)**
For membership, observe that one can guess a WVI $I$ and then check whether $I \models_p \Pi$ 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 $\Pi$ that contains for every variable $v$ of $F$ a rule $\leftarrow \neg K v$, $\neg K \neg v$ and for every clause $c_i = \ell_1 \lor \ell_2 \lor \ell_3$ of $F$ a rule $\leftarrow \neg K \ell_1, \neg K \ell_2, \neg K \ell_3$. Then, the number of plausible WVI of $\Pi$ precisely captures the number of models of $F$. $\square$
Fig. 1. Epistemic primal graph $E_{\Pi}$ (left) of $\Pi$ from Example 2 and a TD $\mathcal{T}$ (right) of $E_{\Pi}$.

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 $E_{\Pi}$ of an ELP $\Pi$, whose vertices stem only from the epistemic atoms $e$-ats($\Pi$) and there is an edge between two vertices whenever the corresponding epistemic atoms appear together in a common purely-epistemic rule of $\Pi$. Formally, let $E_{\Pi} = (e$-ats($\Pi$)$^e, E)$ with $E$ being $\{\{a^e, b^e\} \mid r \in \Pi, a$-ats($r$) = $\emptyset, \{a, b\} \subseteq e$-ats($r$)$\}$. Now, let $\mathcal{T} = (T, \chi)$ be a TD of the epistemic primal graph $E_{\Pi}$ and $t$ be a node of $T$. Then, the epistemic bag program for $t$ is given by $\Pi_t := \{r \in \Pi \mid a$-ats($r$) = $\emptyset, e$-ats($r$)$^e \subseteq \chi(r)\}$. This allows us to refer to the epistemic bag program up to $t$ by $\Pi_{\leq t} := \bigcup_{i \in T} \Pi_{ti}$, where $t_i$ is a node of $t$ in $T$ and $\Pi_{ti} \cup \Pi_{t'}$, 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 $\Pi$.

Example 4

Figure 1 depicts the epistemic primal graph $E_{\Pi}$ for $\Pi$ as defined in Example 2 as well as one corresponding TD $\mathcal{T}$ of $E_{\Pi}$ 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 $\Pi_t$ only rules solely built from $e$-ats($\Pi$), i.e., 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 $\#\text{PWV}$ for counting plausible world view interpretations. 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 $\#\text{PWV}$ 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 $\Pi$.

As already mentioned above, for the ease of presentation, table algorithm $\#\text{PWV}$ is given for nice tree decompositions, i.e., 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 $\Pi_t$. For nodes $t$ with $\text{type}(t) = \text{rem}$, 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, i.e., 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, e.g., when processing a node of type join, since there are at most two child nodes.

Example 5

Considering program $\Pi$ from Example 2, we obtain three world views as described earlier. Table

\[ \begin{array}{c|c}
\hline
X & \text{count} \\
\hline
\{\{a^e, b^e\}\} & 1 \\
\{\{e^e\}\} & 1 \\
\{\{d^e\}\} & 1 \\
\hline
\end{array} \]

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

Listing 1: Table algorithm $\mathbb{PWW}(\chi_t, \Pi_t, (\tau_1, \ldots, \tau_i))$ for Counting Plausible WVIs.

In: Node $t$, bag $\chi_t$, epistemic bag program $\Pi_t$, and child tables $(\tau_1, \ldots, \tau_i)$ of $t$.

Out: Table $\tau_i$.

1. If $\text{type}(t) = \text{leaf}$ then
   2. $\tau_i \leftarrow \{(\emptyset, 1)\}$
2. Else if $\text{type}(t) = \text{intr}$ and $a^\epsilon \in \chi_t$ is introduced then
   3. $\tau_i \leftarrow \{\langle I, c \rangle \mid \langle I, c \rangle \in \tau_1, J \in \{I \cup \{a\}, I \cup \{-a\}, J \models \Pi_t \}$
4. Else if $\text{type}(t) = \text{rem}$ and $a^\epsilon \notin \chi_t$ is removed then
   5. $\tau_i \leftarrow \{\langle \lnot I, \sum_j \sum_{j \in} I_j \cup c_j \rangle \mid \langle I, c \rangle \in \tau_1, I' = I \setminus \{a, -a\}\}$
6. Else if $\text{type}(t) = \text{join}$ then
   7. $\tau_i \leftarrow \{\langle I, c_1 \cdot c_2 \rangle \mid \langle I, c_1 \rangle \in \tau_1, \langle I, c_2 \rangle \in \tau_2\}$
8. Return $\tau_i$

