ELSEVIER

Artificial Intelligence 138 (2002) 55–86

Artificial Intelligence

www.elsevier.com/locate/artint

Fixed-parameter complexity in AI and nonmonotonic reasoning ☆

Georg Gottlob [a], Francesco Scarcello [b,*], Martha Sideri [c]

[a] *Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria*
[b] *Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria, I-87036 Rende (CS), Italy*
[c] *Department of Informatics, Athens University of Economics and Business, GR-10434 Athens, Greece*

Received 15 May 2000

Abstract

Many relevant intractable problems become tractable if some problem parameter is fixed. However, various problems exhibit very different computational properties, depending on how the runtime required for solving them is related to the fixed parameter chosen. The theory of parameterized complexity deals with such issues, and provides general techniques for identifying fixed-parameter tractable and fixed-parameter intractable problems.

We study the parameterized complexity of various problems in AI and nonmonotonic reasoning. We show that a number of relevant parameterized problems in these areas are fixed-parameter tractable. Among these problems are constraint satisfaction problems with bounded treewidth and fixed domain, restricted forms of conjunctive database queries, restricted satisfiability problems, propositional logic programming under the stable model semantics where the parameter is the dimension of a feedback vertex set of the program's dependency graph, and circumscriptive inference from a positive k-CNF restricted to models of bounded size. We also show that circumscriptive inference from a general propositional theory, when the attention is restricted to models of bounded size, is fixed-parameter intractable and is actually complete for a novel fixed-parameter complexity class.

Keywords: Parameterized complexity; Fixed-parameter tractability; Nonmonotonic reasoning; Constraint satisfaction; Conjunctive queries; Prime implicants; Logic programming; Stable models; Circumscription

☆ Part of this work has been published in preliminary form in the proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '99), El Paso, TX, 1999.

* Corresponding author.
E-mail addresses: gottlob@dbai.tuwien.ac.at (G. Gottlob), scarcello@deis.unical.it (F. Scarcello), sideri@aueb.gr (M. Sideri).

0004-3702/02/$ – see front matter
PII: S0004-3702(02)00182-0

1. Introduction

Many hard decision or computation problems are known to become tractable if a problem parameter is fixed or bounded by a fixed value. For example, consider the vertex-cover problem. A vertex cover of a graph (V, E) is a set of vertices $S \subseteq V$ such that, for each edge $\{u, v\} \in E$, at least one of u and v belongs to S. Deciding whether there is a vertex cover of size at most k, and of computing such a vertex cover if so, are NP-hard problems [23]. Nevertheless, both problems become tractable if the integer k is a fixed constant, rather than being part of the problem instance. Similarly, the well known NP-hard problem of finding a clique of size k in a graph becomes tractable for every fixed k. Note, however, that there is an important difference between these problems:

- The vertex cover problem is solvable in linear time *for every fixed constant* k. Thus the problem is not only polynomially solvable for each fixed k, but, moreover, can be solved in time bounded by a polynomial p_k *whose degree does not depend on* k.
- The best known algorithms for finding a clique of size k in a graph all require time $n^{\Omega(k)}$. Thus, for fixed k, the problem is solvable in time bounded by a polynomial p_k, *whose degree depends crucially on* k.

Problems of the first type are called *fixed-parameter* (short: *fp*) *tractable*, while problems of the second type can be classified as *fixed-parameter intractable* [14]. It is clear that fixed-parameter tractability is a highly desirable feature.

The theory of *parameterized complexity*, mainly developed by Downey and Fellows [11, 12,14], deals with general techniques for proving that certain problems are fp-tractable, and with the classification of fp-intractable problems into a hierarchy of fixed-parameter complexity classes.

In this paper we study the fixed-parameter complexity of a number of relevant AI and NMR problems. In particular, we show that the following problems are all fixed-parameter tractable (the parameters to be fixed are added in square brackets after the problem description):

- Constraint Satisfiability and computation of the solution to a constraint satisfaction problem (CSP) [fixed parameters: (cardinality of) domain and treewidth of constraint scopes].
- Satisfiability of CNF [fixed parameter: treewidth of variable connection graph].
- Prime Implicants of a q-CNF [fixed parameters: maximal number q of literals per clause and size of the prime implicants to be computed].
- Propositional logic programming [fixed parameter: size of a minimal feedback vertex set of a suitable atom connection graph].
- Circumscriptive inference from a positive q-CNF [fixed parameters: maximal number q of literals per clause and size of the models to be considered].

We believe that these results are useful both for a better understanding of the computational nature of the above problems and for the development of smart parameterized algorithms for the solution of these and related problems.

We also study the complexity of circumscriptive inference from a general propositional theory when the attention is restricted to models of size k. This problem, referred-to as *small model circumscription* (*SMC*), is easily seen to be fixed-parameter intractable, but it does not seem to be complete for any of the fp-complexity classes defined by Downey and Fellows. We introduce the new class $\Sigma_2 W[SAT]$ as a miniaturized version of the class Σ_2^{P} of the polynomial hierarchy, and prove that SMC is complete for $\Sigma_2 W[SAT]$. This seems to be natural, given that the nonparameterized problem corresponding to SMC is Σ_2^{P}-complete [15]. Note, however, that completeness results for parameterized classes are more difficult to obtain. In fact, for obtaining our completeness result we had to resort to the general version of circumscription (called $P;Z$-circumscription) where the propositional letters of the theory to be circumscribed are partitioned into two subsets P and Z, and only the atoms in P are minimized, while those in Z can float. The restricted problem, where P consists of *all* atoms and Z is empty does not seem to be complete for $\Sigma_2 W[SAT]$, even though its nonparameterized version is still Σ_2^{P}-complete [15].

This paper is one of the first investigating fixed-parameter tractability in the context of AI. Very interesting work concerning the computation of stable models was recently carried out in [39]. There, the size of the stable models to be computed is taken as the fixed parameter, while, in the present paper, we consider the computation of stable models of any size, taking as the fixed parameter a suitable graph theoretical measure of the logic program. In [39], the following remarkable result is proven: computing small stable models is fixed-parameter intractable, whereas computing large stable models is fixed-parameter tractable if the parameter is the number of rules in the program. These results, orthogonal to ours, provide a wider picture of the fixed-parameter complexity of computing stable models of logic programs.

The rest of this paper is organized as follows. In Section 2, we state the relevant formal definitions related to fixed-parameter complexity. In Section 3, we deal with constraint satisfaction problems. In Section 4, we study fp-tractable satisfiability problems. In Section 5, we deal with logic programming. In Section 6, we study the problem of circumscriptive inference with small models. Conclusions are given in Section 7.

2. Parameterized complexity

Parameterized complexity [14] deals with parameterized problems, i.e., problems with an associated parameter. Any instance S of a parameterized problem P can be regarded as consisting of two parts: the "regular" instance I_S, which is usually the input instance of the classical—nonparameterized—version of P; and the associated parameter k_S, usually of integer type.

Definition 1. A parameterized problem P is *fixed-parameter tractable* if there is an algorithm that correctly decides, for input S, whether S is a yes instance of P in time $f(k_S)|I_S|^c$, where $|I_S|$ denotes the size of I_S, k_S is the parameter, c is a constant, and f is an arbitrary function.

A notion of problem reduction proper to the theory of parameterized complexity has been defined.

Definition 2. A parameterized problem P *fp-reduces* to a parameterized problem P' by an *fp-reduction* if there exist two functions f, f' and a constant c such that we can associate to any instance S of P an instance S' of P' satisfying the following conditions:

(1) the parameter $k_{S'}$ of S' is $f(k_S)$;
(2) the regular instance $I_{S'}$ is computable from S in time at most $f'(k_S)|I_S|^c$;
(3) S is a yes instance of P if and only if S' is a yes instance of P'.

A parameterized class of problems C is a (possibly infinite) set of parameterized problems. A problem P is C-complete if $P \in C$ and every problem $P' \in C$ is fp-reducible to P.[1]

A hierarchy of fp-intractable classes, called *W-hierarchy*, has been defined to properly characterize the degree of fp-intractability associated to different parameterized problems. The relationship among the classes of problems belonging to the W-hierarchy is given by the following chain of inclusions:

$$W[1] \subseteq W[2] \subseteq \cdots \subseteq W[SAT] \subseteq W[P]$$

where, for each natural number $t > 0$, the definition of the class $W[t]$ is based on the degree t of the complexity of a suitable family of Boolean circuits.

The most prominent $W[1]$-complete problem is the parameterized version of CLIQUE, where the parameter is the clique size. $W[1]$ can be characterized as the class of parameterized problems that fp-reduce to parameterized CLIQUE. Similarly, $W[2]$ can be characterized as the class of parameterized problems that fp-reduce to the parameterized Hitting Set problem, i.e., to the problem of checking whether a collection $\mathcal{C}$ of subsets of a finite set S has a hitting set of size at most k, where the parameter is the positive integer k. (We recall that a hitting set for $\mathcal{C}$ is a subset of S that contains at least one element from each subset in $\mathcal{C}$.)

A k-truth value assignment for a formula E is a truth value assignment which assigns true to *exactly k* propositional variables of E. Consider the following problem

Parameterized SAT:

Instance: A Boolean formula E.
Parameter: k.
Question: Does there exist a k-truth value assignment satisfying E?

$W[SAT]$ is the class of parameterized problems that fp-reduce to Parameterized SAT. $W[SAT]$ is contained in $W[P]$, where Boolean circuits are used instead of formulae. It is

[1] Typically, the functions f and f' used in fp-reductions are exponential in the parameter of the problem. Usually, and in particular in the proofs in this paper, they are simple functions of the form $k_1^{c_1 k_2 + c_2}$, where k_2 is the parameter, c_1, c_2 are fixed constants, and k_1 is either a fixed constant or the parameter.

not known whether any of the above inclusionships is proper or not, however the difference of all classes is conjectured.

The *AW-hierarchy* has been defined in order to deal with some problems that do not fit the *W*-classes [14]. The *AW*-hierarchy represents in a sense the parameterized counterpart of PSPACE in the classical complexity setting. In this paper we are mainly interested in the class $AW[SAT]$. Consider the following problem

Parameterized QBFSAT:

Instance: A quantified boolean formula $\Phi = Q_1^{k_1} \boldsymbol{x}_1 Q_2^{k_2} \boldsymbol{x}_2 \cdots Q_n^{k_n} \boldsymbol{x}_n E$.

Parameter: $k = \langle k_1, k_2, \ldots, k_n \rangle$.

Question: Is Φ valid? (Here, E is a Boolean formula, and each quantifier $Q_i^{k_i}$, $1 \leqslant i \leqslant n$, may be either $\exists^{k_i} \boldsymbol{x}$, or $\forall^{k_i} \boldsymbol{x}$. The former quantifier denotes the choice of some k_i-truth value assignment for the variables $\boldsymbol{x}$, whilst the latter denotes all choices of k_i-truth value assignments for the variables $\boldsymbol{x}$.)

$AW[SAT]$ is the class of parameterized problems that fp-reduce to Parameterized QBFSAT.

3. Constraint satisfaction problems, bounded treewidth, and fp-tractability

In this section we prove that constraint satisfaction problems of bounded treewidth over a fixed domain are fp-tractable. In order to obtain these results we need a number of definitions. In Section 3.1 we give a very general definition of CSPs; in Section 3.2 we define the treewidth of CSP problems and quote some recent results; in Section 3.3 we show the main tractability result.

