The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Beyond NP: Technical Report WS-16-05

Subset Minimization in Dynamic Programming on Tree Decompositions

Bernhard Bliem and **Günther Charwat** and **Markus Hecher** and **Stefan Woltran**
Institute of Information Systems 184/2, TU Wien
Favoritenstrasse 9–11, 1040 Vienna, Austria
[bliem,gcharwat,hecher,woltran]@dbai.tuwien.ac.at

Abstract

Many problems from the area of AI have been shown tractable for bounded treewidth. In order to put such results into practice, quite involved dynamic programming (DP) algorithms on tree decompositions have to be designed and implemented. These algorithms typically show recurring patterns that call for tasks like subset minimization. In this paper, we provide a new method for obtaining DP algorithms from simpler principles, where the necessary data structures and algorithms for subset minimization are automatically generated. Moreover, we discuss how this method can be implemented in systems that perform more space-efficiently than current approaches.

Introduction

Many prominent NP-hard problems in the area of AI have been shown tractable for bounded treewidth. Thanks to Courcelle's theorem (Courcelle 1990), it is sufficient to encode a problem as an MSO sentence in order to obtain such a result. To put this into practice, tailored systems for MSO logic are required, however. While there has been remarkable progress in this direction (Kneis, Langer, and Rossmanith 2011) there is still evidence that designing DP algorithms for the considered problems from scratch results in more efficient software solutions (cf. (Niedermeier 2006)).

The actual design of these algorithms can be quite tedious, especially for problems located at the second level of the polynomial hierarchy like the AI problems circumscription, abduction, answer set programming or abstract argumentation (see (Dvořák, Pichler, and Woltran 2012; Jakl, Pichler, and Woltran 2009; Jakl et al. 2008; Gottlob, Pichler, and Wei 2010)). In many cases, the increased complexity of such problems is caused by subset minimization or maximization subproblems (e.g., minimality of models in circumscription). It is exactly the handling of these subproblems that makes the design of the DP algorithms difficult.

What we aim for in this paper is thus the automatic generation of intricate DP algorithms from simpler principles. To the best of our knowledge, there is only a little amount of work in this direction. The D-FLAT system (Abseher et al. 2014) – a declarative framework for rapid prototyping of DP algorithms on tree decompositions – offers a few built-ins for cost minimization and the handling of join nodes in standard DP algorithms; the LISP-based Autograph approach (see, e.g., (Courcelle and Durand 2013)) on the other hand makes it possible to obtain a specification of the problem at hand via combinations of (pre-defined) fly-automata.

What we have in mind is different and motivated by recent developments in the world of answer set programming (ASP) (Brewka, Eiter, and Truszczyński 2011): For exploiting the full expressive power of ASP, a saturation programming technique (see, e.g., (Leone et al. 2006)) is often required for the encoding of co-NP subproblems. Several approaches for relieving the user from this task have been proposed (Eiter and Polleres 2006; Gebser, Kaminski, and Schaub 2011; Brewka et al. 2015) that employ metaprogramming techniques. For instance, in order to compute minimal models of a propositional formula, one can simply express the SAT problem in ASP together with a special minimize statement (recognized by systems like *metasp*). In this way, one obtains a program computing minimal models. Unfortunately, easy-to-use facilities like such minimize statements had no analog in the area of DP so far.

In this paper, we propose a solution to this issue: We provide a method for automatically obtaining DP algorithms for problems requiring minimization, given only an algorithm for a problem variant without minimization. For example, given a DP algorithm for SAT (Samer and Szeider 2010), our approach enables us to generate a new algorithm for finding only subset-minimal models. Making minimization implicit in this way makes the programmer's life considerably easier.

The contributions of this paper are the following:

- We introduce a formal model of DP computations, abstracting from concrete algorithms. Our results are therefore generally applicable, not just to a particular problem.
- We show how our model captures typical DP computations for subset minimization problems.
- We discuss how computations can be compressed to ensure fixed-parameter tractability.
- Our main contribution is a formal definition of a transformation that turns non-minimizing computations into ones that perform minimization. We identify under which conditions this procedure is sound and give a formal proof.
- Finally, we discuss implementation issues. Compared to naive DP algorithms with minimization (which often suffer

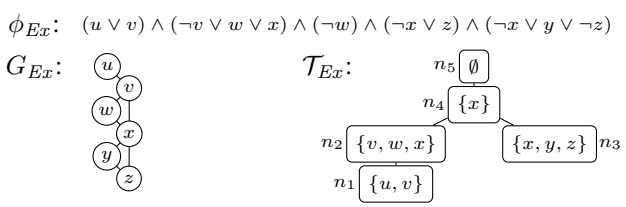

Figure 1: Primal graph G_{Ex} and a TD $\mathcal{T}_{Ex}$ of ϕ_{Ex}.

from a naive check for subset minimality) we propose an implementation that avoids redundant computations.

Background

In this section we outline DP on tree decompositions. The ideas underlying this concept stem from the field of parameterized complexity. Many computationally hard problems become tractable in case a certain problem parameter is bound to a fixed constant. This property is referred to as *fixed-parameter tractability* (Downey and Fellows 1999), and the complexity class FPT consists of problems that are solvable in $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a function that only depends on the parameter k, and n is the input size.

For problems whose input can be represented as a graph, an important parameter is *treewidth*, which measures "tree-likeness" of a graph. It is defined by means of tree decompositions (TDs) (Robertson and Seymour 1984).

Definition 1. *A* tree decomposition *of a graph $G = (V, E)$ is a pair $\mathcal{T} = (T, \chi)$ where $T = (N, F)$ is a (rooted) tree and $\chi : N \to 2^V$ assigns to each node a set of vertices (called the node's* bag*), such that the following conditions are met: (1) For every $v \in V$, there exists a node $n \in N$ such that $v \in \chi(n)$. (2) For every edge $e \in E$, there exists a node $n \in N$ such that $e \subseteq \chi(n)$. (3) For every $v \in V$, the subtree of T induced by $\{n \in N \mid v \in \chi(n)\}$ is connected.*

The width *of $\mathcal{T}$ is $\max_{n \in N} |\chi(n)| - 1$. The* treewidth *of a graph is the minimum width over all its tree decompositions.*

Although constructing a minimum-width TD is intractable in general (Arnborg, Corneil, and Proskurowski 1987), it is in FPT (Bodlaender 1996) and there are polynomial-time heuristics giving "good" TDs (Dechter 2003; Dermaku et al. 2008; Bodlaender and Koster 2010).

