Answer-Set Programming with Bounded Treewidth*

Michael Jakl, Reinhard Pichler and Stefan Woltran
Institute of Information Systems, Vienna University of Technology
Favoritenstrasse 9–11; A-1040 Wien; Austria
{jakl, pichler, woltran}@dbai.tuwien.ac.at

Abstract

In this paper, we present a novel approach to the evaluation of propositional answer-set programs. In particular, for programs with bounded treewidth, our algorithm is capable of (i) computing the number of answer sets in linear time and (ii) enumerating all answer sets with linear delay. Our algorithm relies on dynamic programming. Therefore, our approach significantly differs from standard ASP systems which implement techniques stemming from SAT or CSP, and thus usually do not exploit fixed parameter properties of the programs. We provide first experimental results which underline that, for programs with low treewidth, even a prototypical implementation is competitive compared to state-of-the-art systems.

1 Introduction

Over the past decade, *Answer-Set Programming* (ASP, for short) [Marek and Truszczyński, 1999; Niemelä, 1999], also known as A-Prolog [Baral, 2002], has become an increasingly acknowledged paradigm for declarative programming. The basic idea of ASP is to encode solutions to a problem into the models of a program in such a way that the solutions are described in terms of rules and constraints. ASP enjoys a large collection of successful applications in the areas of AI and KR showing the potential of this paradigm. However, the underlying complexity of evaluating propositional disjunctive programs (which are the objects we deal with here) shows that the problems ASP has to deal with are highly intractable: the decision problems are located on the second level of the polynomial hierarchy (see [Eiter and Gottlob, 1995]), and the problem of counting all answer sets can analogously be shown to be #NP-complete.

An interesting approach to dealing with such intractable problems is parameterized complexity. In fact, hard problems can become tractable if some problem parameter is bounded by a fixed constant. Such problems are also called fixed-parameter tractable (FPT). One important parameter is treewidth, which measures the "tree-likeness" of a graph. By using a seminal result due to Courcelle [1990], several FPT results in the area of AI and KR have been recently proven tractable by Gottlob *et al.* [2006], among them also the problem of deciding ASP consistency (i.e. whether a disjunctive logic program has at least one answer set). Treewidth hereby has to be adapted suitably, for instance, by using the incidence graph of the program. However, an FPT result itself does not immediately lead to an efficient algorithm. Indeed, quite some work has been done within the last years to overcome this obstacle. We mention here only some recent results for counting problems: Samer and Szeider [2007] have proposed algorithms for #SAT (counting the number of models of a CNF formula) which follow the principle of dynamic programming; Jakl *et al.* [2008], on the other hand, map different counting problems to a certain (tractable) datalog fragment. Both approaches have in common that they use the concept of tree-decompositions and proceed by a bottom-up traversal of the tree, such that at each node n, certain information about the subproblem (represented by the subtree rooted at n) is available. Consequently, results for the entire problem can be read off the root of the tree decomposition. Algorithms for counting problems are of particular interest here, since they are closely related to the problem of enumerating solutions, which is, of course, a central requirement in ASP.

In this work, we generalize the dynamic programming approach for #SAT due to Samer and Szeider [2007] to the world of ASP in order to count and enumerate all answer sets of a given program. We thus provide a novel approach for computing answer sets, which significantly differs from standard ASP systems (see [Gebser *et al.*, 2007] for an overview) that usually do not exploit fixed parameter properties. We implemented the proposed method in a prototype system. Our system should not be seen as competitor to general-purpose ASP solvers, but as an alternative for application scenarios where the problems possess low treewidth; usually the ASP encodings then have similarly low treewidth. It is generally believed that many practically relevant problems have low treewidth. Thorup [1998], for instance, shows that the treewidth of the control-flow graph of structured programs (more precisely, goto-free C programs) is at most six. A detailed discussion of applications to which treewidth has been successfully applied is given by Bodlaender [1993].

Results. Our main contributions are as follows.

- An FPT algorithm for deciding ASP consistency in *linear time* w.r.t. the size of the program.
- An FPT algorithm for counting the number of answer

*This work was supported by the Austrian Science Fund (FWF), project P20704-N18.

sets in *linear time* w.r.t. the size of the program (assuming unit cost for arithmetic operations).

- A novel method for enumerating *all* answer sets with *linear delay*.
- Presentation of a first prototype implementation and some preliminary experimental results.

2 Preliminaries

Throughout the paper, we assume a universe U of propositional atoms. A literal is either an atom or a negated atom $\overline{a}$. For a set A of atoms, $\overline{A}$ denotes $\{\overline{a} \mid a \in A\}$. Clauses are sets of literals. An interpretation I is a set of atoms and we define, for a clause c and $O \subseteq U$, $I \models_O c$ iff $((I \cap O) \cup \overline{(O \setminus I)}) \cap c \neq \emptyset$. For a set C of clauses, $I \models_O C$ holds iff $I \models_O c$, for each $c \in C$. For $O = U$, we usually write $\models$ instead of $\models_O$.

Answer-Set Semantics for Logic Programs. A propositional disjunctive logic program (or simply, a program) is a set of rules $a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$, ($n > 0$, $n \geq m \geq l$), where all a_i are from U. A rule r of this form consists of a head $\mathrm{H}(r) = \{a_1, \ldots, a_l\}$ and a body, given by $\mathrm{B}^+(r) = \{a_{l+1}, \ldots, a_m\}$ and $\mathrm{B}^-(r) = \{a_{m+1}, \ldots, a_n\}$. By $At(R)$ we denote the set of atoms occurring in program R. We often identify a program R with the clause set $\{\mathrm{H}(r) \cup \mathrm{B}^-(r) \cup \overline{\mathrm{B}^+(r)} \mid r \in R\}$, and likewise, define the reduct R^I of a program R wrt. an interpretation I as $\{\mathrm{H}(r) \cup \overline{\mathrm{B}^+(r)} \mid \mathrm{B}^-(r) \cap I = \emptyset, r \in R\}$. Following Gelfond and Lifschitz [1991], an interpretation I is an *answer set* of a program R iff $I \models R$ and for no $J \subset I$, $J \models R^I$. The set of all answer sets of a program R is denoted by $\mathcal{AS}(R)$.

Example 2.1 *We will use as a running example throughout the paper the program P which consists of the following rules*

$$r_1 = u \leftarrow v, y; \quad r_2 = z \leftarrow u; \quad r_3 = v \leftarrow w; \quad r_4 = w \leftarrow x; \quad r_5 = x \leftarrow \neg y, \neg z.$$

P has a unique answer set $\{v, w, x\}$.

