Answer Set Solving with Bounded Treewidth Revisited*

Johannes K. Fichte✉ (0000-0002-8681-7470), Markus Hecher (0000-0003-0131-6771),
Michael Morak, and Stefan Woltran

Institute of Information Systems, TU Wien, Wien, Austria
lastname@dbai.tuwien.ac.at

Abstract Parameterized algorithms are a way to solve hard problems more efficiently, given that a specific parameter of the input is small. In this paper, we apply this idea to the field of answer set programming (ASP). To this end, we propose two kinds of graph representations of programs to exploit their treewidth as a parameter. Treewidth roughly measures to which extent the internal structure of a program resembles a tree. Our main contribution is the design of parameterized dynamic programming algorithms, which run in linear time if the treewidth and weights of the given program are bounded. Compared to previous work, our algorithms handle the full syntax of ASP. Finally, we report on an empirical evaluation that shows good runtime behaviour for benchmark instances of low treewidth, especially for counting answer sets.

Keywords: Parameterized algorithms · Tree decompositions

1 Introduction

Parameterized algorithms [5] have attracted considerable interest in recent years and allow to tackle hard problems by directly exploiting a small parameter of the input problem. One particular goal in this field is to find guarantees that the runtime is exponential exclusively in the parameter, and polynomial in the input size (so-called fixed-parameter tractable algorithms). A parameter that has been researched extensively is treewidth [2]. Generally speaking, treewidth measures the closeness of a graph to a tree, based on the observation that problems on trees are often easier than on arbitrary graphs. A parameterized algorithm exploiting small treewidth takes a tree decomposition, which is an arrangement of a graph into a tree, and evaluates the problem in parts, via dynamic programming (DP) on the tree decomposition.

ASP [3] is a logic-based declarative modelling language and problem solving framework where solutions, so called answer sets, of a given logic program directly

* For additional details and proofs, we refer to an extended self-archived version [8]. A preliminary version of the paper was presented on the workshop TAASP'16. Research was supported by the Austrian Science Fund (FWF), Grant Y698. The first and second author are also affiliated with the University of Potsdam, Germany.

represent the solutions of the modelled problem. Jakl et al. [9] give a DP algorithm for disjunctive rules only, whose runtime is linear in the input size of the program and double exponential in the treewidth of a particular graph representation of the program structure. However, modern ASP systems allow for an extended syntax that includes, among others, weight rules and choice rules. Pichler et al. [10] investigated the complexity of programs with weight rules. They also presented DP algorithms for programs with cardinality rules (i.e., restricted version of weight rules), but without disjunction.

In this paper, we propose DP algorithms for finding answer sets that are able to directly treat all kinds of ASP rules. While such rules can be transformed into disjunctive rules, we avoid the resulting polynomial overhead with our algorithms. In particular, we present two approaches based on two different types of graphs representing the program structure. Firstly, we consider the primal graph, which allows for an intuitive algorithm that also treats the extended ASP rules. While for a given disjunctive program the treewidth of the primal graph may be larger than treewidth of the graph representation used by Jakl et al. [9], our algorithm uses simpler data structures and lays the foundations to understand how we can handle also extended rules. Our second graph representation is the incidence graph, a generalization of the representation used by Jakl et al. Algorithms for this graph representation are more sophisticated, since weight and choice rules can no longer be completely evaluated in the same computation step. Our algorithms yield upper bounds that are linear in the program size, double-exponential in the treewidth, and single-exponential in the maximum weights. We extend our two algorithms to count optimal answer sets. For this particular task, experiments show that we are able to outperform existing systems from multiple domains, given input instances of low treewidth, both randomly generated and obtained from real-world graphs of traffic networks. Our system is publicly available on github[1].

2 Formal Background

Answer Set programming (ASP). ASP is a declarative modeling and problem solving framework; for a full introduction, see, e.g., [3]. State-of-the-art ASP grounders support the full ASP-Core-2 language [4] and output smodels input format [13], which we will use for our algorithms. Let ℓ, m, n be non-negative integers such that $\ell \leq m \leq n$, a_1, ..., a_n distinct propositional atoms, w, w_1, ..., w_n non-negative integers, and $l \in \{a_1, \neg a_1\}$. A *choice rule* is an expression of the form, $\{a_1; \ldots; a_\ell\} \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$, a *disjunctive rule* is of the form $a_1 \vee \cdots \vee a_\ell \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$ and a *weight rule* is of the form $a_\ell \leftarrow w \leqslant \{a_{\ell+1} = w_{\ell+1}, \ldots, a_m = w_m, \neg a_{m+1} = w_{m+1}, \ldots, \neg a_n = w_n\}$. Finally, an *optimization rule* is an expression of the form $\leftsquigarrow l[w]$. A *rule* is either a disjunctive, a choice, a weight, or an optimization rule. For a choice, disjunctive, or weight rule r, let $H_r := \{a_1, \ldots, a_\ell\}$, $B_r^+ := \{a_{\ell+1}, \ldots, a_m\}$, and $B_r^- := \{a_{m+1}, \ldots, a_n\}$. For a weight rule r, let $\mathrm{wght}(r, a)$ map atom a to its corresponding weight w_i in rule r if $a = a_i$ for $\ell + 1 \leq i \leq n$ and to 0 otherwise,

[1] See https://github.com/daajoe/dynasp/tree/v2.0.0.

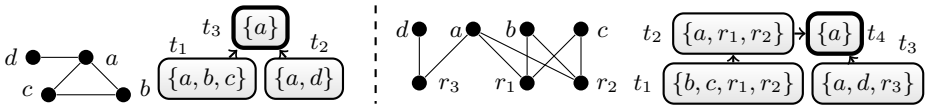

Figure 1 Graph G_1 with a TD of G_1 (left) and graph G_2 with a TD of G_2 (right).