Example 2. *Let us consider the enumeration variant of the* SAT *problem. Given a propositional formula ϕ in CNF, we first have to find an appropriate graph representation. Here, we construct the* primal *graph G of ϕ, that is, vertices in G represent atoms of ϕ, and atoms occurring together in a clause form a clique in G. An example formula ϕ_{Ex}, its graph representation G_{Ex} and a possible TD $\mathcal{T}_{Ex}$ are given in Figure 1. The width of $\mathcal{T}_{Ex}$ is 2.*

TD-based DP algorithms generally traverse the TD in post-order. At each node, partial solutions for the subgraph induced by the vertices encountered so far are computed and stored in a data structure associated with the node. The size of the data structure is typically bounded by the TD's width and the number of TD nodes is linear in the input size. So if

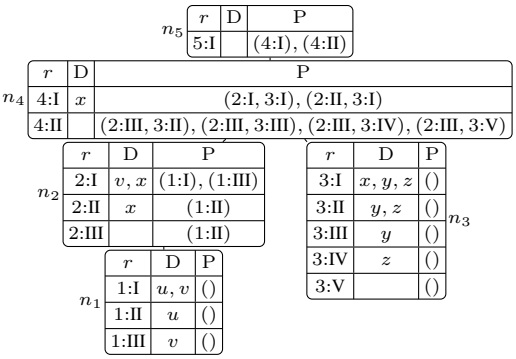

Figure 2: DP computation for SAT.

the width is bounded by a constant, the search space for subproblems is constant as well, and the number of subproblems only grows linearly for larger instances. We now illustrate the DP for SAT (Samer and Szeider 2010) on our running example; formal details are given in the next section.

Example 3. *The tables in Figure 2 are computed as follows. For a TD node n, each row r stores data $\mathrm{D}(r)$ that contains partial truth assignments over atoms in $\chi(n)$. Here, $\mathrm{D}(r)$ only contains atoms that get assigned "true", atoms in $\chi(n) \setminus \mathrm{D}(r)$ get assigned "false". In r, all clauses covered by $\chi(n)$ must be satisfied by the partial truth assignment. The set $\mathrm{P}(r)$ contains so-called extension pointer tuples (EPTs) that denote the rows in the children where r was constructed from. First consider node n_1: here, $\chi(n_1) = \{u, v\}$ covers clause $(u \vee v)$, yielding three partial assignments for ϕ_{Ex}. In n_2, the child rows are extended and the partial assignments are updated (by removing atoms not contained in $\chi(n_2)$ and guessing truth assignments for atoms in $\chi(n_2) \setminus \chi(n_1)$). Here, clauses $(\neg v \vee w \vee x)$ and $(\neg w)$ must be satisfied. In n_3 we proceed as before. In n_4, we join only partial assignments that agree on the truth assignment for common atoms. We continue like this until we reach the TD's root.*

To decide satisfiability of a formula, it suffices to check if the table in the root node is non-empty. The overall procedure is in FPT time as the number of TD nodes is bounded by the input size (i.e., the number of atoms) and each node n is has a table of size at most $\mathcal{O}(2^{\chi(n)})$ (i.e., the possible truth assignments). To enumerate models of ϕ_{Ex} with linear delay, we start at the root and follow the EPTs while combining the partial assignments associated with the rows. For instance, we obtain $\{u, v, x, y, z\}$ (i.e., the model where atoms $\{u, v, x, y, z\}$ are true, and $\{w\}$ is false) is constructed by starting at $5{:}I$ and following EPTs $(4{:}I)$, $(2{:}I, 3{:}I)$ and $(1{:}I)$.

A Formal Account of DP on TDs

The algorithms that are of interest in this paper take a problem instance along with a corresponding TD as input and use DP to produce a table at each TD node such that existence of solutions can be determined by examining the root table. Such algorithms obtain complete solutions by recursively combining rows with their predecessors from child nodes (cf. (Niedermeier 2006)). We call the resulting tree of tables

a *computation*, which we now formalize.

Definition 4. *A* computation *is a rooted ordered tree whose nodes are called* tables*. Each table R is a set of* rows *and each row $r \in R$ possesses*

- *some problem-specific* data $\mathrm{D}(r)$,
- *a non-empty set of* extension pointer tuples *(EPTs)* $\mathrm{P}(r)$ *such that each tuple is of arity k, where k is the number of children of R, and for each $(p_1, \ldots, p_k) \in \mathrm{P}(r)$ it holds that each p_i is a row of the i-th child of R,*
- *a* subtable $\mathrm{S}(r)$*, which is a set of* subrows*, where each subrow $s \in \mathrm{S}(r)$ possesses*
  - *some problem-specific data* $\mathrm{D}(s)$,
  - *a non-empty set of EPTs* $\mathrm{P}(s)$ *such that for each $(p_1, \ldots, p_k) \in \mathrm{P}(s)$ there is some $(q_1, \ldots, q_k) \in \mathrm{P}(r)$ with $p_i \in \mathrm{S}(q_i)$ for $1 \leq i \leq k$,*
  - *an inclusion status flag* $\mathrm{inc}(s) \in \{\mathrm{eq}, \subset\}$.

For rows or subrows a, b we write $a \approx b$, $a \leq b$ and $a < b$ to denote $\mathrm{D}(a) = \mathrm{D}(b)$, $\mathrm{D}(a) \subseteq \mathrm{D}(b)$ and $\mathrm{D}(a) \subset \mathrm{D}(b)$, respectively. For sets of rows or subrows R, S we write $\mathrm{D}(R)$ to denote $\bigcup_{r \in R} \mathrm{D}(r)$, and we write $R \approx S$, $R \leq S$ and $R < S$ to denote $\mathrm{D}(R) = \mathrm{D}(S)$, $\mathrm{D}(R) \subseteq \mathrm{D}(S)$ and $\mathrm{D}(R) \subset \mathrm{D}(S)$, respectively.