Tree Decomposition and Treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, β), where T is a tree and β maps each node n of T (we use $n \in T$ as a shorthand below), to a *bag* $\beta(n) \subseteq 2^V$, such that the following conditions are met:

- For each $v \in V$, there is an $n \in T$ such that $v \in \beta(n)$.
- For each $(v, w) \in E$. there is an $n \in T$, s.t. $v, w \in \beta(n)$.
- For any three nodes $n_1, n_2, n_3 \in T$, if n_2 lies on the path from n_1 to n_3, then $\beta(n_1) \cap \beta(n_3) \subseteq \beta(n_2)$.

A tree decomposition (T, β) is called *normalized* (or *nice*) [Kloks, 1994] if (i) each node in T has at most two children; (ii) for each node n with two children n_1, n_2, $\beta(n) = \beta(n_1) = \beta(n_2)$; and (iii) for each node n with one child n', $\beta(n)$ and $\beta(n')$ differ in exactly one element.

The *width* of a tree decomposition is defined as the cardinality of its largest bag $\beta(n)$ minus one. It is known that every tree decomposition can be normalized in linear time without increasing the width. The *treewidth* of graph G, denoted as $tw(G)$, is the minimum width over all tree decompositions of G. For arbitrary but fixed $w \geq 1$, it is feasible in linear time to decide if a graph has treewidth $\leq w$ and, if so, to compute a tree decomposition of width w [Bodlaender, 1996].

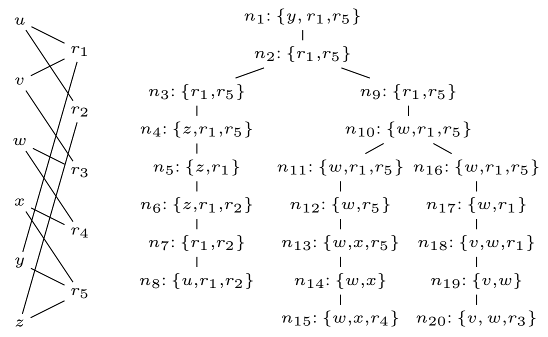

Figure 1: Incidence graph G_P of example program P (left) and a *normalized tree decomposition* $\mathcal{T}$ of G_P.

Tree Decompositions of Programs. To build tree decompositions for programs, we shall use incidence graphs.[1] Given a program R, such a graph has as vertices $R \cup At(R)$, and as edges all pairs (a, r) with an atom a appearing in a rule r of R. In case of normalized tree decompositions, we distinguish between six types of nodes: atom introduction (AI), rule introduction (RI), atom removal (AR), rule removal (RR), branch (B), and leaf (L) nodes. The first four types are usually augmented with the element e (either an atom or rule) which is removed or added compared to the bag of the child node.

Example 2.2 *Figure 1 shows the incidence graph G_P of program P and a normalized tree decomposition $\mathcal{T}$ of G_P having width 2. Indeed, we have $tw(G_P) = 2$, so $\mathcal{T}$ has optimal width. Examples for node types are n_8 as (L) node, n_7 as (u-AR) node, n_6 as (z-AI) node, n_5 as (r_2-RR) node, n_4 as (r_5-RI) node, and n_2 as (B) node.*

3 The Dynamic Programming Approach

For this section, let $\mathcal{T} = (T, \beta)$ be a normalized tree decomposition of the incidence graph of a given program R. For $M \subseteq R \cup At(R)$, we use A_M (resp. R_M) as a shorthand for $At(R) \cap M$ (resp. $R \cap M$). We refer to the root node of T as rt. For a node $n \in T$, T_n denotes the subtree of T rooted at n. We use $A_{(n)}$ (resp. $R_{(n)}$) to denote the set of all atoms (resp. rules) which appear in $\bigcup_{m \in T_n} \beta(m)$ (i.e. in some bag of the tree T_n); moreover, $A_{[n]}$ (resp. $R_{[n]}$) abbreviates $A_{(n)} \setminus \beta(n)$ (resp. $R_{(n)} \setminus \beta(n)$), i.e. the set of atoms (resp. rules) which appear only in bags "below" the root of T_n.

We proceed as follows: First, we define the mathematical objects (*tree interpretations*) which will underly our algorithms. We construct a mapping $\mathcal{E}(\cdot)$ from tree interpretations to (standard) interpretations and observe that a certain subset S of the tree interpretations characterizes the answer sets of R. However, we never compute $\mathcal{E}(\cdot)$ explicitly. Instead, we define a relation $\prec_{\mathcal{T}}$ along the structure of T, in order to efficiently compute S in a bottom-up manner via so-called *tree models*, a subset of the tree interpretations. Finally, we show

[1]See [Samer and Szeider, 2007] for other possible types of graphs and a discussion why incidence graphs are favorable.

how to use this method to count and enumerate the answer sets of program R from a given tree decomposition $\mathcal{T}$ for R.

3.1 Tree interpretations

Definition 3.1 *A* tree interpretation *for $\mathcal{T}$ ($\mathcal{T}$-*interpretation, *for short) is a tuple $(n, M, \mathcal{C})$ where $n \in T$ is a node, $M \subseteq \beta(n)$ is called* assignment, *and $\mathcal{C} \subseteq 2^{\beta(n)}$ is called* certificate.

The basic intuition behind $\mathcal{T}$-interpretations is as follows: the assignment M of a $\mathcal{T}$-interpretation $(n, M, \mathcal{C})$ contains an interpretation A_M over $A_{\beta(n)}$ (implicitly it refers to interpretations I over $A_{(n)}$) together with rules $r \in R_{\beta(n)}$ satisfied by I, i.e. $I \models_{A_{(n)}} r$. Certificate $\mathcal{C}$ can be understood as a set of assignments and carries interpretations (together with satisfied rules in $(R_{\beta(n)})^I$) which are in a certain subset-relation to M. The following definitions make this more precise.

Definition 3.2 *Given $n \in T$ and $I, J \subseteq A_{(n)}$, define $\text{SAT}_n(I) = \{r \mid r \in R_{(n)}, I \models_{A_{(n)}} r\}$ and $\text{RSAT}_n(J, I) = \{r \mid r \in R_{(n)}$ s.t. $J \models_{A_{(n)}} r$ or $\text{B}^-(r) \cap I \neq \emptyset\}$.*

Roughly speaking, $\text{SAT}_n(I)$ yields those rules of R which occur in bags of the subtree T_n and are satisfied by I. Analogously, $\text{RSAT}_n(J, I)$ yields such rules which are either satisfied by J or not contained in the reduct R^I (thus we can view them as satisfied by J in a trivial way).