let $\mathrm{wght}(r, A) := \sum_{a \in A} \mathrm{wght}(r, a)$ for a set A of atoms, and let $\mathrm{bnd}(r) := w$ be its *bound*. For an optimization rule r, let $\mathrm{cst}(r) := w$ and if $l = a_1$, let $B_r^+ := \{a_1\}$ and $B_r^- := \emptyset$; or if $l = \neg a_1$, let $B_r^- := \{a_1\}$ and $B_r^+ := \emptyset$. For a rule r, let $\mathrm{at}(r) := H_r \cup B_r^+ \cup B_r^-$ denote its *atoms* and $B_r := B_r^+ \cup \{\neg b \mid b \in B_r^-\}$ its *body*. A *program* Π is a set of rules. Let $\mathrm{at}(\Pi) := \{\mathrm{at}(r) \mid r \in \Pi\}$ and let $\mathrm{CH}(\Pi), \mathrm{DISJ}(\Pi), \mathrm{OPT}(\Pi)$ and $\mathrm{WGT}(\Pi)$ denote the set of all choice, disjunctive, optimization and weight rules in Π, respectively. A set $M \subseteq \mathrm{at}(\Pi)$ *satisfies* a rule r if (i) $(H_r \cup B_r^-) \cap M \neq \emptyset$ or $B_r^+ \not\subseteq M$ for $r \in \mathrm{DISJ}(\Pi)$, (ii) $H_r \cap M \neq \emptyset$ or $\Sigma_{a_i \in M \cap B_r^+} w_i + \Sigma_{a_i \in B_r^- \setminus M} w_i < \mathrm{bnd}(r)$ for $r \in \mathrm{WGT}(\Pi)$, or (iii) $r \in \mathrm{CH}(\Pi) \cup \mathrm{OPT}(\Pi)$. M is a model of Π, denoted by $M \vDash \Pi$, if M satisfies every rule $r \in \Pi$. Further, let $\mathrm{Mod}(\mathcal{C}, \Pi) := \{C \mid C \in \mathcal{C}, C \vDash \Pi\}$ for $\mathcal{C} \subseteq 2^{\mathrm{at}(\Pi)}$. The *reduct* r^M (i) of a choice rule r is the set $\{a \leftarrow B_r^+ \mid a \in H_r \cap M, B_r^- \cap M = \emptyset\}$ of rules, (ii) of a disjunctive rule r is the singleton $\{H_r \leftarrow B_r^+ \mid B_r^- \cap M = \emptyset\}$, and (iii) of a weight rule r is the singleton $\{H_r \leftarrow w' \leqslant [a = \mathrm{wght}(r, a) \mid a \in B_r^+]\}$ where $w' = \mathrm{bnd}(r) - \Sigma_{a \in B_r^- \setminus M} \mathrm{wght}(r, a)$. $\Pi^M := \{r' \mid r' \in r^M, r \in \Pi\}$ is called *GL reduct* of Π with respect to M. A set $M \subseteq \mathrm{at}(\Pi)$ is an *answer set* of program Π if (i) $M \vDash \Pi$ and (ii) there is no $M' \subsetneq M$ such that $M' \vDash \Pi^M$, that is, M is *subset minimal with respect to* Π^M. We call $\mathrm{cst}(\Pi, M, A) := \Sigma_{r \in \mathrm{OPT}(\Pi),\ A \cap [(B_r^+ \cap M) \cup (B_r^- \setminus M)] \neq \emptyset} \mathrm{cst}(r)$ the *cost* of model M for Π with respect to the set $A \subseteq \mathrm{at}(\Pi)$. An answer set M of Π is *optimal* if its cost is minimal over all answer sets.

Example 1. Let $\Pi := \{\overbrace{\{a; b\} \leftarrow c}^{r_1};\ \overbrace{c \leftarrow 1 \leqslant \{b = 1, \neg a = 1\}}^{r_2};\ \overbrace{d \vee a \leftarrow}^{r_3}\}$. Then, the sets $\{a\}$, $\{c, d\}$ and $\{b, c, d\}$ are answer sets of Π. ■

Given a program Π, we consider the problems of computing an answer set (called AS) and outputting the number of optimal answer sets (called #ASPO).

Tree Decompositions. Let $G = (V, E)$ be a graph, $T = (N, F, n)$ a rooted tree, and $\chi : N \to 2^V$ a function that maps each node $t \in N$ to a set of vertices. We call the sets $\chi(\cdot)$ *bags* and N the set of nodes. Then, the pair $\mathcal{T} = (T, \chi)$ is a *tree decomposition (TD)* of G if the following conditions hold: (i) all vertices occur in some bag, that is, for every vertex $v \in V$ there is a node $t \in N$ with $v \in \chi(t)$; (ii) all edges occur in some bag, that is, for every edge $e \in E$ there is a node $t \in N$ with $e \subseteq \chi(t)$; and (iii) the *connectedness condition*: for any three nodes $t_1, t_2, t_3 \in N$, if t_2 lies on the unique path from t_1 to t_3, then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$. We call $\max\{|\chi(t)| - 1 \mid t \in N\}$ the *width* of the TD. The *treewidth* $tw(G)$ of a graph G is the minimum width over all possible TDs of G. For some arbitrary but fixed integer k and a graph of treewidth at most k, we can compute a TD of width $\leqslant k$ in time $2^{\mathcal{O}(k^3)} \cdot |V|$ [2]. Given a TD (T, χ) with $T = (N, \cdot, \cdot)$, for a node $t \in N$ we say that $\mathrm{type}(t)$ is *leaf* if t has no children; *join* if t has children t' and t'' with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; *int* ("introduce")

Algorithm 1: Algorithm $\mathcal{DP}_{\mathcal{A}}(\mathcal{T})$ for Dynamic Programming on TD $\mathcal{T}$ for ASP.

In: Table algorithm $\mathcal{A}$, nice TD $\mathcal{T} = (T, \chi)$ with $T = (N, \cdot, n)$ of $G(\Pi)$ according to $\mathcal{A}$.
Out: Table: maps each TD node $t \in T$ to some computed table τ_t.

```
for iterate t in post-order(T,n) do
    Child-Tabs := {Tables[t'] | t' is a child of t in T}
    Tables[t] := A(t, χ(t), Π_t, at_≤t, Child-Tabs)
```

if t has a single child t', $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; *rem* ("removal") if t has a single child t', $\chi(t) \subseteq \chi(t')$ and $|\chi(t')| = |\chi(t)| + 1$. If every node $t \in N$ has at most two children, $\text{type}(t) \in \{\mathit{leaf}, \mathit{join}, \mathit{int}, \mathit{rem}\}$, and bags of leaf nodes and the root are empty, then the TD is called *nice*. For every TD, we can compute a nice TD in linear time without increasing the width [2]. In our algorithms, we will traverse a TD bottom up, therefore, let post-order(T, t) be the sequence of nodes in post-order of the induced subtree $T' = (N', \cdot, t)$ of T rooted at t.

Example 2. Figure 1 (left) shows a graph G_1 together with a TD of G_1 that is of width 2. Note that G_1 has treewidth 2, since it contains a clique on the vertices $\{a, b, c\}$. Further, the TD $\mathcal{T}$ in Figure 2 is a nice TD of G_1. ■

Graph Representations of Programs. In order to use TDs for ASP solving, we need dedicated graph representations of ASP programs. The *primal graph* $P(\Pi)$ of program Π has the atoms of Π as vertices and an edge $a\,b$ if there exists a rule $r \in \Pi$ and $a, b \in \text{at}(r)$. The *incidence graph* $I(\Pi)$ of Π is the bipartite graph that has the atoms and rules of Π as vertices and an edge $a\,r$ if $a \in \text{at}(r)$ for some rule $r \in \Pi$. These definitions adapt similar concepts from SAT [11].

Example 3. Recall program Π of Example 1. We observe that graph G_1 (G_2) in the left (right) part of Figure 1 is the primal (incidence) graph of Π. ■

Sub-Programs. Let $\mathcal{T} = (T, \chi)$ be a nice TD of graph representation $H \in \{I(\Pi), P(\Pi)\}$ of a program Π. Further, let $T = (N, \cdot, n)$ and $t \in N$. The *bag-rules* are defined as $\Pi_t := \{r \mid r \in \Pi, \text{at}(r) \subseteq \chi(t)\}$ if H is the primal graph and as $\Pi_t := \Pi \cap \chi(t)$ if H is the incidence graph. Further, the set $\text{at}_{\leq t} := \{a \mid a \in \text{at}(\Pi) \cap \chi(t'), t' \in \text{post-order}(T, t)\}$ is called *atoms below* t, the *program below* t is defined as $\Pi_{\leq t} := \{r \mid r \in \Pi_{t'}, t' \in \text{post-order}(T, t)\}$, and the *program strictly below* t is $\Pi_{<t} := \Pi_{\leq t} \setminus \Pi_t$. It holds that $\Pi_{\leq n} = \Pi_{<n} = \Pi$ and $\text{at}_{\leq n} = \text{at}(\Pi)$.