3.1. Definition of CSPs

An instance of a *constraint satisfaction problem* (*CSP*) (also *constraint network*) is a triple $I = (Var, U, \mathcal{C})$, where *Var* is a finite set of variables, U is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_i is a pair (S_i, r_i), where S_i is a list of variables of length m_i called the *constraint scope*, and r_i is an m_i-ary relation over U, called the *constraint relation*. The tuples of r_i indicate the allowed combinations of simultaneous values for the variables S_i. A *solution* to a CSP instance is a substitution $\theta : Var \to U$, such that for each $1 \leqslant i \leqslant q$, $S_i\theta \in r_i$. The problem of deciding whether a CSP instance has any solution is called *constraint satisfiability* (*CS*). (This definition is taken almost verbatim from [28].)

To any CSP instance $I = (Var, U, \mathcal{C})$, we associate a hypergraph $\mathcal{H}(I) = (V, H)$, where $V = Var$, and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, where $var(S)$ denotes the set of variables in the scope S of the constraint C.

Let $\mathcal{H}(I) = (V, H)$ be the constraint hypergraph of a CSP instance I. The *primal graph* of I is a graph $G = (V, E)$, having the same set of variables (vertices) as $\mathcal{H}(I)$ and an edge connecting any pair of variables $X, Y \in V$ such that $\{X, Y\} \subseteq h$ for some $h \in H$.

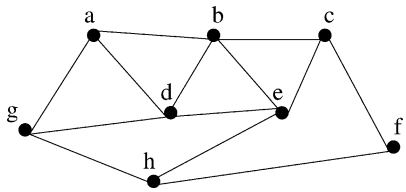

Fig. 1. The primal graph $G(I_e)$ of the CSP instance I_e in Example 3.

Note that if all constraints of a CSP are binary, then its associated hypergraph is identical to its primal graph.

Constraint Satisfaction is easily seen to be NP-complete. However, there are classes of CSP instances such that either the constraint relations or the structure of the constraint scopes have some properties that make these instances tractable. The most basic and most fundamental structural property considered in the context of CSPs (and conjunctive database queries) is *acyclicity*. It was recognized independently in AI and in database theory that *acyclic* CSPs (respectively, conjunctive queries) are solvable in polynomial time. There are many equivalent characterizations of acyclity [2]. We will use a characterization in terms of join trees. A *join tree* $JT(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose vertices are the edges of $\mathcal{H}$ such that, whenever the same variable $X \in V$ occurs in two edges A_1 and A_2 of $\mathcal{H}$, then A_1 and A_2 are connected in $JT(\mathcal{H})$, and X occurs in each vertex on the unique path linking A_1 and A_2 in $JT(\mathcal{H})$. In other words, the set of vertices in which X occurs induces a (connected) subtree of $JT(\mathcal{H})$. A CSP I is acyclic if and only if its constraint hypergraph $\mathcal{H}(I)$ has a join tree [2,30].

Example 3. Consider a CSP I_e having the following constraint scopes:

$$S_1(a,b,d);\ S_2(b,c,e);\ S_3(b,d,e);\ S_4(a,d,g);$$
$$S_5(g,h);\ S_6(e,h);\ S_7(h,f);\ S_8(c,f)$$

The constraint hypergraph $\mathcal{H}(I_e)$ associated to I_e contains the following edges: $\{a,b,d\}$, $\{b,c,e\}$, $\{b,d,e\}$, $\{a,d,g\}$, $\{g,h\}$, $\{e,h\}$, $\{h,f\}$, and $\{c,f\}$. The hypergraph $\mathcal{H}(I_e)$ is cyclic. In fact, it can be verified that does not exist any join tree for $\mathcal{H}(I_e)$. Fig. 1 shows the primal graph $G(I_e)$ of I_e.

3.2. Treewidth of CSPs

The treewidth of a graph is a measure of the degree of cyclicity of a graph.

Definition 4 [38]. A *tree decomposition* of a graph $G = (V, F)$ is a pair $\langle T, \lambda \rangle$, where $T = (N, E)$ is a tree, and λ is a labeling function associating to each vertex $p \in N$ a set of vertices $\lambda(p) \subseteq V$, such that the following conditions are satisfied:

(1) for each vertex b of G, there is a $p \in N$ such that $b \in \lambda(p)$;
(2) for each edge $\{b,d\} \in F$, there is a $p \in N$ such that $\{b,d\} \subseteq \lambda(p)$;
(3) for each vertex b of G, the set $\{p \in N \mid b \in \lambda(p)\}$ induces a (connected) subtree of T.

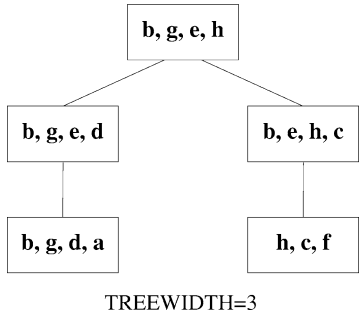

Fig. 2. A tree decomposition for the graph $G(I_e)$ in Example 3.

The *width* of the tree decomposition is the maximum cardinality over the labels of the vertices of T, i.e., $\max_{p\in N}\{|\lambda(p)|-1\}$. The *treewidth* of G is the minimum width over all its tree decompositions.

Example 5. Fig. 2 shows a tree decomposition of the graph $G(I_e)$ in Example 3. This decomposition has width 3. Moreover, it is easy to see that $G(I_e)$ has no tree decomposition of width 2, and thus its treewidth is 3.

Bodlaender [4] has shown that, for each fixed k, there is a linear time algorithm for checking whether a graph G has treewidth bounded by k and, if so, computing a tree decomposition of G having width k at most. Thus, the problem of computing a tree decomposition of a graph of width k is fp-tractable in the parameter k.

The treewidth of a CSP instance I is the treewidth of its primal graph $G(I)$. Accordingly, a tree decomposition of I is a tree decomposition of $G(I)$.

3.3. Fp-tractable CSPs

The parameterized version of Constraint Satisfaction, where the parameter is the total size of all constraint scopes, is $W[1]$-complete, and thus not fp-tractable. This follows from well-known results on conjunctive query evaluation [13,35], which is equivalent to constraint satisfaction (cf. [3,25,29]). Therefore, also bounded treewidth CSP is fp-intractable and $W[1]$-hard. Indeed, the CSPs having total size of the constraint scopes at most k form a subclass of the CSPs having treewidth at most $k-1$. Note that, for each fixed k, CSPs of treewidth at most $k-1$ can be evaluated in time $O(n^{k+1}\log n)$ [26].

In this section we show that, however, if as an additional parameter we fix the size of the domain U, then bounded treewidth *CSP* is fixed-parameter tractable.

It is worthwhile noting that the general CSP problem remains NP-complete even for constant domain U. (See, e.g., the 3-SAT problem discussed below, or the 3-coloring problem, where the universes are $\{0,1\}$ or the three colors, respectively.)

Theorem 6. *Constraint Satisfaction with parameters treewidth k and universe size $u=|U|$ is fp-tractable. So is the problem of computing a solution of a CSP problem with fixed parameters k and u.*

Proof. Let $I = (Var, U, \mathcal{C})$ be a CSP instance having treewidth k and $|U| = u$. We exhibit an fp-transformation from I to an equivalent instance $I' = (Var, U, \mathcal{C}')$. We assume without loss of generality that no constraint scope S in I contains multiple occurrences of variables. (In fact, such occurrences can be easily removed by a simple preprocessing of the input instance.) Note that, from the bound k on the treewidth, it follows that each constraint scope contains at most $k+1$ variables, and thus the constraint relations have arity at most $k+1$. (Indeed, any constraint scope of arity $a+1$ gives rise to a clique of size $a+1$ in the primal graph, which implies that the primal graph has treewidth at least a.)

Let $\langle T = (V, E), \lambda \rangle$ be a k-width tree decomposition of $G(I)$ such that $|V| \leqslant c|G(I)|$, for a fixed predetermined constant c. Note that this is always possible, because Bodlaender's algorithm [4] runs in linear time. Thus, also the size of any tree decomposition of $G(I)$ computed using this algorithm is $O(|G(I)|)$. For each vertex $p \in V$, the new instance I' has a constraint $C_p = (S', r') \in \mathcal{C}'$, where the scope S' is a list containing the variables belonging to $\lambda(p)$, and r' is the associated relation, computed as described below.

The relations associated to the constraints of I' are computed through the following two steps:

(1) For each constraint $C' = (S', r') \in \mathcal{C}'$, initialize r' as $U^{|var(S')|}$, i.e., the $|var(S')|$-fold cartesian product of the domain U with itself.
(2) For each constraint $C = (S, r) \in \mathcal{C}$, let $C' = (S', r') \in \mathcal{C}'$ be any constraint of I' such that $var(S) \subseteq var(S')$. Note that such a constraint must exist. Indeed, the variables in the list S form a clique in $G(I)$, and hence they should occur together in the label $\lambda(p)$ of some vertex p of the tree decomposition. Modify r' as follows. $r' := \{t' \in r' \mid \exists$ a substitution θ s.t. $S'\theta = t'$ and $S\theta \in r\}$. (In database terms, r' is semijoin-reduced by r.)

It is not hard to see that the instance I' is equivalent to I, in that they have exactly the same set of solutions. Indeed, any solution of I is a solution of I', because the new constraint relations added at step (1) above contain all the possible tuples for the variables in their constraint scopes, and thus no combination of values for these variables is forbidden by the new constraints belonging to I'. Moreover, any solution of I' is a solution of I, because every constraint of I must be satisfied, after step (2) above.

Note that the size of I' is at most $|U|^{k+1}(c|G(I)|)$, and even computing I' from I is feasible in linear time. Thus, this reduction is actually an fp-reduction.

The resulting instance I' is an acyclic constraint satisfaction problem. Indeed, from the construction above and from property (3) of Definition 4, it follows that T corresponds to a join tree of the hypergraph associated to I'. Thus, I' is equivalent to an acyclic conjunctive query over a fixed database [25]. Checking whether such a query has a nonempty result and, in the positive case, computing a single tuple of the result, is feasible in linear time by Yannakakis' well-known algorithm [41]. □

There are many methods for solving constraint satisfaction problems whose cost depends on some parameter of the constraint hypergraph. These methods are called structural CSP decomposition methods, because they are based on suitable decompositions

that transform cyclic CSP instances into acyclic CSP instances. (See [7,26,36] for descriptions and comparisons among these methods.) In particular, the tree-clustering method [8] is based on the same kind of construction we used in the proof above. The primal graph of a cyclic instance is first triangulated and then the maximal cliques of the resulting graph are used for building an acyclic hypergraph. By exploiting this hypergraph, one can build an acyclic CSP instance that is equivalent to the original (cyclic) instance. From the proof of Theorem 6, it follows easily that Constraint Satisfaction having as joint parameters the tree-clustering width and the universe size is fp-tractable. Similarly, it can be shown that also Constraint Satisfaction having the biconnected width [22,26] and the universe size as joint parameters, and Constraint Satisfaction having the cutset width [7,26] and the universe size as joint parameters, are all fp-tractable problems. Indeed, the construction used for the proof of Theorem 6 can be adapted to these problems, too. Moreover, the biconnected decomposition of a CSP instance can be computed in polynomial time, and finding a cutset (which is a synonym of feedback vertex set) of a CSP instance, with the cutset width as the parameter, is fp-tractable [14].

Note that, since solving a CSP is equivalent to conjunctive query evaluation (CQ) [3,25, 29], we can obtain a similar result on the complexity of evaluating conjunctive queries over a fixed size database. This is a generalization of the program complexity of conjunctive queries, i.e., the complexity of evaluating conjunctive queries over a fixed database [40].