Definition 3.3 *Let $\theta = (n, M, \mathcal{C})$ be a $\mathcal{T}$-interpretation, $I \subseteq A_{(n)}$, and let $R^* = R_M \cup R_{[n]}$. We define*

$$e_n(M) = \{A_M \cup K \mid K \subseteq A_{[n]}, \text{SAT}_n(A_M \cup K) = R^*\};$$
$$re_n(M, I) = \{A_M \cup K \mid K \subseteq A_{[n]}, \text{RSAT}_n(A_M \cup K, I) = R^*\}.$$

Moreover, $\mathcal{C}$ is called valid *wrt. I in θ, if, for each $N \subseteq \beta(n)$, it holds that $N \in \mathcal{C}$ iff there exists $J \in re_n(N, I)$ s.t. $J \subset I$.*

The rationale behind $e_n(M)$ is to yield those extensions of the interpretation A_M stored in the assignment M of a $\mathcal{T}$-interpretation $\theta = (n, M, \mathcal{C})$ to an interpretation I over $A_{(n)}$ (i.e. over all atoms occurring in bags of T_n), such that the rules R_M plus *all* rules in $R_{[n]}$ (i.e. all rules occurring in bags of T_n, but below n) are satisfied by I. A similar idea is followed by $re_n(M, I)$ which additionally takes the concept of reduct into account.

We are now prepared to define the mapping $\mathcal{E}(\cdot)$ and we shall see that for certain $\mathcal{T}$-interpretations θ, $\mathcal{E}(\theta) \subseteq \mathcal{AS}(R)$.

Definition 3.4 *For a $\mathcal{T}$-interpretation $\theta = (n, M, \mathcal{C})$, let*

$$\mathcal{E}(\theta) = \{I \mid I \in e_n(M) \text{ and } \mathcal{C} \text{ is valid wrt. } I \text{ in } \theta\}.$$

Definition 3.5 *A $\mathcal{T}$-interpretation $(n, M, \mathcal{C})$ is called a* root model *for $\mathcal{T}$ iff $n = rt$, $R_M = R_{\beta(n)}$ and, for each $N \in \mathcal{C}$, $R_N \subset R_M$.*

Theorem 3.6 *Let Θ be the set of all root models for $\mathcal{T}$. Then, $\mathcal{AS}(R) = \bigcup_{\theta \in \Theta} \mathcal{E}(\theta)$.*

Proof. We only show the $\subseteq$-direction. The $\supseteq$-direction is proved analogously. We write A_{rt} as shorthand for $A_{\beta(rt)}$ and likewise R_{rt} for $R_{\beta(rt)}$. Let $I \in \mathcal{AS}(R)$ and let $\theta = (rt, M, \mathcal{C})$ with $M = (I \cap A_{rt}) \cup R_{rt}$ and $\mathcal{C} = \{N \subseteq \beta(rt) \mid \exists J \in re_{rt}(N, I)$ s.t. $J \subset I\}$. Note that $\mathcal{C}$ is thus valid wrt. I in θ. It remains to show (i) $I \in e_{rt}(M)$ and (ii) $\theta \in \Theta$. (i) holds since $R_M \cup R_{[rt]} = R_{rt} \cup R_{[rt]} = R$, and $I \models R$ by assumption $I \in \mathcal{AS}(R)$. For (ii), we have $R_M = R_{rt}$ by definition. We show that for each $N \in \mathcal{C}$, $R_N \subset R_{rt}$ holds. Suppose this is not the case, i.e. let $N \in \mathcal{C}$, such that $R_N = R_{rt}$. By definition of $re_{rt}(N, I)$, there exists a $J \subset I$, such that for each $r \in R = R_{rt} \cup R_{[rt]}$, either $J \models r$ or $\text{B}^-(r) \cap I \neq \emptyset$. Hence, $J \models R^I$, a contradiction to $I \in \mathcal{AS}(R)$. □

3.2 Tree models

Theorem 3.6 tells us that tree interpretations θ which satisfy $\mathcal{E}(\theta) \neq \emptyset$ are of particular interest.

Definition 3.7 *A $\mathcal{T}$-interpretation θ is called* tree model *of $\mathcal{T}$ ($\mathcal{T}$-*model, *for short) iff $\mathcal{E}(\theta) \neq \emptyset$. A $\mathcal{T}$-model that is also a root model for $\mathcal{T}$ is called $\mathcal{T}$-*root-model.

For leaf nodes n, tree models can be determined as follows. For every $M \subseteq \beta(n)$, we either have $e_n(M) = A_M$ in case $R_M = \{r \mid r \in R_{\beta(n)}, A_M \models_{A_{\beta(n)}} r\}$, or $e_n(M) = \emptyset$, otherwise. Hence, to compute all $\mathcal{T}$-models for a leaf node n, one considers each $A_M \subseteq A_{\beta(n)}$ and determines $R_M = \{r \mid r \in R_{\beta(n)}, A_M \models_{A_{\beta(n)}} r\}$; then $A_M \cup R_M$ yields the assignment M for a $\mathcal{T}$-model $(n, M, \mathcal{C})$. Certificate $\mathcal{C}$ is given by all $J \subset A_M$ together with the rules $r \in R_{\beta(n)}$, for which either $J \models_{A_{\beta(n)}} r$ or $\text{B}^-(r) \cap A_M \neq \emptyset$ holds.

Example 3.8 *Take our example tree-decomposition in Figure 1 and consider leaf node n_8. We have $\beta(n_8) = \{u, r_1, r_2\}$. Recall $r_1 = u \leftarrow v, y$ and $r_2 = z \leftarrow u$. We first set u to true. This satisfies r_1, i.e. $\{u\} \models_{\{u\}} r_1$. For the corresponding certificate, there is only one possibility: We set u to false and observe that this satisfies r_2. Hence, $(n_8, \{u, r_1\}, \{\{r_2\}\})$ is a $\mathcal{T}$-model. Another $\mathcal{T}$-model is $(n_8, \{r_2\}, \emptyset)$ and these are the only $\mathcal{T}$-models for n_8.*

We next define a relation $\prec_{\mathcal{T}}$ between $\mathcal{T}$-interpretations. The concrete definition depends on the node type. We first give the definition for the removal and introduction nodes.