Fig. 2. A nice TD $\mathcal{T}$ of the epistemic primal graph $E_{11}$ of program $\Pi$ from Example 2 as well as selected tables obtained by $\mathbb{PWW}$ on $\Pi$ and $\mathcal{T}$.

Algorithm $\mathbb{PWW}$ can be used to restrict the possible WVIs. Figure 2 shows a nice tree decomposition $\mathcal{T} = (T, \chi)$ of $E_{11}$ and a selection of the tables $\tau_1, \ldots, \tau_{12}$, which illustrate computation results obtained during post-order traversal of $\mathcal{T}$ by $\mathbb{PWW}$.

Table $\tau_1 = \{(\emptyset, 1)\}$ as per definition for $\text{type}(t_1) = \text{leaf}$. Since $\text{type}(t_2) = \text{intr}$, we construct table $\tau_2$ from $\tau_1$ by taking $I_{1,i}, I_{1,i} \cup \{b\}$ and $I_{1,i} \cup \{-b\}$ for each $\langle I_{1,i}, c_{1,i} \rangle \in \tau_1$ (corresponding to a guess on $b$). Since $\text{e-ats}(r_1) \subseteq \chi(t_2)$ we have $\Pi_{t_2} = \{r_0\}$ for $t_2$ as described in Example 4. In consequence, for each $I_{2,i}$ of table $\tau_2$, we have $I_{2,i} \models \{r_0\}$ since $\mathbb{PWW}$ enforces satisfiability of $\Pi_t$ in node $t$. Then, $t_2$ introduces $a^c$ and $t_3$ 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 $\tau_7$ to $t_9$ similarly. Since $\text{type}(t_5) = \text{rem}$, we remove atom $a$ from all elements in $\tau_4$ to construct $\tau_5$. As described earlier, this is accomplished by summing up the counters for matching WVIs when removing the atom $a$, e.g., since the remaining, guessed atoms $b^c$ and $c^e$ are matching, counters for line 2 and 4 in table $\tau_4$ are summed up, resulting in line 2 in table $\tau_5$. Note that we have already seen all rules where $a^c$ occurs and hence $a^c$ can no longer affect witnesses during the remaining traversal. We similarly construct $\tau_6 = \{(\emptyset, 4), \{c, 3\}, \{-c, 2\}\}$ and $\tau_{10} = \{(\emptyset, 3), \{c, 2\}, \{-c, 3\}\}$. Since $\text{type}(t_{11}) = \text{join}$, we construct table $\tau_{11}$ 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 tree decompositions) for every $r \in \Pi$, atoms $a$-ats($r$) occur together in at least one common bag. Hence, $\Pi = \Pi_{t_{12}}$ and since
Utilizing Treewidth for Quantitative Reasoning on Epistemic Logic Programs

Given a bag program \( \Pi \), we ensure that each nesting atom is evaluated in one unique node of the primal graph \( G^\Pi \). By construction of the nested primal graph, any atom not in \( A^e \) is subject to nesting looks like. To formalize this, we assume a TD \( \mathcal{T} \) for the nested primal graph \( G^\Pi \), and there is a connected component \( \Pi \) such that all neighbor vertices of \( \Pi \) are contained in \( A^e \). Hence, to enable nested evaluation, we require suitable abstractions of the primal graph, where we can construct from the tables. For example, we obtain the interpretation \( \{ \neg a, b, c, \neg d \} = I_{112} \cup I_{48} \cup I_{96} \), 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 \( \Pi \) be an epistemic logic program. Then, the primal graph \( G^\Pi \) uses atoms and epistemic atoms as vertices and it is defined by \( G^\Pi := \langle \{ a^e \mid a \in \circ-ats(\Pi), \circ \in \{ a, e \} \}, E \rangle \), where \( E := \{ \{ a^e, b^e \} \mid r \in \Pi, a \in \circ-ats(r), b \in \star-ats(r), \{ \circ, \star \} \subseteq \{ a, e \} \} \cup \{ \{ a^e, a^e \} \mid a \in e-ats(\Pi) \} \). For our purposes, we require suitable abstractions of \( G^\Pi \), given as follows. A non-epistemic path in \( G^\Pi \) is a path of the form \( a^e, v_1^e, \ldots, v_l^e, b^e \) with \( l \geq 0 \).