Let Q be the conjunctive query

$$ans(\boldsymbol{u}) \leftarrow r_1(\boldsymbol{u}_1) \wedge \cdots \wedge r_n(\boldsymbol{u}_n),$$

where $n \geqslant 0$; $r_1, \ldots, r_n$ are relation names (not necessarily distinct); *ans* is a relation name ($ans \neq r_i$, $\forall 1 \leqslant i \leqslant n$); and $\boldsymbol{u}, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ are lists of terms (i.e., variables or constants) of appropriate length. The set of variables occurring in Q is denoted by $var(Q)$. The *answer* of Q on a database instance $\boldsymbol{db}$ with associated universe U, consists of a relation *ans* whose arity is equal to the length of $\boldsymbol{u}$, defined as follows. Relation *ans* contains all tuples $\boldsymbol{u}\theta$ such that $\theta : var(Q) \to U$ is a substitution replacing each variable in $var(Q)$ by a value of U and such that for $1 \leqslant i \leqslant n$, $r_i(\boldsymbol{u}_i)\theta \in \boldsymbol{db}$. (For any atom $r(\boldsymbol{u})$, $r(\boldsymbol{u})\theta$ denotes the atom obtained from $r(\boldsymbol{u})$ by uniformly substituting $\theta(X)$ for each variable X occurring in $\boldsymbol{u}$.) The conjunctive query Q is a *Boolean conjunctive query* if its head predicate *ans* has arity 0, thus the head is a purely propositional atom and does not contain variables. A Boolean CQ Q evaluates to *true* if there exists a substitution θ such that, for $1 \leqslant i \leqslant n$, $r_i(\boldsymbol{u}_i)\theta \in \boldsymbol{db}$; otherwise, the query evaluates to *false*. The size of the Conjunctive Query problem instance $\langle Q, \boldsymbol{db}\rangle$ is defined in the standard way, i.e., as the number of bits needed for encoding the query Q plus the number of bits needed for encoding the database $\boldsymbol{db}$ listing, for each relation $r \in \boldsymbol{db}$ occurring in Q, all the tuples belonging to r. The output size of a nonboolean CQ instance is the size of the output relation *ans*, while the output size of a Boolean CQ instance is one. Note that the output size of a conjunctive query may be exponentially greater than its input size, because the answer relation may contain exponentially many tuples.

The hypergraph $\mathcal{H}(Q) = (V, H)$ associated to the conjunctive query Q has $V = Var(Q)$ and $H = \{var(r_i(\boldsymbol{u}_i)) \mid 1 \leqslant i \leqslant n\}$, where $var(r_i(\boldsymbol{u}_i))$ denotes the set of variables occurring in $r_i(\boldsymbol{u}_i)$. The treewidth of Q is the treewidth of the primal graph of $\mathcal{H}(Q)$.

The following result complements some recent results on fixed-parameter tractability of database problems by Papadimitriou and Yannakakis [35].

Corollary 7. *The evaluation of Boolean conjunctive queries is fp-tractable if the parameters are jointly the treewidth of the query and the size of the database universe. Moreover, evaluating a nonboolean conjunctive query is fp-tractable in the input and output size with respect to the treewidth of the query and the size of the database universe.*

Proof. Let $\langle Q, \boldsymbol{db}\rangle$ be a CQ instance. As shown in [25], $\langle Q, \boldsymbol{db}\rangle$ is equivalent to a CSP instance $I = (Var, U, \mathcal{C})$ such that Q and I have the same set of variables, the same universe, and the same associated hypergraph, i.e., $\mathcal{H}(Q) = \mathcal{H}(I)$. It follows that Q and I have the same treewidth, too. Moreover, there is a one-to-one correspondence between the constraints in I and the atoms in Q, but the special output atom *ans*.

Exploiting this equivalence between CSP and CQ instances, the construction described in the proof of Theorem 6 can be modified in a straightforward way for obtaining in linear time an acyclic conjunctive query instance $\langle Q', \boldsymbol{db}'\rangle$ such that the answer of Q' on $\boldsymbol{db}'$ is the same as the answer of Q on $\boldsymbol{db}$.

Then, the corollary follows from the fact that computing the answer of an acyclic conjunctive query is feasible in time polynomial in the input and output size [41]. □

Let us conclude this section by mentioning two very interesting recent papers [20,27] that report independent work. These papers study the connection between parameterized complexity and logic (specifically, model checking) and contain some further results on the fp-tractability of constraint satisfaction problems.

4. Fp-tractable satisfiability problems

4.1. Bounded-width CNF formulae

As an application of our general result on fp-tractable CSPs we show that a relevant satisfiability problem is also fp-tractable.

The graph $G(F)$ of a CNF formula F has as vertices the set of propositional variables occurring in F and has an edge $\{x, y\}$ iff the propositional variables x and y occur together in a clause of F. The treewidth of F is defined to be the treewidth of the associated graph $G(F)$.

Example 8. Consider the following formula:

$$F_1 = (d \vee a \vee \neg g) \wedge (b \vee d \vee \neg a) \wedge (g \vee h) \wedge (\neg e \vee h) \wedge (b \vee c \vee e) \wedge (\neg b \vee d \vee \neg e) \wedge (c \vee f) \wedge (h \vee \neg f)$$

Fig. 1 shows the graph of the formula F_1. We already observed that the treewidth of this graph is 3, and thus the treewidth of F_1 is 3, too.

Theorem 9. *CNF Satisfiability with parameter treewidth k is fp-tractable. So is the problem of computing a model of a CNF formula with parameter k.*

Proof. We fp-transform a CNF formula F into a constraint satisfaction instance $I(F) = (Var, U, \mathcal{C})$ defined as follows. *Var* contains a variable X_p for each propositional variable p occurring in F; $U = \{0, 1\}$; and for each clause D of F, $I(F)$ contains a constraint (S, r) where the constraint scope S is the list containing all variables X_p such that p is a propositional variable occurring in D, and the constraint relation $r \subseteq U^{|D|}$ consists of all tuples corresponding to truth value assignments satisfying D.

For instance, consider the formula F_1 in Example 8. For the first clause $d \vee a \vee \neg g$, $I(F_1)$ contains the constraint (S_1, r_1), where $S_1 = (d, a, g)$ and r contains all tuples corresponding to truth value assignment for (d, a, g) satisfying D_1, such as $\langle 0, 0, 0\rangle$, $\langle 1, 0, 1\rangle$, and so on.

It is obvious that every model of F correspond to a solution of $I(F)$ and vice versa. Thus, in particular, F is satisfiable if and only if $I(F)$ is a positive CSP instance.

Since $G(F)$ is isomorphic to $G(I(F))$, both F and $I(F)$ have the same treewidth. Moreover, any CNF formula F of treewidth k has clauses of cardinality at most $k + 1$. Therefore, our reduction is feasible in time $O(2^{k+1}|F|)$ and is thus an fp-reduction with respect to parameter k.

By this fp-reduction, fp-tractability of CNF-SAT with the treewidth parameter follows from the fp-tractability of CSPs with respect to treewidth, as stated in Theorem 6. □

4.2. CNF with short prime implicants

The problem of finding the prime implicants of a CNF formula is relevant to a large number of different areas, e.g., diagnosis [37], truth maintenance systems [9], knowledge compilation [5], and many other AI applications.

Clearly, the set of the prime implicants of a CNF formula F can be viewed as a compact representation of the satisfying truth assignments for F. It is worthwhile noting that the restriction of *Parameterized SAT* to CNF formulae is fp-intractable. More precisely, deciding whether a q-CNF formula F has a k-truth value assignment is $W[2]$-complete [14]. (We recall that a k-truth value assignment assigns true to exactly k propositional variables.)

Nevertheless, we identified a very natural parameterized version of the prime implicant problem which is fp-tractable. We simply take as the parameter the length of the prime implicants of the Boolean formula.

Given a q-CNF formula F, the *Short Prime Implicants problem* (*SPI*) is the problem of computing the (consistent) prime implicants of F having length $\leqslant k$, with parameters k and q.

Theorem 10. *SPI is fixed-parameter tractable.*

Proof. Let F be a q-CNF formula. Without loss of generality, assume that F does not contain tautological clauses. We generate a set $IM_k(F)$ of implicants of F from which it is possible to compute the set of all prime implicants of F having length $\leqslant k$. (This is very similar to the well-known procedure of generating vertex covers of bounded size, cf. [10,14].) Pick an arbitrary clause C of F. Clearly, each implicant I of F must contain at least one literal of C. We construct a labeled tree t each of whose vertices V is labeled

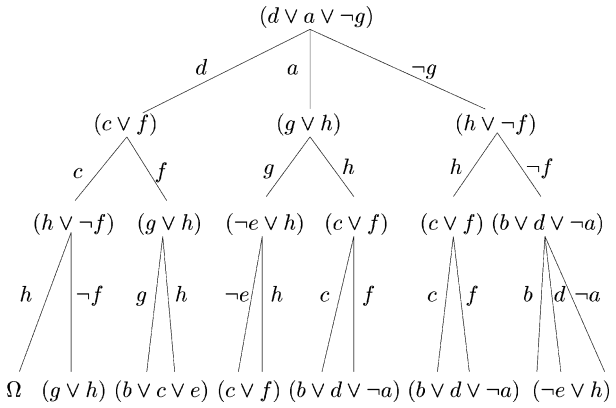

Fig. 3. The tree for computing the short prime implicants of F_1.

by a clause in F or by a special symbol Ω, and each of whose edges is labeled by a literal occurring in F. The root of t is C. Each vertex $D \in V$, which is not labeled by Ω and has level (strictly) less than k, has an edge labeled ℓ to a descendant G_ℓ, for each literal $\ell \in D$. If there are some clauses of F that do not intersect the set of all edge-labels from the root to the current position, then the vertex G_ℓ is labeled by one of these clauses; otherwise, G_ℓ is labeled by the special symbol Ω. Vertices labeled by Ω are always leaves of t. Note that, by construction, the maximum length of every branch in t is k.

As a running example, consider again the formula F_1 in Example 8, and assume we are going to compute the prime implicants of F_1 having length $\leqslant 3$. Since F_1 is a 3-CNF formula, the parameters in this example are $q = 3$ and $k = 3$. Fig. 3 shows a tree computed for F_1 following the procedure above. For space reasons, sibling leaves sharing the same label L are depicted just through their label L.

For each root-leaf branch β of the tree, let $I(\beta)$ be the set containing the $\leqslant k$ literals labeling the edges of β. If the leaf of β is labeled by Ω and $I(\beta)$ is a consistent set of literals, then $I(\beta)$ is an implicant of F and has no more than k literals. Indeed, by construction, there is no clause of F that does not intersect $I(\beta)$, and the length of any root-leaf branch in T is bounded by k. In this case, we add $I(\beta)$ to the set $IM_k(F)$. In our running example, let β_1 be the leftmost branch in the tree shown in Fig. 3. Since the set $I(\beta_1) = \{d, c, h\}$ is a consistent set of literals, it is an implicant of F_1, and will be included in the set $IM_3(F_1)$. Observe that, in this example, there are no further implicants of F_1 having length 3. For instance, consider the branch β_2 traversing the edge labeled by d, c, and $\neg f$. The set $I(\beta_2) = \{d, c, \neg f\}$ is consistent, but it is not an implicant of F_1 and hence will not be included in $IM_3(F_1)$. Indeed, at least the clause $(g \vee h)$ labeling the leaf of β_2 is not satisfied by $I(\beta_2)$.