Definition 3.9 *For $\mathcal{T}$-interpretations $\theta = (n, M, \mathcal{C})$ and $\theta' = (n', M', \mathcal{C}')$, we have $\theta' \prec_{\mathcal{T}} \theta$ iff n has a single child n', and (depending on the node type of n) the conditions as depicted in the table of Figure 2 are fulfilled.*[2]

Example 3.10 *Recall $\mathcal{T}$-model $\theta_8 = (n_8, \{u, r_1\}, \{\{r_2\}\})$. To obtain $\mathcal{T}$-models for n_7 (which is a (u-AR) node) from θ_8 we have to remove all occurrences of u in θ_8, i.e. we get $\theta_8 \prec_{\mathcal{T}} (n_7, \{r_1\}, \{\{r_2\}\}) = \theta_7$. Next, we consider n_6 which is a (z-AI) node. We have two possibilities. First, we set the new atom z to true, i.e. we get as assignment $\{z, r_1, r_2\} = \{r_1\} +_{n_6} z$ (for the definition of operators as $+_n$, see Figure 2). The certificate consists of $N_1 = \{r_1\} = \{r_1\} \times_{n_6} z$, $N_2 = \{z, r_2\} = \{r_2\} +_{n_6} z$ and $N_3 = \{r_2\} = \{r_2\} \times_{n_6} z$. Hence, $\theta_7 \prec_{\mathcal{T}} (n_6, \{z, r_1, r_2\}, \{N_1, N_2, N_3\}) = \theta_6^1$ (also note here that $R^-_{z,n_6} = \emptyset$). Second, we set the new atom z to false, which yields $\theta_6^2 = (n_6, \{r_1\}, \{N_3\})$. For the next node n_5, which is of type (r_2-RR), thus only θ_6^1 plays a role since its assignment contains r_2 and we obtain $\theta_6^1 \prec_{\mathcal{T}} (n_5, \{z, r_1\}, \{\{z\}, \emptyset\}) = \theta_5$ (the set $\{r_1\}$ from the certificate of θ_6^1 also drops out, since it does not contain r_2). Finally, we use θ_5 to compute $\mathcal{T}$-models of n_4, an (r_5-RI)*

[2]Note that in case n is an (AI)-node, there are two ways how θ' and θ can be related to each other.

node-type of n	conditions	
(a-AR)	$M = M' \setminus \{a\}$	$\mathcal{C} = \{C \setminus \{a\} \mid C \in \mathcal{C}'\}$
(r-RR)	$r \in M', M = M' \setminus \{r\}$	$\mathcal{C} = \{C \setminus \{r\} \mid C \in \mathcal{C}', r \in C\}$
(a-AI)	$M = M' +_n a$	$\mathcal{C} = \{(M' \times_n a) \cup R^-_{a,n}\} \cup \{(C +_n a) \cup R^-_{a,n}, (C \times_n a) \cup R^-_{a,n} \mid C \in \mathcal{C}'\}$
(a-AI)	$M = M' \times_n a$	$\mathcal{C} = \{C \times_n a \mid C \in \mathcal{C}'\}$
(r-RI)	$M = M' \uplus_n r$	$\mathcal{C} = \begin{cases} \{C \cup \{r\} \mid C \in \mathcal{C}'\} & \text{if } \mathrm{B}^-(r) \cap M \neq \emptyset \\ \{C \uplus_n r \mid C \in \mathcal{C}'\} & \text{otherwise} \end{cases}$

$$\begin{aligned} M +_n a &= M \cup \{a\} \cup \{r \in R_{\beta(n)} \mid a \in r\} & M \uplus_n r &= \begin{cases} M \cup \{r\} & \text{if } r \cap (A_M \cup \overline{(A_{\beta(n)} \setminus A_M)}) \neq \emptyset \\ M & \text{otherwise} \end{cases} \\ M \times_n a &= M \cup \{r \in R_{\beta(n)} \mid \bar{a} \in r\} & R^-_{a,n} &= \{r \in R_{\beta(n)} \mid a \in \mathrm{B}^-(r)\} \end{aligned}$$

Figure 2: Conditions for $(n', M', \mathcal{C}') \prec_{\mathcal{T}} (n, M, \mathcal{C})$.

node. We observe that $\{z, r_1, r_5\} = \{z, r_1\} \uplus_{n_4} r_5$ since $r_5 = x \leftarrow \neg y, \neg z$ contains z negated. For the same reason, r_5 is added to all sets of the certificate. We obtain $\theta_5 \prec_{\mathcal{T}} (n_4, \{z, r_1, r_5\}, \{\{z, r_5\}, \{r_5\}\})$. We refer already to Figure 4 (which is explained in detail later) to follow this sequence of $\mathcal{T}$-models.

For branch nodes, we partially extend (with a slight abuse of notation) $\prec_{\mathcal{T}}$ to a ternary relation as follows.

Definition 3.11 *For $\mathcal{T}$-interpretations $\theta = (n, M, \mathcal{C})$, $\theta_1 = (n_1, M_1, \mathcal{C}_1)$, $\theta_2 = (n_2, M_2, \mathcal{C}_2)$ we have $(\theta_1, \theta_2) \prec_{\mathcal{T}} \theta$ iff the following conditions hold: (1) n_1 and n_2 are the two children of n; (2) $A_{M_1} = A_{M_2}$ and $M = M_1 \cup M_2$; (3) $\mathcal{C}$ is given by the set $(\mathcal{C}_1 \bowtie \mathcal{C}_2) \cup (\{M_1\} \bowtie \mathcal{C}_2) \cup (\mathcal{C}_1 \bowtie \{M_2\})$; where $\mathcal{C} \bowtie \mathcal{C}'$ is defined as $\{C \cup C' \mid C \in \mathcal{C}, C' \in \mathcal{C}', A_C = A_{C'}\}$.*

Example 3.12 *Consider $\theta' = (n_{11}, \{w, r_5\}, \{\emptyset, \{w\}\})$ and $\theta'' = (n_{16}, \{w\}, \{\emptyset, \{r_1\}\})$. Both are $\mathcal{T}$-models. We determine a θ for branch node n_{10}, such that $(\theta', \theta'') \prec_{\mathcal{T}} \theta$. Such θ exists (since w is true in the assignment of both θ' and θ''), and is of the form $(n_{10}, \{w, r_5\}, \mathcal{C})$ with $\mathcal{C} = \{\emptyset, \{w\}, \{r_1\}\}$ obtained as follows: $\{\emptyset, \{w\}\} \bowtie \{\emptyset, \{r_1\}\} = \{\emptyset, \{r_1\}\}$; $\{\{w, r_5\}\} \bowtie \{\emptyset, \{r_1\}\} = \emptyset$; $\{\emptyset, \{w\}\} \bowtie \{\{w\}\} = \{\{w\}\}$.*

The following lemma is central.