The reason that each row possesses a subtable is that we consider subset-optimization problems, and we assume that algorithms for such problems use subtables to store potential counterexamples to a solution candidate being subset-minimal. The intuition of each subrow s of a row r is that s represents all solution candidates that are subsets of the candidates represented by r. If one of these subset relations is proper, we indicate this by $\mathrm{inc}(s) = \subset$.

Example 5. *Let us now consider* $\subseteq$-MINIMAL SAT *(i.e., enumerating models that are subset-minimal w.r.t. the atoms that get assigned "true"). Figure 3 illustrates the computation for parts of our example. At n_1, R is computed as before. For any $r \in R$, each subrow $s \in \mathrm{S}(r)$ represents a partial "true" assignment that is a subset of the one in r (i.e., $\mathrm{D}(s) \subseteq \mathrm{D}(r)$), and $\mathrm{inc}(s)$ is set appropriately. Now consider* 2:*I*. *Here,* 2:*I*:1 *represents the same partial assignment (therefore marked with* eq*). However, although* 2:*I*:4 $\approx$ 2:*I, we have* $\mathrm{inc}(2{:}I{:}4) = \subset$ *since* $\mathrm{P}(2{:}I{:}4) = \{(1{:}I{:}3)\}$ *with* $\mathrm{inc}(1{:}I{:}3) = \subset$ *(i.e.,* $\mathrm{inc}(2{:}I{:}4) = \subset$ *because* $\mathrm{D}(2{:}I{:}4) \cup \mathrm{D}(1{:}I{:}3) = \{v, x\}$ *is a subset of* $\mathrm{D}(2{:}I) \cup \mathrm{D}(1{:}I) = \{u, v, x\}$*). Furthermore, consider* 2:*II and* 2:*III. They both stem from* 1:*II, but yield different rows since x is either contained in the partial "true" assignment, or not. Thus, also the subrows differ.*

The EPTs of a table row r are used for recursively combining the problem-specific data $\mathrm{D}(r)$ with data from "compatible" rows that are in descendant tables. The fact that each set of EPTs is required to be non-empty entails that for each (sub)row r at a leaf table it holds that $\mathrm{P}(r) = \{()\}$. We disallow rows with an empty set of EPTs because in the end we are only interested in rows than can be extended to complete solutions, consisting of one row per table. For this we introduce the notion of an *extension* of a table row.

Definition 6. *Let $\mathcal{C}$ be a computation and R be a table in $\mathcal{C}$ with k children. We inductively define the* extensions *of a row $r \in R$ as* $\mathrm{E}(r) = \{\{r\} \cup A \mid A \in \bigcup_{(p_1,\ldots,p_k) \in \mathrm{P}(r)} \{X_1 \cup \cdots \cup X_k \mid X_i \in \mathrm{E}(p_i) \text{ for all } 1 \leq i \leq k\}\}$.

n_2

R			$\mathrm{S}(r)$			
r	D	P	s	D	P	inc
2:I	v, x	(1:I)	2:I:1	v, x	(1:I:1)	eq
			2:I:2	x	(1:I:2)	$\subset$
			2:I:3		(1:I:2)	$\subset$
			2:I:4	v, x	(1:I:3)	$\subset$
2:II	x	(1:II)	2:II:1	x	(1:II:1)	eq
			2:II:2		(1:II:1)	$\subset$
2:III		(1:II)	2:III:1		(1:II:1)	eq
2:IV	v, x	(1:III)	2:IV:1	v, x	(1:III:1)	eq

n_1

R			$\mathrm{S}(r)$			
r	D	P	s	D	P	inc
1:I	u, v	()	1:I:1	u, v	()	eq
			1:I:2	u	()	$\subset$
			1:I:3	v	()	$\subset$
1:II	u	()	1:II:1	u	()	eq
1:III	v	()	1:III:1	v	()	eq

Figure 3: (Partial) DP computation for $\subseteq$-MINIMAL SAT.

Note that any extension $X \in \mathrm{E}(r)$ contains r and exactly one row from each table that is a descendant of R. If r is a row of a leaf table, $\mathrm{E}(r) = \{\{r\}\}$ because $\mathrm{P}(r) = \{()\}$.

While the extensions from the root table of a computation represent complete solution candidates, the purpose of subtables is to represent possible counterexamples that would cause a solution candidate to be invalidated. More precisely, for each extension X that can be obtained by extending a root table row r, we check if we can find an extension Y of an element $s \in \mathrm{S}(r)$ with $\mathrm{inc}(s) = \subset$ such that every element of Y is listed as a subrow of a row in X (i.e., we check if for every $y \in Y$ there is some $x \in X$ with $y \in \mathrm{S}(x)$). If this is so, then Y witnesses that X represents no solution because Y then represents a solution candidate that is a proper subset. For this reason, we need to introduce the notion of extensions (like Y) relative to another extension (like X).

Definition 7. *Let $\mathcal{C}$ be a computation, R be a table in $\mathcal{C}$ with k children, $r \in R$ be a row and $s \in \mathrm{S}(r)$ be a subrow of r. We first define, for any $X \in \mathrm{E}(r)$, a restriction of* $\mathrm{P}(s)$ *to EPTs where each element is a subrow of a row in X, as* $\mathrm{P}_X(s) = \{(p_1, \ldots, p_k) \in \mathrm{P}(s) \mid r_i \in X,\ p_i \in \mathrm{S}(r_i) \text{ for all } 1 \leq i \leq k\}$*. Now we define the set of* extensions *of s* relative to *some extension $X \in \mathrm{E}(r)$ as* $\mathrm{E}_X(s) = \{\{s\} \cup A \mid A \in \bigcup_{(p_1,\ldots,p_k) \in \mathrm{P}_X(s)} \{Y_1 \cup \cdots \cup Y_k \mid Y_i \in \mathrm{E}_X(p_i) \text{ for all } 1 \leq i \leq k\}\}$.