Example 4. Intuitively, TDs of Figure 1 enable us to evaluate Π by analyzing sub-programs ($\{r_1, r_2\}$ and $\{r_3\}$) and combining results agreeing on a. Indeed, for the given TD of Figure 1 (left), $\Pi_{\leq t_1} = \{r_1, r_2\}$, $\Pi_{\leq t_2} = \{r_3\}$ and $\Pi = \Pi_{\leq t_3} = \Pi_{<t_3} = \Pi_{t_1} \cup \Pi_{t_2}$. For the TD of Figure 1 (right), we have $\Pi_{\leq t_1} = \{r_1, r_2\}$ and $\text{at}_{\leq t_1} = \{b, c\}$, as well as $\Pi_{\leq t_3} = \{r_3\}$ and $\text{at}_{\leq t_3} = \{a, d\}$. Moreover, for TD $\mathcal{T}$ of Figure 2, $\Pi_{\leq t_1} = \Pi_{\leq t_2} = \Pi_{\leq t_3} = \Pi_{<t_4} = \emptyset$, $\text{at}_{\leq t_3} = \{a, b\}$ and $\Pi_{\leq t_4} = \{r_1, r_2\}$. ■

3 ASP via Dynamic Programming on TDs

In the next two sections, we propose two dynamic programming (DP) algorithms, $\mathcal{DP}_{\mathsf{PRIM}}$ and $\mathcal{DP}_{\mathsf{INC}}$, for ASP without optimization rules based on two different

Algorithm 2: Table algorithm $\mathsf{PRIM}(t, \chi_t, \Pi_t, \cdot, \text{Child-Tabs})$.

In: Bag χ_t, bag-rules Π_t and child tables Child-Tabs of node t. **Out:** Table τ_t.

1 **if** type$(t) =$ *leaf* **then** $\tau_t := \{\langle \emptyset,\ \emptyset \rangle\}$ `/* Abbreviations see Footnote 2. */`

2 **else if** type$(t) =$ *int*, $a \in \chi_t$ *is introduced and* $\tau' \in$ *Child-Tabs* **then**

3 $\quad \tau_t := \{\langle M_a^+,\ \mathrm{Mod}(\{M\} \cup [\mathcal{C} \sqcup \{a\}] \cup \mathcal{C}, \Pi_t^{M_a^+}) \rangle \quad | \ \langle M, \mathcal{C} \rangle \in \tau', M_a^+ \vDash \Pi_t\} \bigcup$

4 $\qquad \{\langle M,\ \mathrm{Mod}(\mathcal{C}, \Pi_t^M) \rangle \quad | \ \langle M, \mathcal{C} \rangle \in \tau', M \vDash \Pi_t\}$

5 **else if** type$(t) =$ *rem*, $a \notin \chi_t$ *is removed and* $\tau' \in$ *Child-Tabs* **then**

6 $\quad \tau_t := \{\langle M_a^-,\ \{C_a^- \mid C \in \mathcal{C}\} \rangle \quad | \ \langle M, \mathcal{C} \rangle \in \tau'\}$

7 **else if** type$(t) =$ *join and* $\tau', \tau'' \in$ *Child-Tabs with* $\tau' \neq \tau''$ **then**

8 $\quad \tau_t := \{\langle M,\ (\mathcal{C}' \cap \mathcal{C}'') \cup (\mathcal{C}' \cap \{M\}) \cup (\{M\} \cap \mathcal{C}'') \rangle \quad | \ \langle M, \mathcal{C}' \rangle \in \tau', \langle M, \mathcal{C}'' \rangle \in \tau''\}$

graph representations, namely the primal and the incidence graph. Both algorithms make use of the fact that answer sets of a given program Π are (i) models of Π and (ii) subset minimal with respect to Π^M. Intuitively, our algorithms compute, for each TD node t, (i) sets of atoms—(local) *witnesses*—representing parts of potential models of Π, and (ii) for each local witness M subsets of M—(local) *counterwitnesses*—representing subsets of potential models of Π^M which (locally) contradict that M can be extended to an answer set of Π. We give the the basis of our algorithms in Algorithm 1 ($\mathcal{DP}_\mathcal{A}$), which sketches the general DP scheme for ASP solving on TDs. Roughly, the algorithm splits the search space based on a given nice TD and evaluates the input program Π in parts. The results are stored in so-called tables, that is, sets of all possible tuples of witnesses and counterwitnesses for a given TD node. To this end, we define the *table algorithms* PRIM and INC, which compute tables for a node t of the TD using the primal graph $P(\Pi)$ and incidence graph $I(\Pi)$, respectively. To be more concrete, given a table algorithm $\mathcal{A} \in \{\mathsf{PRIM}, \mathsf{INC}\}$, algorithm $\mathcal{DP}_\mathcal{A}$ visits every node $t \in T$ in post-order; then, based on Π_t, computes a table τ_t for node t from the tables of the children of t, and stores τ_t in Tables[t].

3.1 Using Decompositions of Primal Graphs

In this section, we present our algorithm PRIM in two parts: (i) finding models of Π and (ii) finding models which are subset minimal with respect to Π^M. For sake of clarity, we first present only the first tuple positions (red parts) of Algorithm 2 (PRIM) to solve (i). We call the resulting table algorithm MOD.

Example 5. Consider program Π from Example 1 and in Figure 2 (left) TD $\mathcal{T} = (\cdot, \chi)$ of $P(\Pi)$ and the tables $\tau_1, \ldots, \tau_{12}$, which illustrate computation results obtained during post-order traversal of $\mathcal{T}$ by $\mathcal{DP}_\mathsf{MOD}$. Table $\tau_1 = \{\langle\emptyset\rangle\}$ as type$(t_1) =$ *leaf*. Since type$(t_2) =$ *int*, we construct table τ_2 from τ_1 by taking $M_{1.i}$ and $M_{1.i} \cup \{a\}$ for each $M_{1.i} \in \tau_1$ (corresponding to a guess on a). Then, t_3 introduces b and t_4 introduces c. $\Pi_{t_1} = \Pi_{t_2} = \Pi_{t_3} = \emptyset$, but since $\chi(t_4) \subseteq \mathrm{at}(r_1) \cup \mathrm{at}(r_2)$ we have $\Pi_{t_4} = \{r_1, r_2\}$ for t_4. In consequence, for each $M_{4.i}$ of table τ_4, we have $M_{4.i} \vDash \{r_1, r_2\}$ since MOD enforces satisfiability of Π_t in node t.

[2] $\mathcal{S} \sqcup \{e\} := \{S \cup \{e\} \mid S \in \mathcal{S}\}$, $S_e^+ := S \cup \{e\}$, and $S_e^- := S \setminus \{e\}$

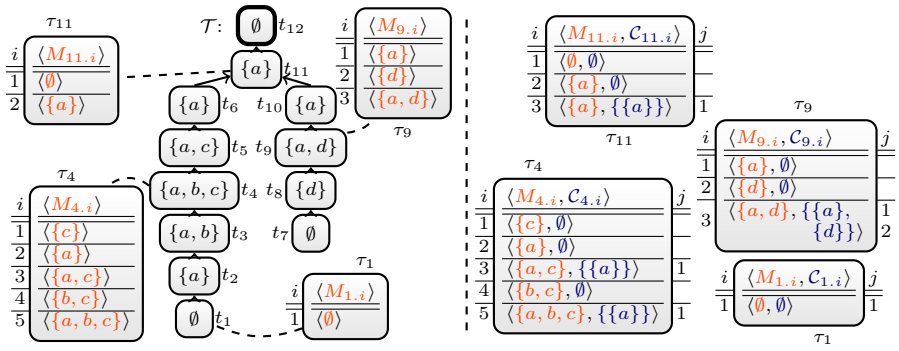

Figure 2 Selected DP tables of MOD (left) and PRIM (right) for nice TD $\mathcal{T}$.