Lemma 3.13 *Let $\theta = (n, M, \mathcal{C})$ be a $\mathcal{T}$-interpretation. If n is of type (RR), (RI), (AR), or of type (a-AI) and $a \notin M$, then $\mathcal{E}(\theta) = \bigcup_{\theta' \prec_{\mathcal{T}} \theta} \mathcal{E}(\theta')$. If n is of type (a-AI) and $a \in M$, then $\mathcal{E}(\theta) = \bigcup_{\theta' \prec_{\mathcal{T}} \theta} (\mathcal{E}(\theta') \cup \{a\})$. If n is a branch node, then $\mathcal{E}(\theta) = \bigcup_{(\theta_1, \theta_2) \prec_{\mathcal{T}} \theta} \{I_1 \cup I_2 \mid I_1 \in \mathcal{E}(\theta_1), I_2 \in \mathcal{E}(\theta_2)\}$.*

Proof (Sketch). Due to space reasons, we only show the case of an (r-RR) node here. The other cases are similar.

In what follows, let n' be the child of n and $M' = M \cup \{r\}$. First, note that $I \in e_n(M)$ iff $I \in e_{n'}(M')$. Indeed, since $\{r\} = R_{[n]} \setminus R_{[n']} = R_{M'} \setminus R_M$, $R_{M'} \cup R_{[n']} = R_M \cup R_{[n]}$ and thus $\mathrm{SAT}_n(I) = \mathrm{SAT}_{n'}(I)$. Similarly, one can show that $J \in re_n(M, I)$ iff $J \in re_{n'}(M', I)$, for any $I, J \subseteq A_{(n)}$.

Let $I \in \mathcal{E}(\theta)$ and $\theta' = (n', M', \mathcal{C}_1 \cup \mathcal{C}_2)$, where $\mathcal{C}_1 = \{C \cup \{r\} \mid C \in \mathcal{C}\}$ and $\mathcal{C}_2 = \{N \subseteq \beta(n') \mid r \notin N, \exists J \subset I \text{ s.t. } J \in re_{n'}(N, I)\}$. In fact, $\theta' \prec_{\mathcal{T}} \theta$ holds. We show $I \in \mathcal{E}(\theta')$. By the observation above, we have $I \in e_{n'}(M')$. To show that $\mathcal{C}_1 \cup \mathcal{C}_2$ is valid wrt. I in θ', suppose it is not the case, i.e. there exists an $N' \subseteq \beta(n')$ such that either $N' \in \mathcal{C}_1 \cup \mathcal{C}_2$ or there exists a $J \in re_{n'}(N', I)$, such that $J \subset I$. By definition of $\mathcal{C}_2$, $r \in N'$ has to hold. We know $J \in re_n(N, I)$ iff $J \in re_{n'}(N', I)$. But then, for $N = N' \setminus \{r\}$, either $N \in \mathcal{C}$ or there exists a $J \in re_n(N, I)$, such that $J \subset I$. This yields that $\mathcal{C}$ is not valid wrt. I in θ. A contradiction to $I \in \mathcal{E}(\theta)$.

Let $I \in \mathcal{E}(\theta')$ for a $\theta' = (n', M', \mathcal{C}')$, such that $\theta' \prec_{\mathcal{T}} \theta$. By definition of $\prec_{\mathcal{T}}$, M' is of the form $M \cup \{r\}$. Using our previous observation, we get $I \in e_n(M)$. Since $I \in \mathcal{E}(\theta')$, $\mathcal{C}'$ is valid wrt. I in θ', and we know $\mathcal{C} = \{C \setminus \{r\} \mid C \in \mathcal{C}', r \in C\}$, since $\theta' \prec_{\mathcal{T}} \theta$. We show that $\mathcal{C}$ is valid wrt. I in θ, which will imply $I \in \mathcal{E}(\theta)$. Again, suppose $\mathcal{C}$ is not valid wrt. I in θ, i.e. there exists an $N \subseteq \beta(n)$, such that either $N \in \mathcal{C}$ or there exists $J \in re_n(N, I)$, such that $J \subset I$. We know that then $J \in re_{n'}(N', I)$ as well, which is in contradiction to the assumption that $\mathcal{C}'$ is valid valid wrt. I in θ'. ☐

Corollary 3.14 *Let $\theta, \theta', \theta''$ be $\mathcal{T}$-interpretations, such that $\theta' \prec_{\mathcal{T}} \theta$ (resp. $(\theta', \theta'') \prec_{\mathcal{T}} \theta$). Then, θ is a $\mathcal{T}$-model iff θ' is $\mathcal{T}$-model (resp. both θ' and θ'' are $\mathcal{T}$-models).*

Theorem 3.15 *Deciding $\mathcal{AS}(R) \neq \emptyset$ can be done in time $O(f(w) \cdot |R|)$, where w denotes the treewidth of R and f is a function that only depends on w but not on R.*

Proof (Sketch). Corollary 3.14 suggests the following algorithm: first, we establish the $\mathcal{T}$-models of leaf nodes, then we compute all remaining $\mathcal{T}$-models via $\prec_{\mathcal{T}}$ in a bottom-up manner. As soon as we have the $\mathcal{T}$-models for the root node, we check whether they include also a root model for $\mathcal{T}$.

The effort needed for processing a leaf node as well as for the transition from the child node(s) to the parent only depends on the treewidth but not on R. Moreover, the size of $\mathcal{T}$ is linearly bounded by the size of R. Hence, this algorithm has the desired time bound. The correctness of this algorithm immediately follows from Theorem 3.6, i.e.: $\mathcal{AS}(R) \neq \emptyset$ holds iff there exists at least one $\mathcal{T}$-root-model. ☐

Theorem 3.15 is the desired FPT result for the ASP consistency problem. Indeed, if the treewidth w is bounded by a constant, then $\mathcal{AS}(R) \neq \emptyset$ can be decided in linear time.

Example 3.16 *The $\mathcal{T}$-models for our running example are depicted in Figure 4, where we grouped them wrt. their nodes and along the structure of the tree of $\mathcal{T}$. $\mathcal{T}$-models which contribute to the single $\mathcal{T}$-root-model $(n_1, \{r_1, r_2\}, \{\{r_1\}\})$ are marked with "+". Following the branches and using the $\mathcal{T}$-models marked with "+", one can see that the used atoms* *are v, w, x which exactly yields the answer set of our example program P. For illustration, we depict for those $\mathcal{T}$-models θ also the set $\mathcal{E}(\theta)$ in the last column (as mentioned before, this set is not explicitly computed). Finally, $\#$ refers to a function which we define in the next section for counting answer sets.*

3.3 Counting and Enumerating Answer Sets

The following observation is important and together with Lemma 3.13 lays the foundation for our counting algorithm.