We can now formalize that the solutions of a computation are the extensions of those rows that do not have a subrow indicating a counterexample.

Definition 8. *Let R be the root table in a computation $\mathcal{C}$. We define the set of* solutions *of $\mathcal{C}$ as* $\mathrm{sol}(\mathcal{C}) = \{\mathrm{D}(X) \mid r \in R,\ X \in \mathrm{E}(r),\ \nexists s \in \mathrm{S}(r) : \mathrm{inc}(s) = \subset\}$

Example 9. *If n_2 in Figure 3 were the root of the TD, only* 2:*III and* 2:*IV would yield solutions. We would obtain $\{u\}$ and $\{v, x\}$, respectively. These represent indeed the subset-*

minimal models of the formula consisting of the clauses encountered until n_2, i.e., $(u \vee v) \wedge (\neg v \vee w \vee x) \wedge (\neg w)$.

Next we formalize requirements on subrows and their inclusion status to ensure that subrows correspond to subsets of their parent row, that each potential counterexample is represented by a subrow and that $\mathrm{inc}(\cdot)$ is used as intended.

Definition 10. *A table R is* normal *if the following properties hold:*

1. *For each $r \in R$, $s \in \mathrm{S}(r)$, $X \in \mathrm{E}(r)$ and $Y \in \mathrm{E}_X(s)$, it holds that $Y \leq X$, and $Y < X$ holds if and only if $\mathrm{inc}(s) = \subset$.*
2. *For each $r \in R$, $s \in \mathrm{S}(r)$ and $Y \in \mathrm{E}(s)$ there is some $r' \in R$ and $X' \in \mathrm{E}(r')$ such that $s \approx r'$ and $Y \approx X'$.*
3. *For each $q, r \in R$, $Z \in \mathrm{E}(q)$ and $X \in \mathrm{E}(r)$, if $Z \leq X$ holds, then there is some $s \in \mathrm{S}(r)$ and $Y \in \mathrm{E}_X(s)$ with $s \approx q$ and $Y \approx Z$.*

A computation is normal *if all its tables are normal.*

This definition ensures that it suffices to examine the root table of a normal computation in order to decide a subset minimization problem correctly, provided that the rows represent all solution candidates.

We will later show how a non-minimizing computation (i.e., one with empty subtables) satisfying certain properties can be transformed into a normal computation. In this transformation, we must avoid redundancies lest we destroy fixed-parameter tractability. For this, we first introduce how tables can be compressed without losing solution candidates.

Table Compression

To compress tables by merging equivalent (sub)rows, which is required for keeping the size of the tables bounded by the treewidth, we first define an equivalence relation on rows, as well as one on subrows.

Definition 11. *Let R be a table and $r \in R$. We define an equivalence relation $\equiv_r$ over subrows of r such that $s_1 \equiv_r s_2$ if $s_1 \approx s_2$ and $\mathrm{inc}(s_1) = \mathrm{inc}(s_2)$.*

We use this notion of equivalence between subrows to compress subtables by merging equivalent subrows.

Definition 12. *Let R be a table and $r \in R$. We define a subtable $\mathrm{S}^*(r)$ called the* compressed subtable *of r that contains exactly one subrow for each $\equiv_r$-equivalence class. For any $s \in \mathrm{S}(r)$, let $[s]$ denote the $\equiv_r$-equivalence class of s and let s' denote the subrow in $\mathrm{S}^*(r)$ corresponding to $[s]$. We define s' by $s' \approx s$, $\mathrm{inc}(s') = \mathrm{inc}(s)$ and $\mathrm{P}(s') = \bigcup_{t \in [s]} \mathrm{P}(t)$.*

Once subtables have been compressed, we can compress the table by merging equivalent rows. For this, we first need a notion of equivalence between rows.

Definition 13. *We define an equivalence relation $\equiv_R$ over rows of a table R such that $r_1 \equiv_R r_2$ if $r_1 \approx r_2$ and there is a bijection $f : \mathrm{S}^*(r_1) \to \mathrm{S}^*(r_2)$ such that for any $s \in \mathrm{S}^*(r_1)$ it holds that $s \approx f(s)$ and $\mathrm{inc}(s) = \mathrm{inc}(f(s))$.*

When rows are equivalent, their compressed subtables only differ in the EPTs. We now define how such compressed subtables can be merged.

Definition 14. *Let R be a table, $r \in R$, and let $[r]$ denote the $\equiv_R$-equivalence class of r. For any $r' \in [r]$, let $f_{r'} : \mathrm{S}^*(r) \to \mathrm{S}^*(r')$ be the bijection such that for any $s \in \mathrm{S}^*(r)$ it holds that $s \approx f_{r'}(s)$ and $\mathrm{inc}(s) = \mathrm{inc}(f_{r'}(s))$. (The existence of $f_{r'}$ is guaranteed by Definition 13.) We define a subtable $\mathrm{mst}([r])$ (for "merged subtable") that contains exactly one subrow for each element of $\mathrm{S}^*(r)$. For any $s \in \mathrm{S}^*(r)$, let s' denote the subrow in $\mathrm{mst}([r])$ corresponding to s. We define s' by $s' \approx s$, $\mathrm{inc}(s') = \mathrm{inc}(s)$ and $\mathrm{P}(s') = \bigcup_{r' \in [r]} \mathrm{P}(f_{r'}(s))$.*

We use these equivalence relations to compress tables in such a way that all equivalent (sub)rows (according to the respective equivalence relation) are merged.