The nested primal graph \( G^\Pi \) over a given set \( A \subseteq e-ats(\Pi) \) of epistemic atoms is given by \( G^\Pi := \langle A^e, E^g \rangle \) with \( E^g := \{ \{ a^e, b^e \} \mid a, b \in A \} \), there is a non-epistemic path from \( a^e \) to \( b^e \) in \( G^\Pi \).

**Example 6**

Recall program \( \Pi \) of Example 2. Figure 3 shows the primal graph \( G^\Pi \) for program \( \Pi \). Given epistemic atoms \( A = \{ b, c, d \} \) the nested primal graph \( G^\Pi \) 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 \( G^\Pi \).

Indeed, in this section we use the nested primal graph \( G^\Pi \) 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 tree decomposition, 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 \( A = e-ats(\Pi) \), we end up with full DP and zero nesting, whereas setting \( A = \emptyset \) results in full nesting, i.e., 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 \( G^\Pi \) 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 \( \chi(t) \) in \( G^\Pi \) that are in \( A^e \), are contained in \( \chi(t) \), i.e., \( \{ a^e \mid a \in A, u \in U \} \), there is a non-epistemic path from \( a^e \) 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 \( \Pi^A_t \) for a node \( t \) of \( T \), i.e., the ELP subject to nesting, equals \( \Pi^A_t := \{ r \in \Pi \mid a-ats(r) \subseteq A_t, e-ats(r) \subseteq \chi(t) \} \).
Example 7

Considering program \( \Pi \) from Example 2 and the nested primal graph \( G_1^\Pi \) for \( A = \{b,c,d\} \), Figure 3 shows a corresponding TD \( T \) of which building a compatible set in the form of \( A \) is sufficient for DP (threshold \( t \)). Then nested bag programs \( \Pi_1^A = \{r_1,r_4,r_5,r_8,r_9,r_{10},r_{11}\} \) and \( \Pi_2^A = \{r_2,r_3,r_6,r_7,r_{12}\} \) emerge from \( A_1 = \{a,b\} \) and \( A_2 = \{c,d\} \), respectively. Note that \( \Pi_3^A = \emptyset \) because of \( A_3 = \emptyset \).

Nested Dynamic Programming for ELPs

Next, we discuss nested dynamic programming (nested DP) in order to count world views of an ELP \( \Pi \). Thereby we aim at solving the more elaborated problem \#ELP(\( \Pi \cup W, \emptyset \)) for a world view interpretation \( W \) over a set \( X \subseteq a-ats(\Pi) \) of atoms of \( \Pi \). This problem amounts to counting the number of world views of \( \Pi \) 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 \#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 \( \Pi \) and the WVI \( W \). Listing 2 consists of four separated blocks. The first block (Lines 1–4) comprises solving the base case where \( \Pi \) has no epistemic atoms, i.e., no epistemic “decisions” are left for solving \( \Pi \). 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 Definition 1 respectively. Otherwise all four conditions of Definition 1 are verified via one ELP solver call. The next block consists of Lines 5–10 which computes a tree decomposition \( T \) of the primal graph of \( \Pi \) (nested primal graph with \( A = e-ats(\Pi) \)). Then this block utilizes standard ELP solvers in case width(\( T \)) is out of reach (threshold\(_{hybrid}\)) or nesting is already too deep (threshold\(_{depth}\)). If this is not the case and width(\( T \)) is insufficient for DP (threshold\(_{abseq}\)), the third block consisting of Lines 8–10 chooses a suitable abstraction \( \Lambda \) and computes a TD \( T \) of the nested primal graph \( G_1^\Lambda \). Finally, the last block comprises of the remaining lines of Listing 2 which performs DP on the TD \( T \) 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 \#ELP that is used during DP in Line 13 discussed next.