Lemma 3.17 *For two distinct $\mathcal{T}$-interpretations $\theta_1 = (n, M_1, \mathcal{C}_1)$ and $\theta_2 = (n, M_2, \mathcal{C}_2)$, $\mathcal{E}(\theta_1) \cap \mathcal{E}(\theta_2) = \emptyset$ holds.*

Proof (Sketch). Suppose to the contrary that there exists an assignment $I \in \mathcal{E}(\theta_1) \cap \mathcal{E}(\theta_2)$. We show that then $\theta_1 = \theta_2$. By definition of $\mathcal{E}(\cdot)$, $A_{M_1} = A_{M_2}$. Moreover, there exists a $K \subseteq A_{[n]}$ such that $\text{SAT}_n(A_{M_i} \cup K) = R_{M_i} \cup R_{[n]}$. By $R_{M_i} \cap R_{[n]} = \emptyset$ and $R_{M_1} \cup R_{[n]} = R_{M_2} \cup R_{[n]}$ we conclude $R_{M_1} = R_{M_2}$. Thus $M_1 = M_2$. Finally, $\mathcal{C}_1 = \{N \subseteq \beta(n) \mid \exists J \in re_n(N, I), \text{ s.t. } J \subseteq I\} = \mathcal{C}_2$ follows by definition. Hence, $\theta_1 = \theta_2$. □

Next, we recursively define a mapping from $\mathcal{T}$-interpretations to numbers.

Definition 3.18 *Let θ be a $\mathcal{T}$-interpretation for node n. If θ is not a $\mathcal{T}$-model, let $\#(\theta) = 0$, otherwise let*

$$\#(\theta) = \begin{cases} 1 & \text{if } n \text{ is leaf node} \\ \sum_{\theta' \prec_{\mathcal{T}} \theta} \#(\theta') & \text{if } n \text{ has one child} \\ \sum_{(\theta', \theta'') \prec_{\mathcal{T}} \theta} \#(\theta') \cdot \#(\theta'') & \text{if } n \text{ is branch node} \end{cases}$$

Using Theorem 3.6 and Lemma 3.13 and 3.17, we obtain

Theorem 3.19 *Let Θ be the set of all root models of $\mathcal{T}$. Then, $|\mathcal{AS}(R)| = \sum_{\theta \in \Theta} \#(\theta)$.*

Using the same algorithm as sketched in the proof of Theorem 3.15, plus keeping track of the $\#$ values for $\mathcal{T}$-models, we immediately obtain the following result.

Theorem 3.20 *Assuming unit cost for arithmetic operations, $|\mathcal{AS}(R)|$ can be computed in time $O(f(w) \cdot |R|)$, where $w = tw(R)$ and f is a function depending on w but not on R.*

For the enumeration problem, we provide in Figure 3 an algorithm which, given a $\mathcal{T}$-root-model $\theta = (n, M, \mathcal{C})$, computes the set $\mathcal{E}(\theta)$ step by step, such that each new element in $\mathcal{E}(\theta)$ requires only linear delay.

To this end, we consider for a given $\mathcal{T}$-model θ, all θ' such that $\theta' \prec_{\mathcal{T}} \theta$ (resp. all (θ', θ'') such that $(\theta', \theta'') \prec_{\mathcal{T}} \theta$) as stored in an ordered list, and for each such list we use an internal pointer p_θ. Function initialize resets all pointers to the first θ' (resp. to the first pair (θ', θ'')) in such a list. Function get_current(θ) yields the object p_θ currently refers to. Function get_next(θ) either moves the pointer to the next θ' (resp. to the next pair (θ', θ'')) in the list and returns 0; or in case the last element was already reached, it resets p_θ to the first element in the list and returns 1. Due to the space restrictions, we cannot discuss the algorithm in detail. However, we note that in general one $\mathcal{T}$-root-model may refer to multiple answer sets. Thus we have to reconstruct all possible such models by traversing the tree downwards – and collect all atoms set to true in at least some assignment – for all possible combinations of $\mathcal{T}$-models related via $\prec_{\mathcal{T}}$.

```
Function getAS($\theta, I, flag$)
input: ($\mathcal{T}$-interpretation $\theta = (n, M, \mathcal{C})$, interpretation, Boolean)
return: (interpretation, Boolean)
begin
  if $n$ is leaf-node then return$(I, flag)$;
  if $n$ is branch node then
    $(\theta', \theta'') = $ get_current$(\theta)$;
    $(J, flag') = $ getAS$(\theta', I, flag)$;
    $(K, flag'') = $ getAS$(\theta'', J, flag')$;
  else
    $\theta' = (n', M', \mathcal{C}') = $ get_current$(\theta)$;
    $(K, flag'') = $ getAS$(\theta', I, flag)$;
    if $n$ is of type ($e$-AR) and $e \in M'$ then $K = K \cup \{e\}$;
  endif
  if $flag''$ then return $(K, $ get_next$(\theta))$;
  return $(K, 0)$;
end
Program enumerateAS
begin
  for each $\mathcal{T}$-root-model $\theta = (rt, M, \mathcal{C})$ do
    initialize;
    repeat
      $(I, flag) = $ getAS$(\theta, A_M, 1)$;
      output $I$;
    until $flag$;
  done
end
```

Figure 3: Program enumerateAS.

Our ASP algorithm first computes all $\mathcal{T}$-models as sketched in the proof of Theorem 3.15 (within this algorithm, we already keep track of the information used later by the pointers p_θ). Then the program enumerateAS iterates through all $\mathcal{T}$-root-models and outputs the corresponding answer sets.

Theorem 3.21 *Program enumerateAS works in space $O(f(w) \cdot |R|)$ and outputs all elements in $\mathcal{AS}(R)$ with delay $O(f(w) \cdot |R|)$, where w denotes the treewidth of R and f is a function that only depends on w but not on R.*

4 Implementation and Results

For our implementation, we have chosen Haskell, a programming language with lazy semantics [Josephs, 1989], thus the desired linear delay is implicit in the evaluation strategy of the language (a computation is only executed when needed). We call our prototype LAPS (lazy answer-set programming system).