It is easy to see that the size of the tree t is bounded by q^{k+1} and that, for every prime implicant S of F having length $\leqslant k$, $S \subseteq I$ holds, for some implicant $I \in IM_k(F)$.

Moreover, note that there are at most q^k implicants in $IM_k(F)$, as the maximum number of leaves in t is q^k. For each implicant $I \in IM_k(F)$, the set of all consistent prime implicants of F included in I can be easily obtained in time $O(2^k|F|)$ from I. It follows that SPI is fp-tractable with respect to parameters q and k. □

5. Logic programs with negation

Logic programming with negation under the stable model semantics [24] is a well-studied form of nonmonotonic reasoning [31]. Moreover, there exist successful systems implementing this form of reasoning, e.g., *Smodels* [33,34] and *dlv* [16,19].

A *literal* L is either an atom A (called *positive*) or a negated atom $\neg A$ (called *negative*). Literals A and $\neg A$ are *complementary*; for any literal L, we denote by $\neg.L$ its complementary literal, and for any set Lit of literals, $\neg.Lit = \{\neg.L \mid L \in Lit\}$.

A *normal clause* r is a rule of the form

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \quad (n \geqslant m \geqslant 0), \tag{1}$$

where A is an atom called the head of r and denoted by $H(r)$, and the conjunction of literals is called the body of r. We denote the set of the literals occurring in the body of r by $B(r)$. Moreover, $B^+(r) = \{A_1, \ldots, A_m\}$ denotes the set of atoms occurring in positive literals in the body of r, and $B^-(r) = \{A_{m+1}, \ldots, A_n\}$ the set of atoms occurring in negative literals in the body of r.

A *normal logic program* is a finite set of normal clauses. The Herbrand base B_P of a logic program P is the set of those atoms A such that either A or $\neg A$ occurs in P. Any subset I of B_P (not necessarily proper) is an interpretation for P; the atoms in I are interpreted true, and the atoms in $B_P - I$ are interpreted false.

A normal logic program P is *stratified* [1], if there is an assignment $str(\cdot)$ of integers $0, 1, \ldots$ to the predicates p in P, such that for each clause r in P the following holds: If p is the predicate in the head of r and q the predicate in an L_i from the body, then $str(p) \geqslant str(q)$ if L_i is positive, and $str(p) > str(q)$ if L_i is negative.

The *reduct* of a normal logic program P by a Herbrand interpretation I [24], denoted P^I, is obtained from P as follows: first remove every clause r with a negative literal $\neg p$ in the body such that $p \in I$, and then remove all negative literals from the remaining rules.

An interpretation I of a normal logic program P is a *stable model* of P [24], if I is the least Herbrand model of P^I. In general, a normal logic program P may have zero, one, or multiple (even exponentially many) stable models. (See [6], for exact estimates on the largest possible number of stable models of logic programs.) Denote by $stabmods(P)$ the set of stable models of P.

It is well known that every stratified logic program has a unique stable model which can be computed in linear time.

The following problems are the main decision and search problems in the context of logic programming.

Main logic programming problems. Let P be a logic program.

(1) *Consistency*: Determine whether P admits a stable model.
(2) *Brave reasoning*: Check whether a given literal is true in a stable model of P.
(3) *Cautious reasoning*: Check whether a literal is true in every stable model of P.
(4) *SM Computation*: Compute an arbitrary stable model of P.
(5) *SM Enumeration*: Compute the set of all stable models of P.

Example 11. Consider the following logic program P_1:

$$p \leftarrow q \wedge \neg r \quad r \leftarrow q \wedge s \quad s \leftarrow \neg p \wedge r \quad q \leftarrow \neg u \quad u \leftarrow \neg q$$

Let $I = \{u\}$. The interpretation I is a stable model for P_1. Indeed, the reduct P_1^I is the program

$$p \leftarrow q \quad r \leftarrow q \wedge s \quad s \leftarrow r \quad u \leftarrow$$

and the minimum model of this program is $\{u\}$. Thus, P_1 is a "yes" instance of the Consistency problem, because it has stable models. Moreover, u is a brave consequence for P_1, i.e., a consequence of P_1 according to brave reasoning.

It is easy to see that $\{p, q\}$ is another stable model for P_1, and that there are no further stable models for P_1. It follows that both $\neg r$ and $\neg s$ are cautious consequences of P_1, because these literals are true according to both of the stable models for P_1.

5.1. Breaking the cycles with negative edges

For a normal logic program P, the *dependency graph* $G(P)$ is an arc-labeled directed graph (N, A) where N is the set of atoms occurring in P and A is a set of arcs such that $(p, q) \in A$ if there exists a rule $r \in P$ having p in its head and q in its body. Moreover, if there is a rule having p in its head and with $\neg q$ in its body, then the edge (p, q) is labeled with the symbol $\neg$. The *undirected dependency graph* $G^*(P) = (V, E)$ of P is an undirected and unlabeled version of $G(P)$. The set of vertices V is given by $N \cup N'$, where N' is a set of new vertices, called *negative vertices* corresponding one-to-one to the labeled arcs in $G(P)$. The set of edges E is defined as follows:

- for each unlabeled arc $(p, q) \in A$ there is an edge $\{p, q\} \in E$;
- for each labeled arc $e = (p, q) \in A$ there are two edges $\{p, r_e\}$ and $\{r_e, q\}$ in E, where $r_e \in N'$ is the vertex corresponding to the labeled arc e and is not connected to any other vertex in V (different from p and q).

A *feedback vertex set* S of an undirected (directed) graph G is a subset X of the vertices of G such that any cycle (directed cycle) contains at least one vertex in S. Clearly, if a feedback vertex set is removed from G, then the resulting graph is acyclic. The *feedback width* of G is the minimum size over its feedback vertex sets.

Example 12. Consider again the program P_1 in Example 11. Fig. 4(a) shows the dependency graph of P_1, and Fig. 4(b) shows the undirected and unlabeled graph $G^*(P_1)$. The circled vertices in Fig. 4(b) denote a feedback vertex set for $G^*(P_1)$. Indeed, dropping the vertices q and s, we get an acyclic graph. Moreover, it is easy to verify that $G^*(P_1)$ has no feedback vertex set of cardinality 1, and hence its feedback width is 2.

It was shown by Downey and Fellows [10,14] that determining whether an undirected graph has feedback width k and, in the positive case, finding a feedback vertex set of size k, is fp-tractable with respect to the parameter k.

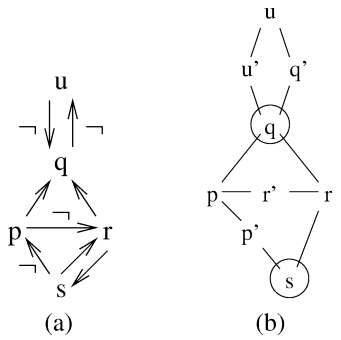

Fig. 4. (a) The dependency graph $G(P_1)$, and (b) the graph $G^*(P_1)$ with a feedback vertex set for it.

Let P be a logic program defined over a set U of propositional atoms. A *partial truth value assignment* (*p.t.a.*) for P is a truth value assignment to a subset U' of U. We say that a positive literal p (respectively, a negative literal $\neg p$) contradicts τ if $\tau(p) = \mathit{false}$ (respectively, $\tau(p) = \mathit{true}$). If τ is a p.t.a. for P, denote by $P[\tau]$ the program obtained from P as follows:

- eliminate all rules whose body contains a literal contradicting τ;
- eliminate from every rule body all literals made true by τ, i.e., every positive literal p such that $\tau(p) = \mathit{true}$ and every negative literal $\neg p'$ such that $\tau(p') = \mathit{false}$.

An interpretation M is said *consistent* with a p.t.a. τ if both $M \cap \{p \mid \tau(p) = \mathit{false}\} = \emptyset$ and $M \supseteq \{p \mid \tau(p) = \mathit{true}\}$ hold.

Lemma 13. *Let M be a stable model of some logic program P, and let τ be a p.t.a. consistent with M. Then, M is a stable model of $P[\tau]$.*

Proof. Assume by contradiction that M is not a stable model of $P[\tau]$. Then, M is not the minimum model of the reduct $P[\tau]^M$ and hence there exists a model $M' \subset M$ for $P[\tau]^M$. However, M is a stable model for P and thus M' cannot be a model of P^M. It follows that there exists a rule $r \in P^M$ such that $H(r) \notin M'$ and $B(r) \subseteq M'$. This rule comes from some rule r' of P such that $H(r) = H(r')$, $B(r) = B^+(r')$, and $B^-(r') \cap M = \emptyset$. Note that $B^+(r') \subseteq M'$ and $M' \subset M$ entail that $B^+(r') \subset M$. Since τ is consistent with M, it follows that there is no $p \in B^+(r')$ such that $\tau(p) = \mathit{false}$. Let $D = \{p \in B^+(r') \mid \tau(p) = \mathit{true}\}$, and recall that all atoms in D will be deleted from r in the simplified program $P[\tau]$. Moreover, since τ is consistent with M and $B^-(r') \cap M = \emptyset$, there is no atom $p \in B^-(r')$ such that $\tau(p) = \mathit{true}$. Thus, the rule $H(r') \leftarrow B(r') - D$ belongs to $P[\tau]$, and the rule $H(r') \leftarrow B^+(r') - D$ belongs to $P[\tau]^M$. However, this is a contradiction, because the latter rule is not satisfied according to M' and we assumed M' be a model for $P[\tau]^M$. □

Let P be a logic program. Given a set of vertices S of the graph $G^*(P)$, let $\mathit{Atoms}(S)$ be any set of vertices (corresponding to atoms occurring in P) obtained from S replacing each negative vertex $q \in S$ by one of its adjacent vertices in $G^*(P)$. Recall that, by definition of

$G^*(P)$, such a negative vertex q has only two neighbours, which are not negative vertices and correspond to the endpoints of the labeled arc of $G(P)$ associated to q. Therefore, it is easy to verify that, if S is a feedback vertex set for $G^*(P)$, then also $Atoms(S)$ is a feedback vertex set for $G^*(P)$. Indeed, deleting a negative vertex from $G^*(P)$ cannot break more cycles than deleting either of its neighbours. Moreover, $|Atoms(S)| \leqslant |S|$ clearly holds. It follows that, if $G^*(P)$ has feedback vertex width $w \leqslant k$, for a fixed k, then there is a feedback vertex set S for $G^*(P)$ not containing negative vertices and having cardinality w. Furthermore, since a bounded-width feedback vertex set of an undirected graph is computable in linear time (see [14]), such a feedback vertex set without negative vertices can be clearly computed in linear time, as well.

Theorem 14. *The logic programming problems* (1)–(5) *listed above are all fp-tractable with respect to the feedback width of the undirected dependency graph of the logic program.*

Proof. Let P be a logic program whose graph $G^*(P) = (V, E)$ has feedback width k, where the set of vertices V is partitioned into the two sets $N = B_P$, which contains all the atoms occurring in P, and N', which contains the negative vertices corresponding to the negated arcs in the dependency graph $G(P)$—see the definition of these graphs above.

We compute in linear time a feedback vertex set S for $G^*(P)$ without negative vertices (see discussion above) s.t. $|S| = k$. Then, we consider the set T of all the 2^k partial truth value assignments to the atoms in S.

For each p.t.a. $\tau \in T$, $P[\tau]$ is a stratified program whose unique stable model M_τ can be computed in linear time. For each $\tau \in T$, compute M_τ and check whether $M_\tau \in stabmods(P)$, where $stabmods(P)$ denotes the set of all stable models of P (this latter can be done in linear time, too, if suitable data structures are used).