Definition 15. *Let R be a table. We now define a table $\mathrm{compr}(R)$ that contains exactly one row for each $\equiv_R$-equivalence class. For any $r \in R$, let $[r]$ denote the $\equiv_R$-equivalence class of r and let r' be the row in $\mathrm{compr}(R)$ corresponding to $[r]$. We define r' by $r' \approx r$, $\mathrm{P}(r') = \bigcup_{q \in [r]} \mathrm{P}(q)$ and $\mathrm{S}(r') = \mathrm{mst}([r])$. For any computation $\mathcal{C}$, we write $\mathrm{compr}(\mathcal{C})$ to denote the computation isomorphic to $\mathcal{C}$ where each table R in $\mathcal{C}$ corresponds to $\mathrm{compr}(R)$.*

Lemma 16. *If a table R is normal, then so is $\mathrm{compr}(R)$.*

Proof sketch. Let R be a normal table and $R' = \mathrm{compr}(R)$. We prove conditions 1–3 of normality of R' separately. Let $r' \in R'$, $s' \in \mathrm{S}(r')$, $X' \in \mathrm{E}(r')$ and $Y'_{X'} \in \mathrm{E}_{X'}(s')$. Then we can find $r \in R$, $s \in \mathrm{S}(r)$, $X \in \mathrm{E}(r)$ and $Y \in \mathrm{E}_X(s)$ such that $r \approx r'$, $s \approx s'$, $X \approx X'$ and $Y \approx Y'_{X'}$. As R is normal, $Y \leq X$ holds, and $Y < X$ if and only if $\mathrm{inc}(s) = \subset$. This entails $Y'_{X'} \leq X'$, and $Y'_{X'} < X'$ if and only if $\mathrm{inc}(s') = \subset$ because $\mathrm{inc}(s') = \mathrm{inc}(s)$. This proves condition 1. Let $Y' \in \mathrm{E}(s')$. Then we can find $Y \in \mathrm{E}(s)$ such that $Y \approx Y'$. As R is normal, there is a row $t \in R$ with $t \approx s$ and an extension $T \in \mathrm{E}(t)$ such that $T \approx Y$. Then there is some $t' \in R'$ with $t' \approx t$ and $(T \setminus \{t\}) \cup \{t'\} \in \mathrm{E}(t')$, which proves condition 2. Let $q' \in R$ and $Z' \in \mathrm{E}(q')$ such that $Z' \leq X'$. Then we can find $q \in R$ and $Z \in \mathrm{E}(q)$ such that $Z \approx Z'$, so $Z \leq X$. As R is normal, there are $u \in \mathrm{S}(r)$ and $U \in \mathrm{E}_X(u)$ such that $u \approx q$ and $U \approx Z$. Then there is some $u' \in \mathrm{S}(r')$ with $u' \approx u$ and $(U \setminus \{u\}) \cup \{u'\} \in \mathrm{E}_{X'}(u')$, which proves condition 3. □

Normalizing Computations

Before we introduce our transformation from "non-minimizing" computations to "minimizing" ones, we define certain conditions that are prerequisites for the transformation. For this, we first define the set of all data of rows that have occurred in a table or any of its descendants.

Definition 17. *Let R be a table in a computation such that $R_1, \ldots, R_k$ are the child tables of R. We inductively define $\mathrm{D}^*(R) = \bigcup_{r \in R} D(r) \cup \bigcup_{1 \leq i \leq k} D^*(R_i)$.*

Now we define conditions that the tables in a computation must satisfy for being eligible for our transformation.

Definition 18. *Let R be a table in a computation such that $R_1, \ldots, R_k$ are the child tables of R, and let $r, r' \in R$. We say that $d \in \mathrm{D}(r)$ has been* illegally introduced *at r if there*

are $(r_1, \ldots, r_k) \in \mathrm{P}(r)$ *such that for some* $1 \leq i \leq k$ *it holds that* $d \notin \mathrm{D}(r_i)$ *while* $d \in \mathrm{D}^*(R_i)$*. Moreover, we say that* $d \in \mathrm{D}(r') \setminus \mathrm{D}(r)$ *has been* illegally removed *at* r *if there is some* $X \in \mathrm{E}(r)$ *such that* $d \in X$*.*

We now define the notion of an *augmentable* table, i.e., a table that can be used in our transformation.

Definition 19. *We call a table* R augmentable *if the following conditions hold:*

1. *For all rows* $r \in R$ *it holds that* $\mathrm{S}(r) = \emptyset$*.*
2. *For all* $r, r' \in R$ *with* $r \neq r'$ *it holds that* $\mathrm{D}(r) \neq \mathrm{D}(r')$*.*
3. *For all* $r \in R$*,* $(r_1, \ldots, r_k) \in \mathrm{P}(r)$*,* $1 \leq h < j \leq k$*,* $H \in \mathrm{E}(r_h)$ *and* $J \in \mathrm{E}(r_j)$ *it holds that* $\mathrm{D}(H) \cap \mathrm{D}(J) \subseteq \mathrm{D}(r)$*.*
4. *No element of* $\mathrm{D}(R)$ *has been illegally introduced.*
5. *No element of* $\mathrm{D}(R)$ *has been illegally removed.*

We call a computation augmentable *if all its tables are augmentable.*

These requirements are satisfied by reasonable TD-based DP algorithms (cf. (Niedermeier 2006)) as these usually do not put arbitrary data into the rows. Rather, the data in a row is typically restricted to information about bag elements of the respective TD node. For instance, condition 3 mirrors condition 3 of Definition 1, and condition 2 is usually satisfied by reasonable FPT algorithms because they avoid redundancies in order to stay fixed-parameter tractable.

Now we describe how augmentable computations can automatically be transformed into normal computations that take minimization into account. For any table R in an augmentable computation, this allows us to compute a new table $\mathrm{aug}(R)$ if for each child table R_i the table $\mathrm{aug}(R_i)$ has already been computed and compressed to $\mathrm{compr}(\mathrm{aug}(R_i))$.