The performance of our straightforward implementation is unprecedented for counting, and is very competitive at low treewidths (up to six) for enumerating the answer sets. Given its early stage of development, LAPS has a high potential for further improvements. We split the evaluation into four steps: (1) parse a disjunctive logic program and generate the data structures for our target language. (2) build the incidence graph of the program and decompose the graph using heuristic methods [Dermaku *et al.*, 2005]. The decomposition is then provided as a data structure for the target language. (3) all parts are merged with the algorithm, compiled and (4) executed. Figure 5 summarizes the runtime behavior of LAPS (assuming the tree decomposition is already given) compared to DLV on a set of randomly generated programs. The first

n_1: yr_1r_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	3	r_1	$\emptyset$	1	
2	1,2	r_1	r_1	2	
+3	6	r_1r_5	r_1	1	vwx
4	4,5,7	r_1r_5	r_1r_5	8	
5	3	yr_1r_5	r_1r_5	1	
6	4	yr_1r_5	r_1r_5, yr_1r_5	2	
7	5	yr_1r_5	r_1r_5, yr_1r_5, yr_5	3	
8	2	yr_5	r_1r_5, yr_1r_5	1	
9	1,6,7	yr_5	r_1r_5, yr_1r_5, yr_5	5	

n_2: r_1r_5

j	(i,k)	M	$\mathcal{C}$	#	$\mathcal{E}$
1	(1,1)	ϵ	ϵ, r_1	1	
2	(1,2)	ϵ	r_1	1	
3	(1,3)	r_1	$\emptyset$	1	
4	(2,3) (3,3)	r_1r_5	r_1r_5	2	
5	(2,1) (2,2) (2,4)	r_1r_5	r_1r_5, r_5	3	
+6	(1,4)	r_5	ϵ,r_1	1	vwx
7	(3,1) (3,2) (3,4)	r_5	r_1r_5, r_5	3	

n_3: r_1r_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
+1	1	ϵ	$\emptyset$	1	ϵ
2	2	r_1r_5	r_5	1	
3	3	r_5	r_5	1	

n_4: zr_1r_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
+1	1	ϵ	$\emptyset$	1	ϵ
2	3	zr_1r_5	r_5, zr_5	1	
3	2	zr_5	r_5	1	

n_5: zr_1

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
+1	2	ϵ	$\emptyset$	1	ϵ
2	4	z	ϵ	1	
3	3	zr_1	ϵ, z	1	

n_6: zr_1r_2

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	r_1	r_2	1	
+2	2	r_2	$\emptyset$	1	ϵ
3	1	zr_1r_2	r_1, r_2, zr_2	1	
4	2	zr_2	r_2	1	

n_7: r_1r_2

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	r_1	r_2	1	
+2	2	r_2	$\emptyset$	1	ϵ

n_8: ur_1r_2

j	M	$\mathcal{C}$	#	$\mathcal{E}$
1	ur_1	r_2	1	
+2	r_2	$\emptyset$	1	ϵ

n_9: r_1r_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	3	ϵ	ϵ, r_1	1	
2	1	ϵ	r_1	1	
3	2	r_1	$\emptyset$	1	
+4	4	r_5	ϵ,r_1	1	vwx

n_{10}: wr_1r_5

j	(i,k)	M	$\mathcal{C}$	#	$\mathcal{E}$
1	(1,1)	ϵ	r_1	1	
2	(1,2)	r_1	$\emptyset$	1	
3	(2,3)	w	ϵ, r_1	1	
+4	(3,3)	wr_5	ϵ,r_1, w	1	vwx

n_{11}: wr_1r_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	ϵ	$\emptyset$	1	
2	2	w	ϵ	1	
+3	3	wr_5	ϵ,w	1	wx

n_{12}: wr_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	ϵ	$\emptyset$	1	
2	2	w	ϵ	1	
+3	3	wr_5	ϵ,w	1	wx

n_{13}: wxr_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	ϵ	$\emptyset$	1	
2	2	w	ϵ	1	
+3	3	wxr_5	ϵ,w	1	wx

n_{14}: wx

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	ϵ	$\emptyset$	1	
2	2	w	ϵ	1	
+3	3	wx	ϵ,w	1	wx

n_{15}: wxr_4

j	M	$\mathcal{C}$	#	$\mathcal{E}$
1	r_4	$\emptyset$	1	
2	wr_4	r_4	1	
+3	wxr_4	r_4, wr_4, x	1	wx
4	x	r_4	1	

n_{16}: wr_1r_5

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	ϵ	r_1	1	
2	2	r_1	$\emptyset$	1	
+3	3	w	ϵ,r_1	1	vw

n_{17}: wr_1

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	2	ϵ	r_1	1	
2	1	r_1	$\emptyset$	1	
+3	3	w	ϵ,r_1	1	vw

n_{18}: vwr_1

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	r_1	$\emptyset$	1	
2	2	v	r_1	1	
+3	3	vw	r_1, v	1	vw

n_{19}: vw

j	i	M	$\mathcal{C}$	#	$\mathcal{E}$
1	1	ϵ	$\emptyset$	1	
2	2	v	ϵ	1	
+3	3	vw	ϵ,v	1	vw

n_{20}: vwr_3

j	M	$\mathcal{C}$	#	$\mathcal{E}$
1	r_3	$\emptyset$	1	
2	vr_3	r_3	1	
+3	vwr_3	r_3, vr_3, w	1	vw
4	w	r_3	1	

Figure 4: The $\mathcal{T}$-models of the tree decomposition $\mathcal{T}$ for our running example program P. We abbreviate sets of atoms and rules via strings. A list of string stands for a set of sets, e.g. r_4, wr_4, x denotes $\{\{r_4\}, \{w, r_4\}, \{x\}\}$. The $\prec_\mathcal{T}$ relation for $\theta_i^{n'} \prec_\mathcal{T} \theta_j^n$ (resp. for $(\theta_i^{n'}, \theta_k^{n''}) \prec_\mathcal{T} \theta_j^n$) can be read off columns "j" and "i" (resp. "(i,k)") in the table of node n.