Let $\Sigma = \{M_\tau \mid M_\tau \in stabmods(P)\}$. By definition of Σ, it suffices to note that every stable model M for P belongs to Σ. Indeed, let τ be the p.t.a. on S determined by M. By Lemma 13, it follows that M is a stable model of $P[\tau]$ and hence $M \in \Sigma$.

Thus, P has at most 2^k stable models whose computation is fp-tractable and actually feasible in linear time. Therefore, the problem (5) above (Stable Model Enumeration) is fp-tractable. The fp-tractability of all other problems follows. □

Here is an example of a class of programs having feedback width 2.

Example 15. Consider the class of all logic programs P whose undirected dependency graph $G^*(P)$ is the book-shaped graph $book(n)$, for some $n > 0$. The graph $book(n)$ has $2n + 2$ vertices and $3n + 1$ edges that form n squares (pages of the book) having exactly one common edge $\{X, Y\}$ (see also [26]). Fig. 5 shows the graph $book(4)$.

No matter on how large n is and how many negative arcs $G(P)$ contains, the feedback width of P is two, because just deleting from $G^*(P)$ the vertices corresponding to X and Y we get an acyclic graph. Moreover, observe that the feedback width remains 2 even if one replaces the pages of the book by cycles of any length.

It appears that an overwhelmingly large number of "natural" logic programs have very low feedback width, thus the technique presented here seems to be very useful in practice.

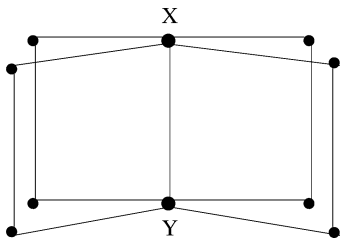

Fig. 5. The graph *book*(4).

Note, however, that the technique does not apply to some important and rather obvious cases. In fact, the method does not take care of the direction and the labeling of the arcs in the dependency graph $G(P)$. Hence, positive programs width large feedback width are not recognized to be tractable, although they are clearly tractable. The same applies, for instance, for stratified programs having large feedback width, or to programs whose high feedback-width is exclusively due to positive cycles.

Unfortunately, it is not known whether computing feedback vertex sets of size k is fixed-parameter tractable for *directed graphs* [14].

Another observation leading to a possible improvement is the following. Call an atom p of a logic program P *malignant* if it lies on at least one simple cycle of $G(P)$ containing a marked (= negated) edge. Call an atom *benign* if it is not malignant. It is easy to see that only malignant atoms can be responsible for a large number of stable models. In particular, every stratified program contains only benign atoms and has exactly one stable model. This suggest the following improved procedure:

- Compute the set of benign atoms occurring in P;
- Drop these benign vertices from $G^*(P)$, yielding a new graph $H(P)$;
- Compute a feedback vertex set S of size $\leqslant k$ of $H(P)$;
- For each p.t.a. τ over S compute the unique stable model M_τ of $P[\tau]$ and check whether this is actually a stable model of P, and if so, output M_τ.

It is easy to see that the above procedure correctly computes the stable models of P. Unfortunately, as shown by the next theorem, it is unlikely that this procedure can run in polynomial time.

Theorem 16. *Determining whether an atom of a propositional logic program is benign is* co-NP-*complete.*

Proof. Membership in co-NP is obvious. For the hardness part, we reduce the node-disjoint path problem for directed graphs, which is an NP-complete problem [23], to the problem of deciding whether an atom q occurring in a logic program P is malign, which is clearly the complement of our problem.

Let $G = (N, A)$ be a directed graph, and $\langle x_1, y_1\rangle$ and $\langle x_2, y_2\rangle$ two pairs of nodes of G. The problem is deciding whether there are two node-disjoint paths linking x_1 to x_2 and y_1 to y_2 [21]. From G, we construct a logic program P_G containing a rule $s \leftarrow t$ for each arc

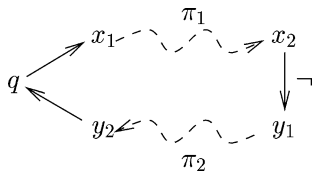

Fig. 6. Disjoint paths π_1 and π_2 in the proof of Theorem 16.

$(s,t) \in A$, plus three additional rules $q \leftarrow x_1$, $y_2 \leftarrow q$, and $x_2 \leftarrow \neg y_1$, where q is a fresh atom not corresponding to any node of G. We show that q is a malignant atom for P_G if and only if there are two node-disjoint paths linking x_1 to x_2 and y_1 to y_2 in G.

If part. Assume there are two node-disjoint paths π_1, linking x_1 to x_2, and π_2, linking y_1 to y_2. Fig. 6 shows these paths.

Then, there is a simple cycle which, starting from the arc (q, x_1), reaches x_2 via path π_1, traverses the labeled arc (x_2, y_1), and finally comes back to q via path π_2 and the edge (y_2, q). Therefore, q is a malignant atom for G_P.

Only if part. Assume q is a malignant atom for G_P. Then there is a simple cycle c that traverses the labeled arc (x_2, y_1). Indeed, this is the only labeled arc in G_P. By construction, q is just connected to x_1 and y_2 through the two arcs (q, x_1) and (y_2, q), respectively. Thus, the cycle c contains a path from x_1 to x_2 and one from y_1 to y_2. Moreover, since c is a simple cycle, there is no node of G_P occurring two times in c, and thus these paths are node disjoint. □

5.2. The weak feedback-width of a dependency graph

In this section, we propose another approach, which is somewhat weaker, but very efficient and easy to deal with. The weaker notion that we propose is based on the notion of benign and malignant atoms, but it is defined on the undirected graph $G^*(P)$ of a logic program P. An atom p occurring in P is called *weakly malignant* if it lies on at least one simple cycle of $G^*(P)$ containing a negative vertex, and hence corresponding to a labeled arc of the dependency graph $G(P)$. An atom is called *strongly benign* if it is not weakly-malignant.

These atoms can be characterized usefully in terms of biconnected components. Recall that a vertex-induced connected subgraph G' of an undirected graph G is a *biconnected component* of G if it is a biconnected graph, i.e., it cannot be disconnected by any one-vertex removal.

Lemma 17. *An atom of a logic program P is weakly malignant if and only if it belongs to a biconnected component of $G^*(P)$ containing some negative vertex.*

Proof. Recall that an (undirected) graph G is biconnected if and only if, for each pair p, q of vertices of G, p and q are biconnected, i.e., p remains reachable from q after any one-vertex removal from G. By Menger's theorem [32], p and q are biconnected if and only if there are two vertex-disjoint paths from p to q in G. This is clearly equivalent to say that

p and q are biconnected in G if and only if there is a simple cycle in G containing both p and q.

It follows that, if a biconnected component of $G^*(P)$ contains a negative vertex q, then any other vertex p in this component is biconnected to q. Hence, p lies on a simple cycle containing q and is thus weakly malignant.

For the other direction of this lemma, assume p is a weakly malignant vertex in $G^*(P)$. Then, there is a negative vertex q in P such that there exists a simple cycle in $G^*(P)$ containing both p and q. Thus, p and q are biconnected in $G^*(P)$, and hence there is a biconnected component containing both of them. □

Lemma 18. *Computing the set of strongly benign* (*weakly malignant*) *atoms of a logic program can be done in linear time.*

Proof. Follows immediately from Lemma 17, and from the fact that the biconnected components of a graph can be computed in linear time [18]. □

We next present an improved algorithm for enumerating the stable models of a logic program P based on the feedback width of a suitable undirected graph associated to P. For a strongly connected component C of the dependency graph $G(P)$, we define a program P/C corresponding to C as follows. For each rule $r \in P$ such that $H(r) \in C$, the program P/C contains a rule r' where:

- the head $H(r')$ coincides with the head $H(r)$ of r; and
- the body of r' contains all the literals of $B(r)$ whose atoms belong to C, i.e., $B(r') = \{p \mid p \in (B^+(r) \cap C)\} \cup \{\neg q \mid q \in (B^-(r) \cap C)\}$.

Modular Stable Model Enumeration procedure (MSME).

(1) Compute the set $\mathcal{C}$ of the strongly connected components of $G(P)$;
(2) Determine the set $\mathcal{UC} \subseteq \mathcal{C}$ of the strongly connected components of $G(P)$ whose corresponding program P/C is not stratified;
(3) For each strongly connected component $C \in \mathcal{UC}$, compute the set of strongly benign atoms $SB(C)$ occurring in P/C;
(4) Let $P' = \bigcup_{C \in \mathcal{UC}} P/C$;
(5) Let $H(P')$ be the subgraph of $G^*(P')$ obtained by dropping every vertex p occurring in some set of strongly benign atoms $SB(C)$, for any $C \in \mathcal{UC}$;
(6) Compute a feedback vertex set S of $H(P')$ having size at most k and not containing negative vertices;
(7) For each p.t.a. τ over S compute the unique stable model M_τ of $P[\tau]$, check whether this is actually a stable model of P, and if so, output M_τ.

The feedback width of the graph $H(P')$ is called the *weak feedback-width* of the dependency graph of P.

Example 19. Let P_2 be the following logic program:

$$
\begin{array}{llll}
a \leftarrow c & b \leftarrow a & b \leftarrow e \wedge \neg d & c \leftarrow b \\
c \leftarrow h \wedge d & d \leftarrow e & e \leftarrow f \wedge i & f \leftarrow \neg d \wedge h \\
g \leftarrow f \wedge \ell & h \leftarrow g & \ell \leftarrow \neg i & i \leftarrow \neg \ell
\end{array}
$$

Fig. 7 shows the dependency graph of P_2. The strongly connected components of $G(P_2)$ are $C_1 = \{a, b, c\}$, $C_2 = \{d, e, f, g, h\}$, and $C_3 = \{\ell, i\}$. At step (2) of algorithm MSME we find the set $\mathcal{UC}$ of the strongly connected components of $G(P_2)$ whose corresponding subprograms of P_2 are stratified. It is easy to see that $P_2/C_1 = \{a \leftarrow c,\ b \leftarrow a,\ c \leftarrow b,\ b \leftarrow,\ c \leftarrow\}$ is stratified, while P_2/C_2 and P_2/C_3 are unstratified programs. Thus, $\mathcal{UC} = \{C_2, C_3\}$.

Now, we have to compute the set of strongly benign atoms belonging to the components in $\mathcal{UC}$. Fig. 8 shows the graph $G^*(P_2/C_2)$ that we use for the computation of $SB(C_2)$. This graph has two biconnected components: $\{e, f, d, d'\}$ and $\{f, g, h\}$. Since d' is a negative vertex (it corresponds to the labeled arc (f, d) in the dependency graph $G(P_2/C_2)$), the atoms e, f, and d are weakly malignant. It follows that the set of strongly benign atoms of C_2 is $SB(C_2) = \{g, h\}$. Similarly, it is easy to see that all the atoms in C_3 are weakly malignant, and hence $SB(C_3) = \emptyset$.

Fig. 9 shows the graph $H(P_2')$ obtained dropping all the strongly benign atoms of C_2 and C_3 from $P_2' = (P_2/C_2 \cup P_2/C_3)$. The feedback width of this graph, and hence the weak feedback-width of P_2, is 2. The set S containing the circled vertices d and ℓ in Fig. 9 is a feedback vertex set of $H(P_2')$ without negative vertices.