We derive tables τ_7 to τ_9 similarly. Since $\text{type}(t_5) = \textit{rem}$, we remove atom b from all elements in τ_4 to construct τ_5. Note that we have already seen all rules where b occurs and hence b can no longer affect witnesses during the remaining traversal. We similarly construct $\tau_{t_6} = \tau_{10} = \{\langle\emptyset\rangle, \langle a\rangle\}$. Since $\text{type}(t_{11}) = \textit{join}$, we construct table τ_{11} by taking the intersection $\tau_6 \cap \tau_{10}$. Intuitively, this combines witnesses agreeing on a. Node t_{12} is again of type *rem*. By definition (primal graph and TDs) for every $r \in \Pi$, atoms $at(r)$ occur together in at least one common bag. Hence, $\Pi = \Pi_{\leq t_{12}}$ and since $\tau_{12} = \{\langle\emptyset\rangle\}$, we can construct a model of Π from the tables. For example, we obtain the model $\{a, d\} = M_{11.2} \cup M_{4.2} \cup M_{9.3}$. ■

PRIM is given in Algorithm 2. Tuples in τ_t are of the form $\langle M, \mathcal{C}\rangle$. Witness $M \subseteq \chi(t)$ represents a model of Π_t witnessing the existence of $M' \supseteq M$ with $M' \vDash \Pi_{\leq t}$. The family $\mathcal{C} \subseteq 2^M$ contains sets of models $C \subseteq M$ of the GL reduct $(\Pi_t)^M$. C witnesses the existence of a set C' with counterwitness $C \subseteq C' \subsetneq M'$ and $C' \vDash (\Pi_{\leq t})^{M'}$. There is an answer set of Π if table t_n for root n contains $\langle\emptyset, \emptyset\rangle$. Since in Example 5 we already explained the first tuple position and thus the witness part, we only briefly describe the parts for counterwitnesses. In the introduce case, we want to store only counterwitnesses for not being minimal with respect to the GL reduct of the bag-rules. Therefore, in Line 3 we construct for M_a^+ counterwitnesses from either some witness M ($M \subsetneq M_a^+$), or of any $C \in \mathcal{C}$, or of any $C \in \mathcal{C}$ extended by a (every $C \in \mathcal{C}$ was already a counterwitness before). Line 4 ensures that only counterwitnesses that are models of the GL reduct Π_t^M are stored (via $\text{Mod}(\cdot, \cdot)$). Line 6 restricts counterwitnesses to its bag content, and Line 8 enforces that child tuples agree on counterwitnesses.

Example 6. Consider Example 5, its TD $\mathcal{T} = (\cdot, \chi)$, Figure 2 (right), and the tables $\tau_1, \ldots, \tau_{12}$ obtained by $\mathcal{DP}_{\text{PRIM}}$. Since we have $\text{at}(r_1) \cup \text{at}(r_2) \subseteq \chi(t_4)$, we require $C_{4.i.j} \vDash \{r_1, r_2\}^{M_{4.i}}$ for each counterwitness $C_{4.i.j} \in \mathcal{C}_{4.i}$ in tuples of τ_4. For $M_{4.5} = \{a, b, c\}$ observe that the only counterwitness of $\{r_1, r_2\}^{M_{4.5}} = \{a \leftarrow c, b \leftarrow c, c \leftarrow 1 \leq \{b = 1\}\}$ is $C_{4.5.1} = \{a\}$. Note that witness $M_{11.2}$ of table τ_{11} is the result of joining $M_{4.2}$ with $M_{9.1}$ and witness $M_{11.3}$ (counterwitness $C_{11.3.1}$) is the result of joining $M_{4.3}$ with $M_{9.3}$ ($C_{4.3.1}$ with $C_{9.3.1}$), and $M_{4.5}$ with $M_{9.3}$ ($C_{4.5.1}$ with $C_{9.3.2}$). $C_{11.3.1}$ witnesses that neither $M_{4.3} \cup M_{9.3}$ nor $M_{4.5} \cup M_{9.3}$ forms an answer set of Π. Since τ_{12} contains $\langle\emptyset, \emptyset\rangle$ there is no counterwitness

Algorithm 3: Table algorithm $\mathsf{INC}(t, \chi_t, \Pi_t, \mathrm{at}_{\leq t}, \text{Child-Tabs})$.

In: Bag χ_t, bag-rules Π_t, atoms-below $\mathrm{at}_{\leq t}$, child tables Child-Tabs of t. **Out:** Tab. τ_t.

if $\mathrm{type}(t) = \mathit{leaf}$ **then** $\tau_t := \{\langle \emptyset, \emptyset, \emptyset \rangle\}$ `/* Abbreviations see Footnote 3. */`

else if $\mathrm{type}(t) = \mathit{int}$, $a \in \chi_t \setminus \Pi_t$ *is introduced and* $\tau' \in$ *Child-Tabs* **then**

$\tau_t := \{\langle M_a^+, \sigma \uplus \mathrm{SatRules}(\dot{\Pi}_t^{(t,\sigma)}, M_a^+),\ \{\langle M, \sigma \uplus \mathrm{SatRules}(\dot{\Pi}_t^{(t,\sigma,M_a^+)}, M)\rangle\} \cup$
$\{\langle C_a^+, \rho \uplus \mathrm{SatRules}(\dot{\Pi}_t^{(t,\rho,M_a^+)}, C_a^+)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\} \cup$
$\{\langle C, \rho \uplus \mathrm{SatRules}(\dot{\Pi}_t^{(t,\rho,M_a^+)}, C)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\} \bigcup$
$\{\langle M, \sigma \uplus \mathrm{SatRules}(\dot{\Pi}_t^{(t,\sigma)}, M),$
$\{\langle C, \rho \uplus \mathrm{SatRules}(\dot{\Pi}_t^{(t,\rho,M)}, C)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

else if $\mathrm{type}(t) = \mathit{int}$, $r \in \chi_t \cap \Pi_t$ *is introduced and* $\tau' \in$ *Child-Tabs* **then**

$\tau_t := \{\langle M, \sigma_r^+ \uplus \mathrm{SatRules}(\{\dot{r}\}^{(t,\sigma_r^+\})}, M),$
$\{\langle C, \rho_r^+ \uplus \mathrm{SatRules}(\{\dot{r}\}^{(t,\rho_r^+,M)}, C)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

else if $\mathrm{type}(t) = \mathit{rem}$, $a \notin \chi_t$ *is removed atom and* $\tau' \in$ *Child-Tabs* **then**

$\tau_t := \{\langle M_a^-, \sigma \uplus \mathrm{UpdtWgt}(\Pi_t, M, a),$
$\{\langle C_a^-, \rho \uplus \mathrm{UpdtWgt\&Ch}(\Pi_t, M, C, a)\rangle \mid \langle C, \rho\rangle \in \mathcal{C}\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau'\}$

else if $\mathrm{type}(t) = \mathit{rem}$, $r \notin \chi_t$ *is removed rule and* $\tau' \in$ *Child-Tabs* **then**