The table algorithm \#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 5 of Listing 3.

Example 8
Finding an algorithm for counting, we only briefly discuss how to extend the table algorithm for counting, as well as selected tables obtained by \#ELP on \( \Pi \) and \( \mathcal{T} \).

Recall program \( \Pi \), set \( A \) of epistemic atoms, TD \( \mathcal{T} \) of nested primal graph \( G_\Pi^3 \) and nested bag programs given in Example 2. Figure 4 illustrates computation results obtained during post-order traversal of \( \mathcal{T} \) by \#ELP. Notice that similar to \#PWV the algorithms enforces the entailment of nested bag programs given in Example 2. 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 2 one can identify that epistemic program \( \Pi \) has three world views which can be reconstructing joining agreeing assignments of the tables in-order. For example, we obtain the (incomplete) world view \( \{b, c, \neg d\} = I_2 \cup I_3 \cup I_2, c, a \}_{b, c, \neg d} \), 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 \( \text{PELP} \) 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 and \( q \) is computed similarly to \( c \), but in Line 5 the recursive call for obtaining \( q' \) involves the nested bag program extended by \( Q \), i.e., \( \Pi^3 \cup Q \). Then, instead of summing up counters \( c \) in Line 14 of Listing 2 these adapted tables computed by \( \text{PELP} \) explained above are used to sum up fractions \( \frac{q}{c} \), which leads the desired result. Detailed algorithms for \( \text{PELP} \) and \( \text{NestELP}_{\text{PELP}} \) are depicted in the appendix, cf. Listings 4 and 5.
5 Implementation & Preliminary Experiments

We implemented the algorithm NestELP, resulting in the solver nestelp\footnote{The solver nestelp is open source and available at github.com/viktorbesin/nestelp}, 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, 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\footnote{htd (Abseher et al. 2017), which for every instance outputs TDs of decent widths in a runtime below some seconds.} (Abseher et al. 2017), which for every instance outputs TDs of decent widths in a runtime below some seconds. For solving decision problems of logic programs 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 \texttt{threshold\textsubscript{hybrid}} = 45, \texttt{threshold\textsubscript{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 2 i.e., searching for epistemic atoms when constructing the nested primal graph, we employ a logic program similar to nesthdb. Intuitively, we thereby aim for a preferably large set \texttt{A} of epistemic atoms such that the resulting graph \texttt{N} is reasonably sparse. This is achieved heuristically by minimizing the number of edges of \texttt{N} in \texttt{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.

### 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 student’s 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, i.e., 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, i.e., 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 student’s 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.
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, \textit{nestelpl} 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 \textit{nestelpl} 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 \textit{nestelpl} can exploit all its features. Further it can be seen that parallelism of \textit{nestelpl} has better performance than the single-core experiments (\textit{nestelpl} (sc)), indicating that there are enough independent nodes such that parallelism is beneficial. Even with the fair comparison to \textit{eclingo}, the solver \textit{nestelpl} 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 \textit{nestelpl} 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. 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 \textit{nestelpl} uses \textit{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 epistemic logic programs (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 nestel1p 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, nestel1p 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 nestel1p. Further, given recent insights on complexity results for treewidth, e.g., [Fichte et al. 2020], 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.

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 Pract. Log. Program. 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. J. 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 Pract. Log. Program. 20, 6, 834–847.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. 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. Ann. Math. Artif. Intell. 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 Pract. Log. Program. 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. 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. J. Log. Comput. 25.

LONC, Z. AND TRUSZCZYSKII, 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. Artif. Intell. 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.

TRUSZCZYSNII, M. 2011. Revisiting epistemic specifications. In Logic Programming. Knowledge Representation, and Nonmonotonic Reasoning. LNCS, vol. 6565. Springer Verlag, 315–333.
Appendix A  Probabilistic Reasoning

Listing 4: Table algorithm $\text{PELP}(\text{depth}, \chi, \Pi, \Pi^\delta, W, Q, (\tau_1, \ldots, \tau_i))$ for nice TDs of the nested primal graph representation.