For any p.t.a. τ over S, we compute a stable model M for the stratified program $P_2[\tau]$, and check whether M is actually a stable model for P_2. For instance, from a p.t.a. τ_1 that

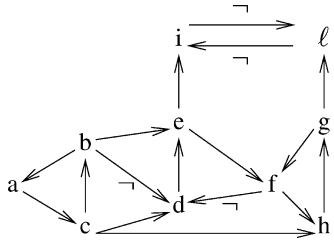

Fig. 7. Dependency graph $G(P_2)$ of the program P_2 in Example 19.

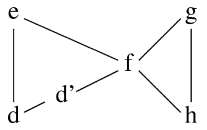

Fig. 8. The graph $G^*(P_2/C_2)$ in Example 19.

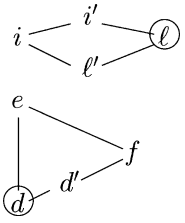

Fig. 9. The graph $H(P_2')$ with a feedback vertex set for it in Example 19.

assigns *false* to d and *true* to ℓ, we get a stable model $\{\ell\}$ for $P_2[\tau_1]$ that is also a stable model for P_2.

Theorem 20. *Given a logic program P, the MSME procedure computes all the stable models of P. Moreover, the logic programming problems* (1)–(5) *listed above are all fp-tractable with respect to the weak feedback-width of the dependency graph of the logic program.*

Proof. Let P be a logic program whose graph $H(P')$, constructed following the algorithm MSME above, has feedback width k. Let S be a feedback vertex set of $H(P')$ having width k and not containing negative vertices.

We claim that, for any p.t.a. τ over S, the program $P[\tau]$ is stratified. Assume by contradiction that this is not the case. Then, there is a cycle $t = p_1, p_2, \ldots, p_\ell, p_1$ (with $\ell > 1$) in the graph $G(P[\tau])$ such that (p_1, p_2) is an arc labeled by $\neg$. Clearly, $G(P[\tau])$ is a subgraph of $G(P)$, hence the cycle t occurs in $G(P)$, too. It follows that the labeled arc (p_1, p_2) occurs in a simple cycle, say t', in some strongly connected component C of $G(P)$. Let A be the set of atoms occurring in t'. Note that $A \subseteq \{p_1, p_2, \ldots, p_\ell\}$ and hence $A \cap S = \emptyset$, because t is a cycle in $P[\tau]$ and no atom in S occurs in this program. By definition of strongly benign atoms, the atoms occurring in t' are not strongly benign, i.e., $A \cap SB(C) = \emptyset$. Therefore, (the undirected and unlabeled version of) t' is a cycle in $H(P')$, and thus S is not a feedback vertex set of $H(P')$, a contradiction.

The theorem then easily follows from the fact that a feedback vertex set can be computed in linear time and from Lemma 13, as already discussed in the proof of Theorem 14. □

Note that, for the sake of simplicity, in this paper we considered just propositional logic programs, where no variable occurs in the literals. However, our algorithms can be extended straightforwardly to non-propositional logic programs. Indeed, for each (non-propositional) logic program P, there is a propositional logic program, called *ground*(P), which is equivalent to P, in that it has the same models and the same stable models as P. For each rule $r \in P$, *ground*(P) contains all the ground rules obtained by uniformly replacing the variables occurring in r by the constants occurring in P.

Moreover, note that the methods used in this section can be adapted to show fixed-parameter tractability results for extended versions of logic programming, such as disjunctive logic programming, and for other types of nonmonotonic reasoning. In the case of disjunctive logic programming, it is sufficient to extend the dependency graph to contain a labeled directed edge between every pair of atoms occurring together in a rule head.

5.3. Possible applications

We believe that the algorithms presented in this section, in particular, the MSME procedure, can be profitably used both to construct new, more efficient AI systems, and to improve current implementations such as existing systems for nonmonotonic logic programming, e.g., *Smodels* [33,34] and `dlv` [16,19]. In fact, both Smodels and `dlv` use powerful heuristics for finding stable models of logic programs very quickly. Roughly, they

first compute all the deterministic consequences of the given program, and then try to fix the truth values of some atoms that heuristics select as the most promising for leading to a stable model of the program. Such atoms are used as the choice points in backtracking-like algorithms that generate the stable models. Our results could be exploited in these systems in at least two ways.

Guarantee of fast answers for large classes of programs. If the weak feedback width of a program is small, i.e., below some fixed threshold, the MSME procedure provides an algorithm for computing the stable models of the program with a known polynomial upper bound (of fixed degree). This provides a *guarantee* that large classes of programs can be solved efficiently. We believe that guarantees of this type are important qualitative statements for assessing the efficiency of an implemented system. Such qualitative statements should complement quantitative arguments such as performance measurements on sample programs. For instance, both Smodels and `dlv` guarantee that stratified logic programs are evaluated in polynomial time. We believe that, due to its generality, the weak feedback width could be a well-suited parameter for assessing a system.

The currently available systems do not *explicitly* guarantee efficient behaviour on inputs of bounded weak feedback width. In case it turns out that a system requires exponential runtime for some input classes of bounded weak feedback width, we suggest to improve the system by adding a filter to deal with this parameter. For instance, one can first compute the deterministic consequences of the program and simplify the program (and hence its dependency graph) according to this information. Then, one can compute the weak feedback width of this program, and check whether it is smaller than some reasonable constant. In this case, it should be convenient to evaluate the program according to the MSME procedure; otherwise, the stable models can be computed through the usual strategy.

New heuristics. The heuristics for choosing the most promising atoms to be fixed may exploit the notion of weakly malignant atoms. Recall that the set of the weakly malignant atoms can be computed very efficiently. As we have shown, by fixing the truth values for these atoms, one gets a stratified program. Thus, the number of weakly malignant atoms provides a measure of the amount of non-determinism in the logic program. A heuristics strategy could prefer the atoms whose deletion from the dependency graph minimizes the number of weakly malignant atoms. Anyway, every strategy can also restrict its search space for choice-point candidates to weakly malignant atoms, as fixing just their values yields a linear time solvable program.

Of course, in practical systems many observations about typical structures of logic programs may help in optimizing and tuning these procedures. For instance, many logic programs for solving search NP problems in a declarative way are based on the "guess-and-check" paradigm [17]. That is, they contain an unstratified component (a kind of module) that "guesses" a solution of the problem at hand, and then other components (modules) that "check" whether the guessed solution is actually a feasible solution of the

problem. The guessing component is usually very complex, i.e., full of cycles involving negated atoms. However, the other components of the program that depends on it are usually much simpler and easy to deal with. Thus, one can of course apply the MSME procedure just to these "upper" components, once the "guessed" atoms have been fixed.

6. The small model circumscription problem

In this section we study the fixed-parameter complexity of a tractable parametric variant of circumscription, where the attention is restricted to models of small cardinality.

6.1. Definition of small model circumscription

The *Small Model Circumscription Problem* (*SMC*) is defined as follows. Given a propositional theory T, over a set of atoms $A = P \cup Z$, and given a propositional formula ϕ over vocabulary A, decide whether ϕ is satisfied in a model M of T such that:

- M is of small size, i.e., at most k propositional atoms are true in M (written $|M| \leqslant k$); and
- M is $P;Z$-minimal with respect to all other small models,[2] i.e., there is no model M' of T such that both $|M'| \leqslant k$ and $M' \cap P \subset M \cap P$ hold.

This problem appears to be a miniaturization of the classical problem of (brave) reasoning with minimal models. We believe that SMC is useful, since in many contexts, one has large theories, but is mainly interested in small models (e.g. in abductive diagnosis).

Example 21. Let $k = 2$, let T be the theory

$$(x_1 \vee x_2 \vee \neg x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_5),$$

and let $P = \{x_1, x_2\}$ and $Z = \{x_3, x_4, x_5\}$.

The propositional formula $\phi = \neg x_2 \wedge \neg x_5$ is a (brave) consequence of T according to small model circumscription. Indeed, the model $M = \{x_1, x_3\}$ is small, as $|M| = 2$, $M \models \phi$ holds, and it is $P;Z$-minimal with respect to all other small models for T. Indeed, any other model of T having a smaller set of atoms from P is not small. E.g., the model $M' = \{x_3, x_4, x_5\}$ is a model for T and does not contain any atom from P. However, since M' contains three atoms and $k = 2$, M' is not a small model and hence it cannot disprove the $P;Z$-minimality of M. For completeness, note that ϕ is not a consequence of every small $P;Z$-minimal model for T. For instance, $\{x_2, x_4\}$ is a small $P;Z$-minimal model for T that does not entail ϕ.

[2] In this paper, whenever we speak about $P;Z$-minimality, we mean minimality as defined here. Note that our requirement $A = P \cup Z$ makes the specification of Z redundant, and we could just speak of P-minimality. However, since we will often explicitly refer to the Z-variables, we keep the P-Z notation.

Clearly, for each fixed k, SMC is tractable. In fact it sufficed to enumerate $|A|^k$ candidate interpretations in an outer loop and for each such interpretation M check whether $M \models T$, $M \models \phi$, and M is $P;Z$-minimal. The latter can be done by an inner loop enumerating all small interpretations and performing some easy checking tasks.

It is also not hard to see that SMC is fp-intractable. In fact the Hitting Set problem, which was shown to be $W[2]$-complete [14], can be fp-reduced to SMC and can be actually regarded as the restricted version of SMC where $P = A$, $Z = \emptyset$, and T consists of a CNF having only positive literals. In Section 6.2 we present the fp-tractable subclass of this version of SMC, where the maximum clause length in the theory is taken as an additional parameter. However, in Section 6.3 we show that, as soon as the set Z of floating variables is not empty, this problem becomes fp-intractable.

Since brave reasoning under minimal models was shown to be Σ_2^{P} complete in [15], and is thus one level above the complexity of classical reasoning, it would be interesting to determine the precise fixed-parameter complexity of the general version of SMC with respect to parameter k. This problem too is tackled in Section 6.3.

6.2. A tractable restriction of SMC

We restrict SMC by requiring that the theory T be a q-CNF with no negative literal occuring in it, and by minimizing over all atoms occurring in the theory. The problem *Restricted Small Model Circumscription* (*RSMC*) is thus defined as SMC except that T is required to be a purely positive q-CNF formula, the "floating" set Z is empty, and the parameters are the maximum size k of the models to be considered, and the maximum size q of the number of literals in the largest conjunct (= clause) of T.

Theorem 22. *RSMC is fixed-parameter tractable.*

Proof. Since T is positive and $Z = \emptyset$, the set of minimal models of T to be considered are exactly the prime implicants of T having size $\leqslant k$. By Theorem 10, computing these prime implicants for a q-CNF theory is fp-tractable with respect to parameters k and q. Thus, the theorem easily follows. □

6.3. The fixed-parameter complexity of SMC

We first show that the slight modification of the fp-tractable problem RSMC where $Z \neq \emptyset$ is fp-intractable and in fact $W[SAT]$ hard.

The problem *Positive Small Model Circumscription* (*PSMC*) is defined as SMC except that T is required to be a purely positive q-CNF formula, and the parameters are the maximum size k of the models to be considered, and the maximum clause length q.