$\tau_t := \{\langle M, \sigma_{\{r\}}^-, \{\langle C, \rho_{\{r\}}^-\rangle \mid \langle C, \rho\rangle \in \mathcal{C}, \rho(r) = \infty\}\rangle \mid \langle M, \sigma, \mathcal{C}\rangle \in \tau', \sigma(r) = \infty\}$

else if $\mathrm{type}(t) = \mathit{join}$ *and* $\tau', \tau'' \in$ *Child-Tabs with* $\tau' \neq \tau''$ **then**

$\tau_t := \{\langle M, \sigma' \uplus \sigma'',\ \{\langle C, \rho' \uplus \rho''\rangle \mid \langle C, \rho'\rangle \in \mathcal{C}', \langle C, \rho''\rangle \in \mathcal{C}''\} \cup$
$\{\langle M, \rho \uplus \sigma''\rangle \mid \langle M, \rho\rangle \in \mathcal{C}'\} \cup$
$\{\langle M, \sigma' \uplus \rho\rangle \mid \langle M, \rho\rangle \in \mathcal{C}''\}\rangle \mid \langle M, \sigma', \mathcal{C}'\rangle \in \tau', \langle M, \sigma'', \mathcal{C}''\rangle \in \tau''\}$

for $M_{11.2}$, we can construct an answer set of Π from the tables, e.g., $\{a\}$ can be constructed from $M_{4.2} \cup M_{9.1}$. ■

Theorem 1. *Given a program Π, the algorithm $\mathcal{DP}_{\mathsf{PRIM}}$ is correct and runs in time $\mathcal{O}(2^{2^{k+2}} \cdot \|P(\Pi)\|)$ where k is the treewidth of the primal graph $P(\Pi)$.*

3.2 Using Decompositions of Incidence Graphs

Our next algorithm ($\mathcal{DP}_{\mathsf{INC}}$) takes the incidence graph as graph representation of the input program. The treewidth of the incidence graph is smaller than the treewidth of the primal graph plus one, cf., [11,7]. More importantly, the primal graph contains a clique on $\mathrm{at}(r)$ for each rule r. The incidence graph, compared to the primal graph, contains rules as vertices and its relationship to the atoms in terms of edges. By definition, we have no guarantee that all atoms of a rule occur together in the same bag of TDs of the incidence graph. For that reason, we *cannot* locally check the satisfiability of a rule when traversing the TD without additional stored information (so-called *rule-states* that intuitively represent how much of a rule is already (dis-)satisfied). We only know that for each rule r there is a path $p = t_{int}, t_1, \ldots, t_m, t_{rem}$ where t_{int} introduces r and t_{rem} removes r and when considering t_{rem} in the table algorithm we have seen all atoms that occur in rule r. Thus, on removal of r in t_{rem} we ensure that r is satisfied while taking

rule-states for choice and weight rules into account. Consequently, our tuples will contain a witness, its rule-state, and counterwitnesses and their rule-states.

A tuple in τ_t for Algorithm 3 (INC) is a triple $\langle M, \sigma, \mathcal{C}\rangle$. The set $M \subseteq \mathrm{at}(\Pi) \cap \chi(t)$ represents again a witness. A *rule-state* σ is a mapping $\sigma : \Pi_t \to \mathbb{N}_0 \cup \{\infty\}$. A rule state for M represents whether rules of $\chi(t)$ are either (i) satisfied by a superset of M or (ii) undecided for M. Formally, the set $SR(\Pi_t, \sigma)$ of satisfied bag-rules Π_t consists of each rule $r \in \Pi_t$ such that $\sigma(r) = \infty$. Hence, M witnesses a model $M' \supseteq M$ where $M' \vDash \Pi_{<t} \cup SR(\Pi_t, \sigma)$. $\mathcal{C}$ concerns counterwitnesses.

We compute a new rule-state σ from a rule-state, "updated" bounds for weight rules (UpdtWgt), and satisfied rules (SatRules, defined below). We define $\mathrm{UpdtWgt}(\Pi_t, M, a) := \sigma'$ depending on an atom a with $\sigma'(r) := \mathrm{wght}(r, \{a\} \cap [(B_r^- \setminus M) \cup (B_r^+ \cap M)])$, if $r \in \mathrm{WGT}(\Pi_t)$. We use binary operator $\uplus$[3] to combine rule-states, which ensures that rules satisfied in at least one operand remain satisfied. Next, we explain the meaning of rule-states.

Example 7. Consider program Π from Example 1 and TD $\mathcal{T}' = (\cdot, \chi)$ of $I(\Pi)$ and the tables $\tau_1, \ldots, \tau_{18}$ in Figure 3 (left). We are only interested in the first two tuple positions (red and green parts) and implicitly assume that "i" refers to Line i in the respective table. Consider $M_{4.1} = \{c\}$ in table τ_4. Since $H_{r_2} = \{c\}$, witness $M_{4.1} = \{c\}$ satisfies rule r_2. As a result, $\sigma_{4.1}(r_2) = \infty$ remembering satisfied rule r_2 for $M_{4.1}$. Since $c \notin M_{4.2}$ and $B^+_{r_1} = \{c\}$, $M_{4.2}$ satisfies rule r_1, resulting in $\sigma_{4.2}(r_1) = \infty$. Rule-state $\sigma_{4.1}(r_1)$ represents that r_1 is undecided for $M_{4.2}$. For weight rule r_2, rule-states remember the sum of body weights involving removed atoms. Consider $M_{6.2} = M_{6.3} = \emptyset$ of table τ_6. We have $\sigma_{6.2}(r_2) \neq \sigma_{6.3}(r_2)$, because $M_{6.2}$ was obtained from some $M_{5.i}$ of table τ_5 with $b \notin M_{5.i}$ and b occurs in $B^+_{r_2}$ with weight 1, resulting in $\sigma_{6.3}(r_2) = 1$; whereas $M_{6.3}$ extends some $M_{5.j}$ with $b \notin M_{5.j}$. ■

In order to decide in node t whether a witness satisfies rule $r \in \Pi_t$, we check satisfiability of program $\dot{\mathcal{R}}(r)$ constructed by $\dot{\mathcal{R}}$, which maps rules to state-programs. Formally, for $M \subseteq \chi(t) \setminus \Pi_t$, $\mathrm{SatRules}(\dot{\mathcal{R}}, M) := \sigma$ where $\sigma(r) := \infty$ if $(r, \mathcal{R}) \in \dot{\mathcal{R}}$ and $M \vDash \mathcal{R}$.

Definition 1. *Let Π be a program, $\mathcal{T} = (\cdot, \chi)$ be a TD of $I(\Pi)$, t be a node of $\mathcal{T}$, $\mathcal{P} \subseteq \Pi_t$, and $\sigma : \Pi_t \to \mathbb{N}_0 \cup \{\infty\}$ be a rule-state. The* state-program $\mathcal{P}^{(t,\sigma)}$ *is obtained from $\mathcal{P} \cup \{\leftarrow B_r \mid r \in \mathrm{CH}(\mathcal{P}), H_r \subsetneq \mathrm{at}_{\leq t}\}$[4] by*

1. *removing rules r with $\sigma(r) = \infty$ ("already satisfied rules");*
2. *removing from every rule all literals $a, \neg a$ with $a \notin \chi(t)$; and*
3. *setting new bound $\max\{0, \mathrm{bnd}(r) - \sigma(r) - \mathrm{wght}(r, \mathrm{at}(r) \setminus \mathrm{at}_{\leq t})\}$ for weight rule r.*

We define $\dot{\mathcal{P}}^{(t,\sigma)} : \mathcal{P} \to 2^{\mathcal{P}^{(t,\sigma)}}$ by $\dot{\mathcal{P}}^{(t,\sigma)}(r) := \{r\}^{(t,\sigma)}$ for $r \in \mathcal{P}$.