In: Nesting depth $\geq 0$, bag $\chi$, epistemic bag program $\Pi$, nested bag program $\Pi^\delta$, world view interpretation $W$, WVI (query) $Q$, and sequence $(\tau_1, \ldots, \tau_i)$ of child tables of $t$.
Out: Table $\tau_i$.

1 if type($t$) = leaf then $\tau_i \leftarrow \{[0, 1, 1]\}$
2 else if type($t$) = intr and $a^\delta \in \chi$ is introduced then
3 $\tau_i \leftarrow \{(l, c, q') \mid (l, c, q) \in \tau_j, J \in \{I, J \cup \{a\}, J \cup \{-a\}\}, J \models p \Pi,\$
4 $P = (\Pi^\delta)^l, c' = c \cdot \text{NestELP}(\text{depth} + 1, P, (W \cup J)_{\{a\}-ats(P)}), c' > 0\}$
5 $P' = (\Pi^\delta \cup Q)^l, q' = q \cdot \text{NestELP}(\text{depth} + 1, P', (W \cup J)_{\{a\}-ats(P')}, q' > 0\}$
6 else if type($t$) = rem and $a^\delta \notin \chi$ is removed then
7 $\tau_i \leftarrow \{(l', \sum_{(c', q') \in \tau_j, J \subseteq J} c', \sum_{(c', q') \in \tau_j, J \subseteq J} q') \mid (l, c, q) \in \tau_j, J' = I \setminus \{a, -a\}\}$
8 else if type($t$) = join then
9 $\tau_i \leftarrow \{(l, c_1, c_2, q_1, q_2) \mid (l, c_1, q_1) \in \tau_1, (l, c_2, q_2) \in \tau_2\}$

Listing 5: Algorithm $\text{NestELP}_{PELP}(\text{depth}, \Pi, W, Q)$ for probabilistic world view acceptance via nested DP.

In: Nesting depth $\geq 0$, epistemic logic program $\Pi$, WVI $W$ over a set $X \subseteq a\text{-ats}(\Pi)$ of atoms, and WVI (query) $Q$.
Out: The probability $\text{prob}(\Pi \cup W, Q)$ of $Q$ being compatible with a world view.

1 $A \leftarrow e\text{-ats}(\Pi)$
2 if $A = \emptyset$ /* No Epistemic Decisions left; Verify Decisions */ then
3 if $\{a \in X \mid a \notin W, -a \notin W\} = \emptyset$ then return $|\text{AS}(\Pi)| = 1 \text{ and } |\text{AS}(\Pi \cup \{\leftarrow W\})| = 0$ /* ASP */
4 else return $\text{WVS}(\Pi \cup W) \neq 0$ /* Verify via Standard ELP Solver */
5 $T = (T, \chi)$ $\leftarrow$ Decompose($G_{H}$) /* Decompose via Heuristics */
6 if width($T$) $\geq$ threshold$_{hybrid}$ or depth $\geq$ threshold$_{depth}$ /* Standard ELP Solver */ then
7 return $\text{prob}(\Pi \cup W, Q)$
8 if width($T$) $\geq$ threshold$_{abstr}$ /* Abstract & Decompose via Heuristics */ then
9 $A \leftarrow$ Choose-Abstraction($A, \Pi$)
10 $T = (T, \chi)$ $\leftarrow$ Decompose($G_{H}$)
11 for iterate $t$ in post-order($T$) /* Dynamic Programming */ do
12 Child-Tabs $\leftarrow (\tau_1, \ldots, \tau_i)$ where children($t$) = ($t_1, \ldots, t_i$)
13 $\tau_i \leftarrow \text{PELP}(\text{depth}, \chi(t), \Pi, \Pi^\delta, W, Q, \text{Child-Tabs})$
14 return $\sum_{(l, c, q) \in \tau_{\text{out}}(t)} c > 0$ /* Return Total Probability */