Definition 20. *We inductively define a function* $\mathrm{aug}(\cdot)$ *that maps each table* R *from an augmentable computation to a table. Let the child tables of* R *be called* $R_1, \ldots, R_k$*. For any* $1 \leq i \leq k$ *and* $r \in R_i$*, we write* $\mathrm{res}(r)$ *to denote* $\{q \in \mathrm{compr}(\mathrm{aug}(R_i)) \mid q \approx r\}$*. We define* $\mathrm{aug}(R)$ *as the smallest table that satisfies the following conditions:*

1. *For any* $r \in R$*,* $(r_1, \ldots, r_k) \in \mathrm{P}(r)$ *and* $(c_1, \ldots, c_k) \in \mathrm{res}(r_1) \times \cdots \times \mathrm{res}(r_k)$*, there is a row* $q \in \mathrm{aug}(R)$ *with* $q \approx r$ *and* $\mathrm{P}(q) = \{(c_1, \ldots, c_k)\}$*.*
2. *For any* $q, q' \in \mathrm{aug}(R)$ *such that* $q' \leq q$*,* $\mathrm{P}(q) = \{(q_1, \ldots, q_k)\}$ *and* $\mathrm{P}(q') = \{(q'_1, \ldots, q'_k)\}$ *the following holds: If for all* $1 \leq i \leq k$ *there is some* $s_i \in \mathrm{S}(q_i)$ *with* $s_i \approx q'_i$*, then there is a subrow* $s \in \mathrm{S}(q)$ *with* $s \approx q'$ *and* $\mathrm{P}(s) = \{(s_1, \ldots, s_k)\}$*. Moreover,* $\mathrm{inc}(s) = \subset$ *if* $q' < q$ *or* $\mathrm{inc}(s_i) = \subset$ *for some* s_i*, otherwise* $\mathrm{inc}(s) = \mathrm{eq}$*.*

For any augmentable computation $\mathcal{C}$*, we write* $\mathrm{aug}(\mathcal{C})$ *to denote the computation isomorphic to* $\mathcal{C}$ *where each table* R *in* $\mathcal{C}$ *corresponds to* $\mathrm{aug}(R)$*.*

Augmentable tables never have two different rows r, r' with $\mathrm{D}(r) = \mathrm{D}(r')$. Moreover, we defined $\mathrm{aug}(R)$ in such a way that for all $r \in R$ there is some $q \in \mathrm{aug}(R)$ with $r \approx q$. In the compression $\mathrm{compr}(\mathrm{aug}(R))$, we only merge rows and subrows having the same data. So with each row and subrow in $\mathrm{aug}(R)$ or $\mathrm{compr}(aug(R))$ we can associate a unique originating row in R. In fact, the extensions from an augmentable table R are in a one-to-one correspondence to the extensions of rows from $\mathrm{aug}(R)$. This is formalized by the following lemma. A proof of the lemma can be found in (Bliem et al. 2015a).

Lemma 21. *Let* R *be a table from an augmentable computation and* $Q = \mathrm{aug}(R)$*. Then for any* $r \in R$ *and* $Z \in \mathrm{E}(r)$ *there are* $q \in Q$ *and* $X \in \mathrm{E}(q)$ *such that* $r \approx q$ *and* $Z \approx X$*. Also, for any* $q \in Q$ *and* $X \in \mathrm{E}(q)$ *there are* $r \in R$ *and* $Z \in \mathrm{E}(r)$ *such that* $q \approx r$ *and* $X \approx Z$*.*

The following lemma is central for showing that $\mathrm{aug}(\cdot)$ works as intended. For a full proof, see (Bliem et al. 2015a).

Lemma 22. *Let* R *be a table from an augmentable computation. Then the table* $\mathrm{aug}(R)$ *is normal.*

Proof sketch. Let R be a table in some augmentable computation such that $R_1, \ldots, R_k$ denote the child tables of R and let $Q = \mathrm{aug}(R)$. We use induction. If Q is a leaf table, then rows and extensions coincide and the construction of Q obviously ensures that Q is normal. If Q has child tables $Q_i = \mathrm{compr}(\mathrm{aug}(R_i))$ and all $\mathrm{aug}(R_i)$ are normal, all Q_i are normal by Lemma 16. Let $q \in Q$, $s \in \mathrm{S}(q)$, $\mathrm{P}(q) = \{(q_1, \ldots, q_k)\}$, $\mathrm{P}(s) = \{(s_1, \ldots, s_k)\}$, $X_i \in \mathrm{E}(q_i)$, $Y_i \in \mathrm{E}_{X_i}(s_i)$, $X = \{q\} \cup X_1 \cup \cdots \cup X_k$ and $Y = \{s\} \cup Y_1 \cup \cdots \cup Y_k$. As for normality condition 1, the construction of Q ensures $s \leq q$ and normality of Q_i ensures $Y_i \leq X_i$, so $Y \leq X$. To show that $\mathrm{inc}(s)$ has the correct value, first suppose $Y < X$ and $\mathrm{inc}(s) = \mathrm{eq}$. The latter would entail $Y_i = X_i$, so $s < q$, but then $\mathrm{inc}(s) = \subset$, which is a contradiction. So suppose $X \approx Y$ and $\mathrm{inc}(s) = \subset$. If $q < s$, there would be an illegal removal at the origin of s in R, contradicting that R is augmentable. So for some j there is a $d \in \mathrm{D}(X_j) \setminus \mathrm{D}(Y_j)$. Due to $X \approx Y$, $d \in \mathrm{D}(s)$ or $d \in \mathrm{D}(Y_h)$ for some $h \neq j$. In the first case, there is an illegal introduction at the origin of s in R. In the other case, $d \in \mathrm{D}(Y_h)$ entails $d \in \mathrm{D}(X_h)$. As row extensions in Q are in a one-to-one correspondence with those in R, and by augmentability of R, $d \in \mathrm{D}(X_j) \cap \mathrm{D}(X_h)$ entails $d \in \mathrm{D}(q)$. But then $d \in \mathrm{D}(s)$, which we already led to a contradiction.

For condition 2, let $Z_i \in \mathrm{E}(s_i)$ and $Z = \{s\} \cup Z_1 \cup \cdots \cup Z_k$. As $s \in \mathrm{S}(q)$, there are $p \in Q$ and $(p_1, \ldots, p_i) \in \mathrm{P}(p)$ with $p \approx s$ and $p_i \approx s_i$ This entails existence of $r \in R$ and $(r_1, \ldots, r_k) \in \mathrm{E}(r)$ with $r \approx p \approx s$ and $r_i \approx p_i$. By hypothesis, there are $q'_i \in Q_i$ and $X'_i \in \mathrm{E}(q'_i)$ with $q'_i \approx s_i$ and $X'_i \approx Z_i$. Each q'_i originates from the unique $r_i \in R$ with $r_i \approx q'_i$. So $q'_i \in \mathrm{res}(r_i)$ holds and there are $q' \in Q$ and $X' \in \mathrm{E}(q')$ with $q' \approx r \approx s$ and $X' \approx Z$.