Example 8. Observe $\Pi_{t_1}^{(t_1,\emptyset)} = \{\{b\} \leftarrow c, \leftarrow c, c \leftarrow 0 \leq \{b = 1\}\}$ and $\Pi_{t_2}^{(t_2,\emptyset)} = \{\{a\} \leftarrow, \leftarrow 1 \leq \{\neg a = 1\}\}$ for Π_{t_1}, Π_{t_2} of Figure 1(right). ■

[3] $\sigma \uplus \rho := \{(x, \sum_{(x,c_1)\in\sigma} c_1 + \sum_{(x,c_2)\in\rho} c_2) \mid (x, \cdot) \in \sigma \cup \rho\}$; $\sigma_r^+ := \sigma \cup \{(r, 0)\}$; $\sigma_S^- := \{(x, y) \in \sigma \mid x \notin S\}$.

[4] We require to add $\{\leftarrow B_r \mid r \in \mathrm{CH}(\mathcal{P}), H_r \subsetneq \mathrm{at}_{\leq t}\}$ in order to decide satisfiability for corner cases of choice rules involving counterwitnesses of Line 3 in Algorithm 3.

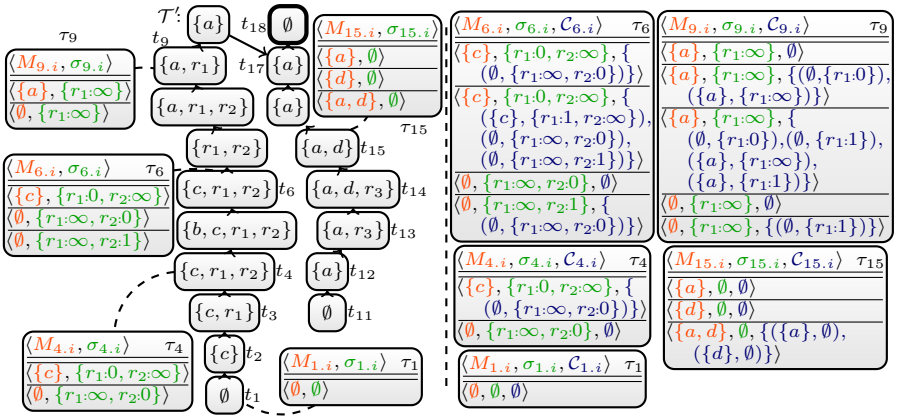

Figure 3 Selected DP tables of IMOD (left) and INC (right) for nice TD $\mathcal{T}'$.

The following example provides an idea how we compute models of a given program using the incidence graph. The resulting algorithm IMOD is the same as INC, except that only the first two tuple positions (red and green parts) are considered.

Example 9. Again, we consider Π of Example 1 and in Figure 3 (left) $\mathcal{T}'$ as well as tables $\tau_1, \ldots, \tau_{18}$. Table $\tau_1 = \{\langle\emptyset, \emptyset\rangle\}$ as type$(t_1) = \mathit{leaf}$. Since type$(t_2) = \mathit{int}$ and t_2 introduces atom c, we construct τ_2 from τ_1 by taking $M_{2.1} := M_{1.1} \cup \{c\}$ and $M_{2.2} := M_{1.1}$ as well as rule-state $\emptyset$. Because type$(t_3) = \mathit{int}$ and t_3 introduces rule r_1, we consider state program $L_3 := \{r_1\}^{(t_3,\{(r_1,0)\})} = \{\leftarrow c\}$ for SatRules$(\dot{L}_3, M_{2.1}) = \{(r_1, 0)\}$ as well as SatRules$(\dot{L}_3, M_{2.2}) = \{(r_1, \infty)\}$ (according to Line 9 of Algorithm 3). Because type$(t_4) = \mathit{int}$ and t_4 introduces rule r_2, we consider $M_{3.1} := M_{2.1}$ and $M_{3.2} := M_{2.2}$ and state program $L_4 := \{r_2\}^{(t_4,\{(r_2,0)\})} = \{c \leftarrow\ 0 \leqslant \{\}\} = \{c \leftarrow\}$ for SatRules$(\dot{L}_4, M_{3.1}) = \{(r_2, \infty)\}$ as well as SatRules$(\dot{L}_4, M_{3.2}) = \{(r_2, 0)\}$ (see Line 9). Node t_5 introduces b (table not shown) and node t_6 removes b. Table τ_6 was discussed in Example 7. When we remove b in t_6 we have decided the "influence" of b on the satisfiability of r_1 and r_2 and thus all rules where b occurs. Tables τ_7 and τ_8 can be derived similarly. Then, t_9 removes rule r_2 and we ensure that every witness $M_{9.1}$ can be extended to a model of r_2, i.e., witness candidates for τ_9 are $M_{8.i}$ with $\sigma_{8.i}(r_2) = \infty$. The remaining tables are derived similarly. For example, table τ_{17} for join node t_{17} is derived analogously to table τ_{17} for algorithm PRIM in Figure 2, but, in addition, also combines the rule-states as specified in Algorithm 3. ■

Since we already explained how to obtain models, we only briefly describe how we handle the counterwitness part. Family $\mathcal{C}$ consists of tuples (C, ρ) where $C \subseteq \mathrm{at}(\Pi) \cap \chi(t)$ is a *counterwitness* in t to M. Similar to the rule-state σ the rule-state ρ for C under M represents whether rules of the GL reduct Π_t^M are either (i) satisfied by a superset of C or (ii) undecided for C. Thus, C witnesses the existence of $C' \subsetneq M'$ satisfying $C' \vDash (\Pi_{<t} \cup SR(\Pi_t, \rho))^{M'}$ since M witnesses a model $M' \supseteq M$ where $M' \vDash \Pi_{<t} \cup SR(\Pi_t, \rho)$. In consequence, there exists an

Algorithm 4: Algorithm #OINC$(t, \chi_t, \Pi_t, \mathrm{at}_{\leq t}$, Child-Tabs).