Let us define the Boolean formula $count^k(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a non-empty list of variables and $0 \leqslant k \leqslant n$, as follows:

- if $k = 0$, then $count^k(\boldsymbol{x}) = \bigwedge_{1 \leqslant i \leqslant n} \neg x_i$;
- if $k \geqslant 1$, then $count^k(\boldsymbol{x}) = A \wedge B \wedge C$, where

$$A = \bigwedge_{1\leqslant i\leqslant n} \bigwedge_{1\leqslant j\leqslant \min\{i,k+1\}} \left(\left(\bigvee_{j-1\leqslant t\leqslant i-1} \left(q_t^{j-1} \wedge \bigwedge_{t+1\leqslant s\leqslant i-1} \neg x_s\right) \wedge x_i\right) \equiv q_i^j\right),$$

$$B = \bigwedge_{k+1\leqslant r\leqslant n} \neg q_r^{k+1} \quad \text{and} \quad C = \bigvee_{k\leqslant r\leqslant n} q_r^k.$$

All the indices in the above formula are greater than zero. Any literal having an index less than one or not satisfying the prescribed bounds with respect to the other indices does not belong to the formula. (See the example below.)

Intuitively, $\mathit{count}^k(\boldsymbol{x})$ "counts" whether exactly k variables among $x_1,\ldots,x_n$ are "true." If $k>0$, in any satisfying truth value assignment for $\mathit{count}^k(\boldsymbol{x})$, the propositional variable q_i^j gets the value *true* iff x_i is the jth true variable among $x_1,\ldots,x_i$. Note that the size of $\mathit{count}^k(\boldsymbol{x})$ is $O(kn^2)$ if $k\geqslant 1$, and $O(n)$ if $k=0$. Moreover, the same bounds hold for the time needed for computing this formula.

Example 23. We next describe the formula $\mathit{count}^2(x_1,x_2,x_3,x_4)$ that "counts" whether exactly two variables among variables x_1,x_2,x_3,x_4 are "true". By definition, $\mathit{count}^2(x_1,x_2,x_3,x_4)$ is the following formula:

$$\begin{aligned}
x_1 &\equiv q_1^1 \wedge\\
\neg x_1 \wedge x_2 &\equiv q_2^1 \wedge\\
q_1^1 \wedge x_2 &\equiv q_2^2 \wedge\\
\neg x_1 \wedge \neg x_2 \wedge x_3 &\equiv q_3^1 \wedge\\
\left(\left(q_1^1 \wedge \neg x_2\right) \vee q_2^1\right) \wedge x_3 &\equiv q_3^2 \wedge\\
q_2^2 \wedge x_3 &\equiv q_3^3 \wedge\\
\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 &\equiv q_4^1 \wedge\\
\left(\left(q_1^1 \wedge \neg x_2 \wedge \neg x_3\right) \vee \left(q_2^1 \wedge \neg x_3\right) \vee q_3^1\right) \wedge x_4 &\equiv q_4^2 \wedge\\
\left(\left(q_2^2 \wedge \neg x_3\right) \vee q_3^2\right) \wedge x_4 &\equiv q_4^3 \wedge\\
\neg q_3^3 \wedge \neg q_4^3 &\wedge\\
q_2^2 \vee q_3^2 \vee q_4^2 &
\end{aligned}$$

There is a one-to-one correspondence between the truth value assignments that make this formula true and the truth value assignments to x_1,x_2,x_3,x_4 that assign *true* to exactly two variables among them. For instance, consider the truth value assignment σ such that $\sigma(x_1)=\sigma(x_3)=\mathit{true}$ and $\sigma(x_2)=\sigma(x_4)=\mathit{false}$. Note that σ determines the truth value for all the other variables in the formula. Indeed, from $x_1\equiv q_1^1$, it follows that q_1^1 must be true. Moreover, since x_2 is false, both q_2^1 and q_2^2 must be false. Similarly, q_3^1 must be false, because the first true variable is x_1 and, in fact, the conjunction $\neg x_1\wedge\neg x_2\wedge x_3$ that "defines" q_3^1 is false. However, x_3 is the second true variable and q_3^2 is true, as $q_1^1\wedge\neg x_2$ holds. In the same way, it can be verified that q_3^3, q_4^1, q_4^2, and q_4^3 must be false. Therefore, the unique extension of σ to all the variables occurring in the count^2 formula is the truth

value assignment σ' that assigns *true* to x_1, x_3, q_1^1, and q_3^2, and *false* to all the other variables.

The variables $x_1, \ldots, x_n$ in the formula above are called the *external* variables of the formula, while all the other variables occurring in the formula are called *private* variables.

Whenever a theory T contains a *count* subformula, we assume without loss of generality that the private variables of this subformula do not occur in T outside the subformula. In particular, if T contains two *count* subformulas, then their set of private variables are disjoint.

The following lemma easily follows from the definition of the *count* formula and from the discussion above.

Lemma 24. *Let F be a formula and $\boldsymbol{x}$ a list of variables occurring in F. Then*

- *$F \wedge count^k(\boldsymbol{x})$ is satisfiable if and only if there exists a truth value assignment σ for F assigning true to exactly k variables from $\boldsymbol{x}$.*
- *Every k-truth value assignment σ satisfying F can be extended in a unique way to an assignment σ' satisfying $F \wedge count^k(\boldsymbol{x})$.*
- *Every satisfying truth value assignment for $F \wedge count^k(\boldsymbol{x})$ assigns true to exactly k private variables of $count^k(\boldsymbol{x})$ and true to exactly k variables from $\boldsymbol{x}$.*

Theorem 25. *PSMC is $W[SAT]$-hard. The problem remains hard even for 2-CNF theories.*

Proof. Let Φ be a Boolean formula over propositional variables $\{x_1, \ldots, x_n\}$. We fp-reduce the $W[SAT]$-complete problem of deciding whether there exists a k-truth value assignment satisfying Φ to an instance of PSMC where the maximum model size is $2k+1$, and the maximum clause length is 2.

Without loss of generality, assume that $k > 0$ and $n > 2$. Let

$$\Phi' = \Phi \wedge count^k(x_1, \ldots, x_n),$$

and let $y_1, \ldots, y_m$ be the private variables of the $count^k$ subformula. Moreover, let T be the following 2-CNF positive theory:

$$(p \vee x_1) \wedge \cdots \wedge (p \vee x_n) \wedge (p \vee y_1) \wedge \cdots \wedge (p \vee y_m).$$

We take $P = \{p\}$ and $Z = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$.

From the definition of $count^k$ and the assumptions that $k > 0$ and $n > 2$, it is easy to verify that $n + m > 2k + 1$ holds. It follows that a set M is a $P;Z$ minimal model of T having size at most $2k+1$ if and only if $M = \{p\} \cup S$, where S is any subset of Z such that $|M| \leqslant 2k$.

From Lemma 24, every satisfying truth value assignment for Φ' must make true exactly k variables from $\{x_1, \ldots, x_n\}$, and k variables from the set of private variables of $count^k$. It follows that there exists a $P;Z$ minimal model M of T such that $|M| \leqslant 2k+1$ and M satisfies Φ' if and only if there exists a k-truth value assignment satisfying Φ. □

Let us now focus on the general SMC problem, where both arbitrary theories are considered and floating variables are permitted. It does not appear that SMC is contained

in $W[SAT]$. On the other hand, it can be seen that SMC is contained in $AW[SAT]$, but it does not seem to be hard (and thus complete) for this class. In fact, $AW[SAT]$ is the miniaturization of PSPACE and not of Σ_2^{P}. No class corresponding to the levels of the polynomial hierarchy have been defined so far in the theory of the fixed-parameter intractability. Nonmonotonic reasoning problems, such as SMC, seem to require the definitions of such classes. We next define the exact correspondent of Σ_2^{P} at the fixed-parameter level.

6.3.1. Definition of the class $\Sigma_2 W[SAT]$

$\Sigma_2 W[SAT]$ is defined similarly to $AW[SAT]$, but the quantifier prefix is restricted to Σ_2.

Parameterized $\mathrm{QBF}_2\mathrm{SAT}$.

Instance: A quantified boolean formula $\exists^{k_1}\boldsymbol{x}\forall^{k_2}\boldsymbol{y}E$.

Parameter: $k = \langle k_1, k_2\rangle$.

Question: Is $\exists^{k_1}\boldsymbol{x}\forall^{k_2}\boldsymbol{y}E$ valid? (Here, $\exists^{k_1}\boldsymbol{x}$ denotes the choice of some k_1-truth value assignment for the variables $\boldsymbol{x}$, and $\forall^{k_2}\boldsymbol{y}$ denotes all choices of k_2-truth value assignments for the variables $\boldsymbol{y}$.)

Definition 26. $\Sigma_2 W[SAT]$ is the set of all problems that fp-reduce to Parameterized $\mathrm{QBF}_2\mathrm{SAT}$.

6.3.2. Membership of SMC in $\Sigma_2 W[SAT]$

Let the problem Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$ be the variant of the problem Parameterized $\mathrm{QBF}_2\mathrm{SAT}$ where the quantifiers $\exists^{k_1}\boldsymbol{x}$ and $\forall^{k_2}\boldsymbol{y}$ are replaced by quantifiers $\exists^{\leqslant k_1}\boldsymbol{x}$ and $\forall^{\leqslant k_2}\boldsymbol{y}$ with the following meaning. $\exists^{\leqslant k_1}\boldsymbol{x}\ \alpha$ means that there exists a truth value assignment making at most k_1 propositional variables from $\boldsymbol{x}$ true such that α is valid. Symmetrically, $\forall^{\leqslant k_2}\boldsymbol{y}\ \alpha$ means that α is valid for every truth value assignment making at most k_2 propositional variables from $\boldsymbol{y}$ true.

Lemma 27. Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$ *is in* $\Sigma_2 W[SAT]$.

Proof. It suffices to show that Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$ is fp-reducible to Parameterized $\mathrm{QBF}_2\mathrm{SAT}$.

Let $\Phi = \exists^{\leqslant k_1} x_1 x_2 \ldots x_n \forall^{\leqslant k_2} y_1 y_2 \ldots y_m\ E(x_1, \ldots, x_n, y_1, \ldots, y_m)$ be an instance of Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$. It is easy to see that the following instance Φ' of Parameterized $\mathrm{QBF}_2\mathrm{SAT}$ is equivalent to Φ:

$$\exists^{2k_1} x_1 x_2 \ldots x_n x_1' x_2' \ldots x_n' \forall^{2k_2} y_1 y_2 \ldots y_m y_1' y_2' \ldots y_m'$$
$$E(x_1 \wedge x_1', \ldots, x_n \wedge x_n', y_1 \wedge y_1', \ldots, y_m \wedge y_m'),$$

where $x_1', x_2', \ldots, x_n', y_1', y_2', \ldots, y_m'$ are new variables and $E(x_1 \wedge x_1', \ldots, x_n \wedge x_n', y_1 \wedge y_1', \ldots, y_m \wedge y_m')$ is obtained from E by substituting $x_i \wedge x_i'$ for x_i $(1 \leqslant i \leqslant n)$ and $y_j \wedge y_j'$ for y_j $(1 \leqslant j \leqslant m)$. □

Theorem 28. *SMC is in* $\Sigma_2 W[SAT]$.

Proof. By Lemma 27 it is sufficient to show that every SMC instance S can be *fp*-reduced to an equivalent instance $\Phi(S)$ of Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$. Let $S = (A = P \cup Z, T(P, Z), \phi, k)$ be an SMC instance, where $P = \{p_1, \ldots, p_n\}$ and $Z = \{z_1, \ldots, z_m\}$. Let $P' = \{p'_1, \ldots, p'_n\}$ and $Z' = \{z'_1, \ldots, z'_m\}$ be two sets of fresh variables. $\Phi(S)$ is defined as follows:

$$\exists^{\leqslant k} p_1 \ldots p_n z_1 \ldots z_m \forall^{\leqslant k} p'_1 \ldots p'_n z'_1 \ldots z'_m$$
$$T(P, Z) \wedge \phi \wedge$$
$$T(P', Z') \Rightarrow \Bigg(\bigwedge_{1 \leqslant i \leqslant n} p_i \equiv p'_i \Bigg) \vee \Bigg(\bigvee_{1 \leqslant i \leqslant n} p'_i \wedge \neg p_i \Bigg),$$

where $T(P', Z')$ is obtained from $T(P, Z)$ by substituting p'_i for p_i $(1 \leqslant i \leqslant n)$ and z'_j for z_j $(1 \leqslant j \leqslant m)$.