For condition 3, let $q' \in Q$, $\mathrm{P}(q') = \{(q'_1, \ldots, q'_k)\}$, $X' \in \mathrm{E}(q')$ and $X'_i \in \mathrm{E}(q'_i)$ for all $1 \leq i \leq k$. Suppose $X' \leq X$ and, for the sake of contradiction, for some j there is a $d \in \mathrm{D}(X'_j) \setminus \mathrm{D}(X_j)$. Then $d \in \mathrm{D}(q)$ or $d \in \mathrm{D}(X_h)$ for some $h \neq j$. In the first case, there is an illegal introduction at the origin of q in R. In the other case, $d \in \mathrm{D}(X_j) \cap \mathrm{D}(X_h)$ entails $d \in \mathrm{D}(q)$, which we already led to a contradiction. So $X'_i \leq X_i$ for each i. By hypothesis then there are $t_i \in \mathrm{S}(q_i)$ and $T_i \in \mathrm{E}_{X_i}(t_i)$ with $t_i \approx q'_i$ and $T_i \approx X'_i$. So there is a $t \in \mathrm{S}(q)$ with $t \approx q'$ and $\mathrm{P}(t) = \{(t_1, \ldots, t_k)\}$. Then $T = \{t\} \cup T_i \cup \cdots \cup T_k$ is in $\mathrm{E}_X(t)$ and $T \approx X'$. □

We can now state our main theorem, which says that exactly the subset-minimal solutions of an augmentable computation are solutions of the augmented computation.

Theorem 23. *Let $\mathcal{C}$ be an augmentable computation. Then* $\mathrm{sol}(\mathrm{aug}(\mathcal{C})) = \{S \in \mathrm{sol}(\mathcal{C}) \mid \nexists S' \in \mathrm{sol}(\mathcal{C}) : S' \subset S\}$.

Proof. Let R be the root table of an augmentable computation $\mathcal{C}$ and R' be the root table of $\mathcal{C}' = \mathrm{aug}(\mathcal{C})$. By Lemma 22, $\mathcal{C}'$ is normal. For the first direction, let $S \in \mathrm{sol}(\mathcal{C}')$. Then there are $r' \in R'$ and $X' \in \mathrm{E}(r')$ such that $\mathrm{D}(X') = S$ and for all $s' \in \mathrm{S}(r')$ it holds that $\mathrm{inc}(s') = \mathrm{eq}$. By Lemma 21, then there are $r \in R$ and $X \in \mathrm{E}(r)$ such that $X \approx X'$. As R is augmentable, $\mathrm{S}(r)$ is empty, so $S \in \mathrm{sol}(\mathcal{C})$ by Definition 8. We must now show that there is no solution in $\mathcal{C}$ smaller than S. For the sake of contradiction, suppose there is some $T \in \mathrm{sol}(\mathcal{C})$ with $T \subset S$. Then there are $q \in R$ and $Z \in \mathrm{E}(q)$ such that $\mathrm{D}(Z) = T$, hence $Z < X'$. By Lemma 21, then there are $q' \in R'$ and $Z' \in \mathrm{E}(q')$ such that $Z' \approx Z$. As R' is normal, due to $Z' < X'$, there is some $s' \in \mathrm{S}(r')$ such that $\mathrm{inc}(s') = \subset$. This contradicts $\mathrm{inc}(s') = \mathrm{eq}$, which we have seen earlier.

For the other direction, let $S \in \mathrm{sol}(\mathcal{C})$ be such that there is no $S' \in \mathrm{sol}(\mathcal{C})$ with $S' \subset S$. Then there are $r \in R$ and $X \in \mathrm{E}(r)$ such that $\mathrm{D}(X) = S$. By Lemma 21, then there are $r' \in R'$ and $X' \in \mathrm{E}(r')$ such that $X' \approx X$, and there are no $q' \in R'$ and $Z' \in \mathrm{E}(q')$ with $Z' < X$. Hence, as R' is normal, there cannot be a $s' \in \mathrm{S}(r')$ with $\mathrm{inc}(s') = \subset$. This proves that $S \in \mathrm{sol}(\mathcal{C}')$. □

Finally, we sketch that $\mathrm{aug}(\cdot)$ does not destroy fixed-parameter tractability.

Theorem 24. *Let $\mathcal{A}$ be an algorithm that takes as input an instance of size n and treewidth w along with a TD $\mathcal{T}$ of width w. Suppose $\mathcal{A}$ produces an augmentable computation $\mathcal{C}$ isomorphic to $\mathcal{T}$ in time $f(w) \cdot n^{\mathcal{O}(1)}$, where f is a function depending only on w. Then $\mathrm{aug}(\mathcal{C})$ can be computed in time $g(w) \cdot n^{\mathcal{O}(1)}$, where g again depends only on w.*

Proof sketch. We assume w.l.o.g. that each node in $\mathcal{T}$ has at most 2 children, as any TD can be transformed to this form in linear time without increasing the width (Kloks 1994). As $\mathcal{A}$ runs in FPT time, i.e., in $f(w) \cdot n^{\mathcal{O}(1)}$ for a function f, no table in $\mathcal{C}$ can be bigger than $f(w) \cdot n^c$ for a constant c. To inductively compute $Q = \mathrm{aug}(R)$ for some table R in $\mathcal{C}$ with child tables $R_1, \ldots, R_k$ ($k \leq 2$), suppose we have already constructed each $Q_i = \mathrm{aug}(R_i)$ in FPT time. Then $|Q_i| = f_i(w) \cdot n^{c_i}$ for some f_i and c_i. We can compute $Q'_i = \mathrm{compr}(Q_i)$ in time polynomial in $|Q_i|$. Then $|Q'_i| = f'_i(w) \cdot n^{c'_i}$ for some f'_i and c'_i. Definition 20 suggests a straightforward way to compute Q in time polynomial in $|R|$ and $\sum_{1 \leq i \leq k} |Q'_i|$. So we can compute Q in FPT time. As $\mathcal{T}$ has size $\mathcal{O}(n)$, we can compute $\mathrm{aug}(\mathcal{C})$ in FPT time. □

Practical Implementation Issues

Our definition of $\mathrm{aug}(\cdot)$ can be implemented to obtain a problem-independent framework whose input is (1) an algorithm $\mathcal{A}$ for solving a problem without subset minimization, (2) an instance of this problem and (3) a tree decomposition. By running $\mathcal{A}$ and transforming the resulting computation $\mathcal{C}$ into $\mathrm{aug}(\mathcal{C})$, we can be sure by Theorem 23 that the solutions are exactly the subset-minimal ones of $\mathcal{C}$.