In: Bag χ_t, bag-rules Π_t, atoms-below $\mathrm{at}_{\leq t}$, child tables Child-Tabs of t. **Out:** Tab. τ_t.
`/* For` $\langle M, \sigma, \mathcal{C}, c, n\rangle$`, we only state affected parts (cost` c `and count` n`);`
`  ‘‘...’’ indicates computation as before.` $\wr \ldots \int$ `denotes a multiset. */`

1 **if** type$(t) =$ *leaf* **then** $\tau_t := \{\langle \emptyset, \ldots, 0, 1\rangle\}$
2 **else if** type$(t) =$ *int*, $a \in \chi_t \setminus \Pi_t$ *is introduced and* $\tau' \in$ *Child-Tabs* **then**
3 $\quad \tau_t := \{\langle M, \ldots, \mathrm{cst}(\Pi, \emptyset, \{a\}) + c, n\rangle \mid \langle M, \sigma, \mathcal{C}, c, n\rangle \in \tau'\} \bigcup$
4 $\qquad \{\langle M_a^+, \ldots, \mathrm{cst}(\Pi, \{a\}, \{a\}) + c, n\rangle \mid \langle M, \sigma, \mathcal{C}, c, n\rangle \in \tau'\}$
5 **else if** type$(t) =$ *int or rem, removed or introduced* $r \in \Pi_t$, $\tau' \in$ *Child-Tabs* **then**
6 $\quad \tau_t := \{\langle M, \ldots, c, n\rangle \mid \langle M, \sigma, \mathcal{C}, c, n\rangle \in \tau', \ldots\}$
7 **else if** type$(t) =$ *rem*, $a \notin \chi_t$ *is removed atom and* $\tau' \in$ *Child-Tabs* **then**
8 $\quad \tau_t := \mathrm{cnt}(\mathrm{kmin}(\wr \langle M_a^-, \ldots, c, n\rangle \mid \langle M, \sigma, \mathcal{C}, c, n\rangle \in \tau' \int))$
9 **else if** type$(t) =$ *join and* $\tau', \tau'' \in$ *Child-Tabs with* $\tau' \neq \tau''$ **then**
10 $\quad \tau_t := \mathrm{cnt}(\mathrm{kmin}(\wr \langle M, \ldots, c' + c'' - \mathrm{cst}(\Pi, M, \chi_t), n' \cdot n''\rangle$
11 $\qquad \mid \langle M, \sigma', \mathcal{C}', c', n'\rangle \in \tau', \langle M, \sigma'', \mathcal{C}'', c'', n''\rangle \in \tau'' \int))$

answer set of Π if the root table contains $\langle \emptyset, \emptyset, \emptyset\rangle$. In order to locally decide rule satisfiability for counterwitnesses, we require state-programs under witnesses.

Definition 2. *Let Π be a program, $\mathcal{T} = (\cdot, \chi)$ be a TD of $I(\Pi)$, t be a node of $\mathcal{T}$, $\mathcal{P} \subseteq \Pi_t$, $\rho : \Pi_t \to \mathbb{N}_0 \cup \{\infty\}$ be a rule-state and $M \subseteq \mathrm{at}(\Pi)$. We define* state-program $\mathcal{P}^{(t,\rho,M)}$ *by* $[\mathcal{S}^{(t,\rho)}]^M$ *where* $\mathcal{S} := \mathcal{P} \cup \{\leftarrow B_r \mid r \in \mathrm{CH}(\mathcal{P}), \rho(r) > 0\}$, *and* $\dot{\mathcal{P}}^{(t,\rho,M)} : \mathcal{P} \to 2^{\mathcal{P}^{(t,\rho,M)}}$ *by* $\dot{\mathcal{P}}^{(t,\rho,M)}(r) := \{r\}^{(t,\rho,M)}$ *for* $r \in \mathcal{P}$.

We compute a new rule-state ρ for a counterwitness from an earlier rule-state, satisfied rules (SatRules), and both (a) “updated” bounds for weight rules or (b) “updated” value representing whether the head can still be satisfied ($\rho(r) \leq 0$) for choice rules r (UpdtWgt&Ch). Formally, UpdtWgt&Ch$(\Pi_t, M, C, a) := \sigma'$ depending on an atom a with (a) $\sigma'(r) := \mathrm{wght}(r, \{a\} \cap [(B_r^- \setminus M) \cup (B_r^+ \cap C)])$, if $r \in \mathrm{WGT}(\Pi_t)$; and (b) $|\{a\} \cap H_r \cap (M \setminus C)|$, if $r \in \mathrm{CH}(\Pi_t)$.

Theorem 2. *The algorithm $\mathcal{DP}_{\mathsf{INC}}$ is correct.*

Proof. (Idea) A tuple at a node t guarantees that there exists a model for the sub-program induced by the subtree rooted at t, which works for all node types. While traversing the tree decomposition, every answer set is indeed considered.

Theorem 3. *Given a program Π, algorithm $\mathcal{DP}_{\mathsf{INC}}$ runs in time $\mathcal{O}(2^{2^{k+2} \cdot \ell^{k+1}} \cdot \|I(\Pi)\|)$, where $k := tw(I(\Pi))$, and $\ell := max\{3, \mathrm{bnd}(r) \mid r \in \mathrm{WGT}(\Pi)\}$.*

The runtime bounds stated in Theorem 3 appear to be worse than in Theorem 1. However, $tw(I(\Pi)) \leq tw(P(\Pi)) + 1$ and $tw(P(\Pi)) \geq \max\{|\mathrm{at}(r)| \mid r \in \Pi\}$ for a given program Π. Further, there are programs where $tw(I(\Pi)) = 1$, but $tw(P(\Pi)) = k$, e.g., a program consisting of a single rule r with $|\mathrm{at}(r)| = k$. Consequently, worst-case runtime bounds of $\mathcal{DP}_{\mathsf{PRIM}}$ are at least double-exponential in the rule size and $\mathcal{DP}_{\mathsf{PRIM}}$ will perform worse than $\mathcal{DP}_{\mathsf{INC}}$ on input programs containing large rules. However, due to the rule-states, data structures

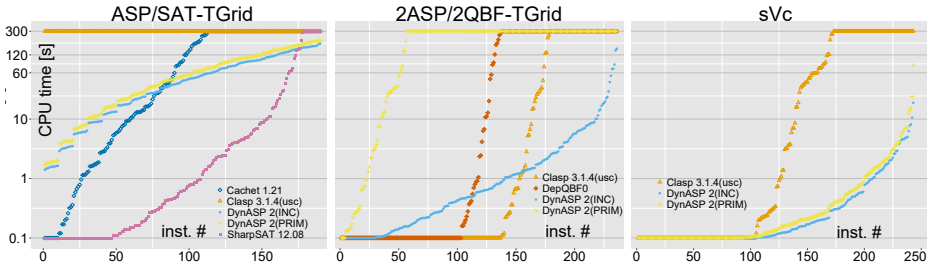

Figure 4 Results of randomly generated and selected real-world instances.

of $\mathcal{DP}_{\mathsf{INC}}$ are much more complex than of $\mathcal{DP}_{\mathsf{PRIM}}$. In consequence, we expect $\mathcal{DP}_{\mathsf{PRIM}}$ to perform better in practice if rules are small and incidence and primal treewidth are therefore almost equal. In summary, we have a trade-off between (i) a more general parameter decreasing the theoretical worst-case runtime and (ii) less complex data structures decreasing the practical overhead to solve AS.