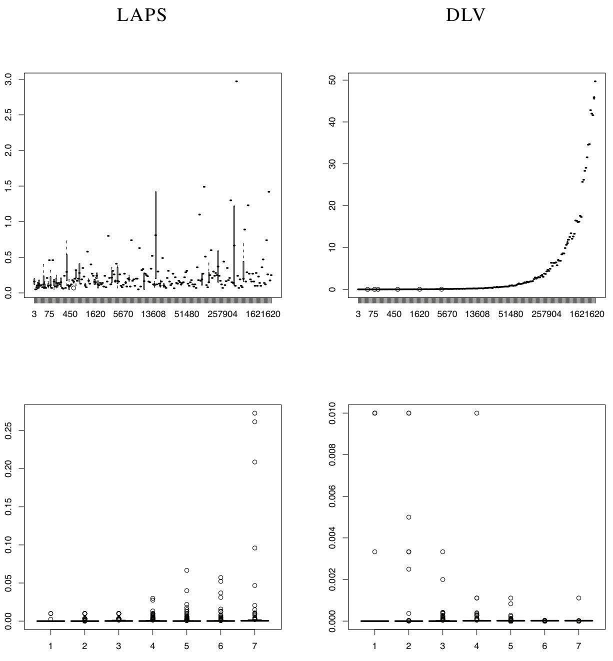

Figure 5: Comparison of the runtime behavior.

row shows the time required to *count* the number of answer sets with increasing number of answer sets (here we fixed the treewidth to 5). Clearly DLV's runtime is dependent on the number of answer sets, whereas LAPS' runtime is not affected (excluding time required to handle very large integers). The second row shows the runtime required for enumerating the answer sets (*time per answer set*) with increasing treewidth. Here, LAPS' runtime increases quickly with larger treewidth whereas DLV even seems to benefit from larger treewidth. The latter effect is due to the fact that our randomly generated programs tend to have more answer sets when the treewidth increases which – in case of DLV – decreases the average cost per answer set. Tests using smodels instead of DLV resulted in a very similar runtime behavior.

5 Related Work and Conclusion

Another FPT result for ASP is due to Lin and Zhao [2004], who use the number of cycles in the (directed) dependency graph as parameter. A further interesting parameter here is the number of loops [Ferraris *et al.*, 2006] of a program. We note that programs with an unbounded number of cycles and/or loops can still have low treewidth. Consider $P_1 = \{a_i \leftarrow \neg b_i;\ b_i \leftarrow \neg a_i \mid 1 \leq i \leq n\}$ or $P_2 = \{a_{n+1} \leftarrow a_1, a_i \leftarrow a_{i+1} \mid 1 \leq i \leq n\}$. Both have treewidth 2, but the number of cycles (P_1), or loops (P_2), clearly depends on n.

The work most closely related to ours is by Samer and Szeider [2007], where the #SAT problem in case of bounded treewidth was solved by dynamic programming. We extend their approach to the counting problem (and also the enumeration problem) of ASP. To this end, we have to introduce sophisticated, additional data structures, which ultimately allow us to distinguish between arbitrary and minimal (wrt. to the reduct) models of a given program. Two related problems are constraint satisfaction problems (CSPs) and conjunctive query (CQ) evaluation, for which, apart from treewidth, further methods based on structural decomposition have been used to construct efficient algorithms [Gottlob *et al.*, 2000; Chekuri and Rajaraman, 2000]. These methods usually also work by a bottom-up traversal of a tree structure. As with #SAT, the data propagated up the tree structure is much simpler than in case of ASP solving. On the other hand, the idea of postprocessing by a top-down traversal in order to compute all solutions is also present in the context of CQ evaluation. Recently, dynamic programming has also been applied to logic programming in the context of query answering over Semantic Web data [Ruckhaus *et al.*, 2008]. In this work, dynamic programming is applied to the computation of an optimal join order for CQ evaluation over deductive databases.

To summarize, we introduced in this work novel algorithms for ASP consistency, as well as for counting and enumerating answer sets. The algorithm runs in linear time (resp. with linear delay) if the treewidth of the logic programs is bounded by a constant. Our experiments indicate that this technique may lead to a promising alternative for evaluating ASP programs, if the treewidth remains low. Since tree decompositions of the logic programs are required for the algorithm, our approach will greatly profit from any future progress in the research for efficient tree-decomposition algorithms.

Future research concerns investigations to improve the performance of the proposed method. This includes concepts like balanced and non-normalized tree-decompositions. As well, we plan to study methods for parallelization, which should be easily applicable to the tree-like structure of the required computations. Finally, we want to use lower-level languages (instead of Haskell, where we rely on the compiler to do adequate optimizations automatically) for our algorithms and perform optimizations by hand.

Acknowledgement

We are very grateful to Stefan Rümmele for valuable comments on a preliminary version of this paper.

References

[Baral, 2002] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2002.

[Bodlaender, 1993] H. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–22, 1993.

[Bodlaender, 1996] H. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[Chekuri and Rajaraman, 2000] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *TCS*, 239(2):211–229, 2000.

[Courcelle, 1990] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Vol. B*, pages 193–242. Elsevier, 1990.

[Dermaku *et al.*, 2005] A. Dermaku, T. Ganzow, G. Gottlob, B. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decompositions. Technical Report DBAI-TR-2005-53, TU Vienna, 2005.

[Eiter and Gottlob, 1995] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3/4):289–323, 1995.

[Ferraris *et al.*, 2006] P. Ferraris, J. Lee, and V. Lifschitz. A generalization of the Lin-Zhao Theorem. *Ann. Math. Artif. Intell.*, 47(1-2):79–101, 2006.

[Gebser *et al.*, 2007] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczyński. The first Answer Set Programming system competition. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *LNCS*, pages 3–17. Springer, 2007.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[Gottlob *et al.*, 2000] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

[Gottlob *et al.*, 2006] G. Gottlob, R. Pichler, and F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 250–256. AAAI Press, 2006.

[Jakl *et al.*, 2008] M. Jakl, R. Pichler, S. Rümmele, and S. Woltran. Fast counting with bounded treewidth. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *LNCS*, pages 436–450. Springer, 2008.

[Josephs, 1989] M. Josephs. The semantics of lazy functional languages. *TCS*, 68(1):105–111, 1989.

[Kloks, 1994] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.

[Lin and Zhao, 2004] F. Lin and X. Zhao. On odd and even cycles in normal logic programs. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 80–85. AAAI Press, 2004.

[Marek and Truszczyński, 1999] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer, 1999.

[Niemelä, 1999] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3–4):241–273, 1999.

[Ruckhaus *et al.*, 2008] E. Ruckhaus, E. Ruiz, and M. Vidal. Query evaluation and optimization in the Semantic Web. *TPLP*, 8(3):393–409, 2008.

[Samer and Szeider, 2007] M. Samer and S. Szeider. Algorithms for propositional model counting. In *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'07)*, volume 4790 of *LNCS*, pages 484–498, Springer, 2007.

[Thorup, 1998] M. Thorup. All structured programs have small tree-width and good register allocation. *Information and Computation*, 142(2):159–181, 1998.