Several optimizations are possible: For one, a counterexample candidate may turn out not to correspond to a solution candidate after all. A lot of resources could be saved by not storing such counterexample candidates in the first place. Our approach can be implemented in an optimized way by proceeding in stages: First we compute all rows without their subtables, then we delete rows that do not lead to solutions and finally we apply $\mathrm{aug}(\cdot)$ to the resulting tables. This way, we avoid storing counterexample candidates that appear in no extension at the root table.

Furthermore, our approach can easily be generalized for computing solutions where only a certain part of the data (instead of all data) is minimal among all candidates. We provided definitions and proofs only for the special case, as the generalization leads to more cumbersome notation and does not change the nature of the approach. Using this generalization, we are able to obtain DP algorithms for further AI problems like circumscription (McCarthy 1980).

We have implemented our approach using the mentioned optimizations. In (Bliem et al. 2015b) we have informally introduced the resulting system and illustrated how it allows us to specify DP algorithms for several AI problems like ASP, circumscription or abstract argumentation. The preliminary experiments reported there show that our approach indeed has significant advantages in terms of running time and especially memory compared to existing solutions for solving subset minimization problems on TDs. The current work complements that paper from a theoretical side: Here we have shown under which circumstances our approach is applicable, and we have proven its correctness.

Conclusion

To put FPT results for bounded treewidth to use in the AI domain, often DP algorithms for problems involving tasks like subset minimization have to be designed. These algorithms exhibit common properties that are tedious to specify but can be automatically taken care of, as we have shown in this paper. In fact, we have provided a translation that turns a given DP algorithm for computing a set S of solution candidates (say, models of a formula) into a DP algorithm that computes only the subset-minimal elements of S (e.g., minimal models). We have shown the translation to be sound and to remain FPT whenever the original DP is. This is indeed superior to a naive way that computes all elements from S first and then filters out minimal ones in a post-processing step, which would not yield an FPT algorithm in general. In a related paper, we have presented a system realizing this idea with further generalizations (e.g., performing minimization only on a given set of atoms). For future work, we would like to investigate the potential of other built-ins for DP algorithms; for instance, checks for connectedness could be treated in a similar way. In the long run, we anticipate a system that facilitates implementing DP algorithms but (in contrast to related systems such as *Sequoia*) keeps the overall design of the concrete DP algorithm in the user's hands.

References

Abseher, M.; Bliem, B.; Charwat, G.; Dusberger, F.; Hecher, M.; and Woltran, S. 2014. The D-FLAT system for dynamic programming on tree decompositions. In *Proc. JELIA*, volume 8761 of *LNCS*, 558–572.

Arnborg, S.; Corneil, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods* 8(2):277–284.

Bliem, B.; Charwat, G.; Hecher, M.; and Woltran, S. 2015a. D-FLATˆ2: Subset minimization in dynamic programming on tree decompositions made easy. Technical Report DBAI-TR-2015-93, DBAI, TU Wien.

Bliem, B.; Charwat, G.; Hecher, M.; and Woltran, S. 2015b. D-FLATˆ2: Subset minimization in dynamic programming on tree decompositions made easy. In *Proc. ASPOCP'15*.

Bodlaender, H. L., and Koster, A. M. C. A. 2010. Treewidth computations I. Upper bounds. *Inf. Comput.* 208(3):259–275.

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

Brewka, G.; Delgrande, J. P.; Romero, J.; and Schaub, T. 2015. asprin: Customizing answer set preferences without a headache. In Bonet, B., and Koenig, S., eds., *Proceedings of the 29th AAAI Conference on Artificial Intelligence, AAAI 2015*, 1467–1474. AAAI Press.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Courcelle, B., and Durand, I. 2013. Computations by fly-automata beyond monadic second-order logic. *CoRR* abs/1305.7120.

Courcelle, B. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.* 85(1):12–75.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Dermaku, A.; Ganzow, T.; Gottlob, G.; McMahan, B. J.; Musliu, N.; and Samer, M. 2008. Heuristic methods for hypertree decomposition. In *Proc. MICAI*, volume 5317 of *LNCS*, 1–11. Springer.

Downey, R. G., and Fellows, M. R. 1999. *Parameterized Complexity*. Monographs in Computer Science. Springer.

Dvořák, W.; Pichler, R.; and Woltran, S. 2012. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artif. Intell.* 186:1–37.

Eiter, T., and Polleres, A. 2006. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *TPLP* 6(1-2):23–60.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex optimization in answer set programming. *TPLP* 11(4-5):821–839.

Gottlob, G.; Pichler, R.; and Wei, F. 2010. Tractable database design and datalog abduction through bounded treewidth. *Inf. Syst.* 35(3):278–298.

Jakl, M.; Pichler, R.; Rümmele, S.; and Woltran, S. 2008. Fast counting with bounded treewidth. In *Proc. LPAR*, volume 5330 of *LNCS*, 436–450. Springer.

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

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

Kneis, J.; Langer, A.; and Rossmanith, P. 2011. Courcelle's theorem – a game-theoretic approach. *Discrete Optimization* 8(4):568–594.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.* 7(3):499–562.

McCarthy, J. 1980. Circumscription – a form of non-monotonic reasoning. *Artif. Intell.* 13(12):27–39.

Niedermeier, R. 2006. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. OUP.

Robertson, N., and Seymour, P. D. 1984. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B* 36(1):49–64.

Samer, M., and Szeider, S. 2010. Algorithms for propositional model counting. *J. Discrete Algorithms* 8(1):50–64.