3.3 Extensions for Optimization and Counting

In order to find an answer set of a program with optimization statements or the number of optimal answer sets (#ASPO), we extend our algorithms PRIM and INC. Therefore, we augment tuples stored in tables with an integers c and n describing the cost and the number of witnessed sets. Due to space restrictions, we only present adaptions for INC. We state which parts of INC we adapt to compute the number of optimal answer sets in Algorithm 4 (#OINC). To slightly simplify the presentation of optimization rules, we assume without loss of generality that whenever an atom a is introduced in bag $\chi(t)$ for some node t of the TD, the optimization rule r, where a occurs, belongs to the bag $\chi(t)$. First, we explain how to handle costs making use of function $\mathrm{cst}(\Pi, M, A)$ as defined in Section 2. In a leaf (Line 1) we set the (current) cost to 0. If we introduce an atom a (Line 2–4) the cost depends on whether a is set to true or false in M and we add the cost of the "child" tuple. Removal of rules (Line 5–6) is trivial, as we only store the same values. If we remove an atom (Line 7–8), we compute the minimum costs only for tuples $\langle M_a^-, \sigma, \mathcal{C}, c, n\rangle$ where c is minimal among M_a^-, σ, $\mathcal{C}$, that is, for a multiset $\mathcal{S}$ we let $\mathrm{kmin}(\mathcal{S}) := \{\!\{\langle M_a^-, \sigma, \mathcal{C}, c, n\rangle \mid c = \min\{c' : \langle M_a^-, \sigma, \mathcal{C}, c', \cdot\rangle \in \mathcal{S}\}, \langle M_a^-, \sigma, \mathcal{C}, c, n\rangle \in \mathcal{S}\}\!\}$. We require a multiset notation for counting (see below). If we join two nodes (Line 9–11), we compute the minimum value in the table of one child plus the minimum value of the table of the other child minus the value of the cost for the current bag, which is exactly the value we added twice. Next, we explain how to handle the number of witnessed sets that are minimal with respect to the cost. In a leaf (Line 1), we set the counter to 1. If we introduce/remove a rule or introduce an atom (Line 2–6), we can simply take the number n from the child. If we remove an atom (Line 7–8) we first obtain a multiset from computing kmin, which can contain several tuples for $M_a^-, \sigma, \mathcal{C}, c$ as we obtained M_a^- either from $M \setminus \{a\}$ if $a \in M$ or M if $a \notin M$ giving rise multiple solutions, that is, $\mathrm{cnt}(\mathcal{S}) := \{\langle M, \sigma, \mathcal{C}, c, \sum_{\langle M,\sigma,\mathcal{C},c,n'\rangle \in \mathcal{S}} n'\rangle \mid \langle M, \sigma, \mathcal{C}, c, n\rangle \in \mathcal{S}\}$. If we join nodes (Line 7–9), we multiply the number n' from the tuple of one

	2Col	3Col	Ds	St	cVc	sVc
Clasp(usc)	31.72 (21)	0.10(0)	8.99 (3)	0.21 (0)	29.88 (21)	98.34 (71)
DynASP2(PRIM)	1.54 (0)	0.53(0)	0.68 (0)	79.36 (221)	0.99 (0)	1.30 (0)
DynASP2(INC)	1.43 (0)	0.58(0)	0.54 (0)	115.02 (498)	0.68 (0)	0.78 (0)

Table 1 Runtimes (given in sec.; #timeouts in brackets) on real-world instances.

child with the number n'' from the tuple of the other child, restrict results with respect to minimum costs, and sum up the resulting numbers.

Corollary 1. *Given a program Π, algorithm #O*INC *runs in time $\mathcal{O}(log(m) \cdot 2^{2^{k+2} \cdot \ell^{k+1}} \|I(\Pi)\|^2)$, where $k := tw(I(\Pi))$, $\ell := max\{3, \mathrm{bnd}(r) : r \in \mathrm{WGT}(\Pi)\}$, and $m := \Sigma_{r \in \mathrm{OPT}(\Pi)} \mathrm{wght}(r)$.*

4 Experimental Evaluation

We implemented the algorithms $\mathcal{DP}_{\mathsf{PRIM}}$ and $\mathcal{DP}_{\mathsf{INC}}$ into a prototypical solver DynASP2(·) and performed experiments to evaluate its runtime behavior. Clearly, we *cannot* hope to solve programs with graph representations of high treewidth. However, programs involving real-world graphs such as graph problems on transit graphs admit TDs of small width. We used both random and structured instances for our benchmarks[5], see also [8]. The random instances (Sat-TGrid, 2QBF-TGrid, ASP-TGrid, 2ASP-TGrid) were designed to have a high number of variables and solutions and treewidth at most three. The structured instances model various graph problems (2Col, 3Col, Ds, St cVc, sVc) on real world mass transit graphs. For a graph, program 2Col counts all 2-colorings, 3Col counts all 3-colorings, Ds counts all minimal dominating sets, St counts all Steiner trees, cVc counts all cardinality-minimal vertex covers, and sVc counts all subset-minimal vertex covers. In order to draw conclusions about the efficiency of DynASP2, we mainly inspected the cpu running time and number of timeouts using the average over three runs per instance (three fixed seeds allow certain variance [1] for heuristic TD computation). We limited available memory (RAM) to 4GB (to run SharpSAT on large instances), and cpu time to 300 seconds, and then compared DynASP2 with the dedicated #SAT solvers SharpSAT [14] and Cachet [12], the QBF solver DepQBF0, and the ASP solver Clasp. Figure 4 illustrates runtime results as a cactus plot. Table 1 reports on the average running times, numbers of solved instances and timeouts on the structured instance sets.

Summary. Our empirical benchmark results confirm that DynASP2 exhibits competitive runtime behavior if the input instance has small treewidth. Compared to modern Asp and Qbf solvers, DynASP2 has an advantage in case of many solutions, whereas Clasp and DepQBF0 perform well if the number of solutions is relatively small. However, DynASP2 is still reasonably fast on structured instances with few solutions as it yields the result mostly within less than 10

[5] https://github.com/daajoe/lpnmr17_experiments

sec. We observed that INC seems to be the better algorithm in our setting, indicating that the smaller width obtained by decomposing the incidence graph generally outweighs the benefits of simpler solving algorithms for the primal graph. However, if INC and PRIM run with graphs of similar width, PRIM benefits from its simplicity. A comparison to #SAT solvers suggests that, on random instances, they have a lower overhead (which is not surprising, since our algorithms are built for ASP), but after about 150 sec. our algorithms solve more instances.

5 Conclusion

We presented novel DP algorithms for ASP, extending previous work [9] in order to cover the full ASP syntax. Our algorithms are based on two graph representations of programs and run in linear time with respect to the treewidth of these graphs and weights used in the program. Experiments indicate that our approach seems to be suitable for practical use, at least for certain classes of instances with low treewidth, and hence could fit into a portfolio solver. A further use of our techniques could be for extensions of ASP such as HEX [6].

References

1. M. Abseher, F. Dusberger, N. Musliu, and S. Woltran. Improving the efficiency of dynamic programming on tree decompositions via machine learning. In *IJCAI'15*.
2. H. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.
3. G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
4. F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-core-2 input language format, 2013.
5. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
6. T. Eiter, M. Fink, G. Ianni, T. Krennwallner, and P. Schüller. Pushing efficient evaluation of hex programs by modular decomposition. In *LPNMR'11*, 2011.
7. J. K. Fichte and S. Szeider. Backdoors to tractable answer-set programming. *Artificial Intelligence*, 220:64–103, 2015.
8. J. K. Fichte, M. Hecher, M. Morak, and S. Woltran. Answer set solving with bounded treewidth revisited. *CoRR*, abs/cs/arXiv:1702.02890, 2017.
9. M. Jakl, R. Pichler, and S. Woltran. Answer-set programming with bounded treewidth. In *IJCAI'09*, volume 2, 2009.
10. R. Pichler, S. Rümmele, S. Szeider, and S. Woltran. Tractable answer-set programming with weight constraints: bounded treewidth is not enough. *Theory Pract. Log. Program.*, 14(2), 2014.
11. M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1), 2010.
12. T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *SAT'04*, 2004.
13. T. Syrjänen. Lparse 1.0 user's manual. tcs.hut.fi/Software/smodels/lparse.ps, 2002.
14. M. Thurley. sharpSAT – counting models with advanced component caching and implicit BCP. In *SAT'06*, 2006.