The first part of $\Phi(S)$ guesses a model M of T with at most k atoms among $P \cup Z$ which satisfies ϕ. The second part makes sure that the M is $P;Z$ minimal by checking that each model M' of T is either equivalent to M over the P variables, or has at least one P variable true whereas the same variable is false in M. Hence T bravely entails ϕ under small models $P;Z$ circumscription if and only if $\Phi(S)$ is valid. □

6.3.3. $\Sigma_2 W[SAT]$-hardness of SMC

Theorem 29. *SMC is $\Sigma_2 W[SAT]$-hard, and thus $\Sigma_2 W[SAT]$-complete.*

Proof. We show that Parameterized $\mathrm{QBF}_2\mathrm{SAT}$ is fp-reducible to SMC. Let Φ be the following instance of Parameterized $\mathrm{QBF}_2\mathrm{SAT}$.

$$\exists^{k_1} x_1 x_2 \ldots x_n \forall^{k_2} y_1 y_2 \ldots y_m \; E(x_1, \ldots, x_n, y_1, \ldots, y_m).$$

We define a corresponding instance of SMC $S(\Phi) = (A = P \cup Z, T, \; \phi = w, \; k = 2k_1 + 2k_2 + 1)$, where w is a fresh variable, $T = (E(\boldsymbol{x}, \boldsymbol{y}) \Rightarrow w) \wedge \mathit{count}^{k_1}(\boldsymbol{x}) \wedge \mathit{count}^{k_2}(\boldsymbol{y})$, $P = \boldsymbol{x} \cup \{w\}$, and Z consists of all the other variables occurring in T, namely, the variables in $\boldsymbol{y}$ and the private variables of the two *count* subformulae.

We prove that Φ is valid if and only if $S(\Phi)$ is a yes instance of SMC.

(*Only if part*) Assume Φ is valid. Then, there exists a k_1-truth value assignment σ to the variables $\boldsymbol{x}$ such that for every k_2-truth value assignment to the variables $\boldsymbol{y}$, the formula E is satisfied.

Let M be an interpretation for T constructed as follows. M contains the k_1 variables from $\boldsymbol{x}$ which are made true by σ and the first k_2 variables of $\boldsymbol{y}$; in addition, M contains w and $k_1 + k_2$ private variables which make true the two *count* subformulae. This is possible by Lemma 24.

It is easy to see that M is a model for T. We now show that M is a $P;Z$ minimal model of T. Assume that M' is a $P;Z$ smaller model. Due to the $\mathit{count}^{k_1}(\boldsymbol{x})$ subformula, M' must contain exactly k_1 atoms from $\boldsymbol{x}$ and therefore M and M' coincide with respect to the $\boldsymbol{x}$ atoms. It follows that $w \notin M'$. However, by validity of Φ and the construction of M, $M' \models E$ holds, and therefore $M' \models w$, as well. Contradiction.

(*If part*) Assume there exists a $P;Z$ minimal model M of T such that M entails w and $|M| \leqslant k$. Note that, by Lemma 24, it must hold that M contains exactly k_1 true variables from $\boldsymbol{x}$ and exactly k_2 true variables from $\boldsymbol{y}$.

Towards a contradiction, assume that Φ is not valid. Then it must hold that for every k_1-truth value assignment σ to the variables $\boldsymbol{x}$, there exists a k_2-truth value assignment σ' to the variables $\boldsymbol{y}$, such that $\sigma \cup \sigma'$ falsifies E. In particular, for the k_1 variables from $\boldsymbol{x}$ which are true according to M, it is possible to make true exactly k_2 variables from $\boldsymbol{y}$ such that the formula E is not satisfied. Consider now the interpretation M' containing these $k_1 + k_2$ true variables plus the $k_1 + k_2$ made true by the two *count* subformulae. M' is a model of T whose P variables coincide with those of M except for w which belongs to M, but not to M'. Therefore, M is not $P;Z$ minimal, a contradiction.

Finally, note that the transformation from Φ to $S(\Phi)$ is an fp-reduction. Indeed it is feasible in polynomial time and is just linear in k. □

Corollary 30. Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$ *is* $\Sigma_2 W[SAT]$*-complete.*

Proof. Completeness follows from the fact that, as shown in Lemma 27, this problem belongs to $\Sigma_2 W[SAT]$, and by Theorem 29, which shows that the $\Sigma_2 W[SAT]$-hard problem SMC is fp-reducible to Parameterized $\mathrm{QBF}_2\mathrm{SAT}_{\leqslant}$. □

Downey and Fellows [14] pointed out that completeness proofs for fixed-parameter intractability classes are generally more involved than classical intractability proofs. Note that this is also the case for the above proof, where we had to deal with subtle counting issues. A straightforward *downscaling* of the standard Σ_2^{P}-completeness proof for propositional circumscription appears not to be possible.

In particular, observe that we have obtained our completeness result for a very general version of propositional minimal model reasoning, where there are variables to be minimized (P) and floating variables (Z). It is well-known that minimal model reasoning remains Σ_2^{P}-complete even if *all* variables of a formula are minimized (i.e., if Z is empty). This result does not seem to carry over to the setting of fixed-parameter intractability. Clearly, this problem, being a restricted version of SMC, is in $\Sigma_2 W[SAT]$. Moreover it is easy to see that the problem is hard for $W[2]$ and thus fixed-parameter intractable. However, we were not able to show that the problem is complete for any class in the range from $W[2]$ to $\Sigma_2 W[SAT]$, and leave this issue as an open problem.

Open problem. Determine the fixed-parameter complexity of SMC when all variables of the theory T are to be minimized.

7. Conclusion

In this paper we have studied the fixed-parameter tractability of several problems in AI and nonmonotonic logic programming. We could show that many relevant problems in these areas are fixed-parameter tractable with respect to natural problem parameters. We have proposed new algorithms for the considered problems, and believe that these

algorithms can be used profitably both to construct new, more efficient AI systems, and to improve current implementations such as existing systems for nonmonotonic logic programming, e.g., *Smodels* [33,34] and `dlv` [16,19].

As a more theoretical result we have shown that the problem of small model circumscription (SMC) is fixed-parameter intractable. This problem does not resemble any other fp-intractable problem previously studied in the literature [14]. In order to determine its exact degree of fp-intractability, we had to define the new fixed-parameter complexity class $\Sigma_2 W[SAT]$ and show that SMC is complete for this class. Note that this result holds for the general version of SMC; the case where all variables are minimized is also fp-intractable, but its exact complexity is open.

This paper and the paper by Truszczyński [39] are the first to explore fp-tractability issues in the context of AI. There are many further AI problems that call for an fp-complexity analysis. We are confident that interesting and useful results can be obtained in the areas of planning and automatic configuration.

Acknowledgements

We thank the anonymous referees for their useful comments and suggestions.

Research supported by *FWF* (*Austrian Science Funds*) under the project Z29-INF. Part of the work of Francesco Scarcello has been carried out while visiting the Technische Universität Wien.

References

[1] K. Apt, H. Blair, A. Walker, Towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Washington, DC, 1988, pp. 89–148.
[2] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, J. ACM 30 (3) (1983) 479–513.
[3] W. Bibel, Constraint satisfaction from a deductive viewpoint, Artificial Intelligence 35 (1988) 401–413.
[4] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[5] M. Cadoli, L. Palopoli, F. Scarcello, Propositional lower bounds: Generalization and algorithms, Ann. Math. Artif. Intell. 27 (1999) 129–148.
[6] P. Cholewiński, M. Truszczyński, Extremal problems in logic programming and stable model computation, J. Logic Programming 38 (1999) 219–242.
[7] R. Dechter, Constraint networks, in: Encyclopedia of Artificial Intelligence, 2nd edition, Wiley, New York, 1992, pp. 276–285.
[8] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (1989) 353–366.
[9] J. de Kleer, A perspective on assumption-based truth maintenance, Artificial Intelligence 59 (1–2) (1993) 63–67.
[10] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness, Congr. Numer. 87 (1992) 161–187.
[11] R.G. Downey, M.R. Fellows, Fixed parameter intractability (extended abstract), in: Proc. 7th Annual Structure in Complexity Theory Conference, Boston, MA, 1992, pp. 36–49.
[12] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness I: Basic results, SIAM J. Comput. 24 (1995) 873–921.

[13] R.G. Downey, M.R. Fellows, On the parametric complexity of relational database queries and a sharper characterization of $W[1]$, in: Proc. DMTCS'96 Combinatorics Complexity and Logics, Springer, Berlin, 1996, pp. 164–213.
[14] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, New York, 1999.
[15] T. Eiter, G. Gottlob, Propositional circumscription and extended closed world reasoning are $\mathbf{\Pi}_2^P$-complete, Theoret. Comput. Sci. 114 (2) (1993) 231–245, Addendum, Theoret. Comput. Sci. 118 (1993) 315.
[16] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, F. Scarcello, The KR system `dlv`: Progress report comparisons and benchmarks, in: Proc. Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, Morgan Kaufmann, San Mateo, CA, 1998, pp. 406–417.
[17] T. Eiter, W. Faber, N. Leone, G. Pfeifer, Declarative problem-solving using the DLV system, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer Academic, Dordrecht, 2000.
[18] S. Even, Graph Algorithms, Computer Science Press, Potomac, MD, 1979.
[19] W. Faber, N. Leone, G. Pfeifer, Pushing goal derivation in DLP computations, in: M. Gelfond, N. Leone, G. Pfeifer (Eds.), Proc. 5th International Conference on Logic Programming Nonmonotonic Reasoning (LPNMR'99), El Paso, TX, Lecture Notes in Computer Science, Vol. 1730, Springer, Berlin, 1999, pp. 177–191.
[20] J. Flum, M. Grohe, Fixed-parameter tractability and logic, Preprint Nr. 23/1999, Mathematische Fakultät, University of Freiburg, Freiburg, Germany, 1999.
[21] S. Fortune, J.E. Hopcroft, J. Wyllie, The directed subgraph homeomorphism problem, Theoret. Comput. Sci. 10 (2) (1980) 111–121.
[22] E.C. Freuder, A sufficient condition for backtrack-bounded search, J. ACM 32 (4) (1985) 755–761.
[23] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-completeness, Freeman, New York, 1979.
[24] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Logic Programming: Proc. Fifth Internat. Conference Symposium, MIT Press, Cambridge, MA, 1988, pp. 1070–1080.
[25] G. Gottlob, N. Leone, F. Scarcello, The complexity of acyclic conjunctive queries, J. ACM 48 (3) (2001) 431–498. An extended abstract concerning part of this work has been published in: Proc. IEEE Symposium on Foundations of Computer Science (FOCS'98), Palo Alto, CA, 1998, pp. 706–715.
[26] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural CSP decomposition methods, Artificial Intelligence 124 (2000) 243–282. An extended abstract concerning part of this work has been published in: Proc. IJCAI-99, Stockholm, Sweden, 1999, pp. 394–399.
[27] M. Grohe, Descriptive and parametrized complexity, in: Proc. 13th International Workshop on Computer Science Logic (CSL'99), Lecture Notes in Computer Science, Vol. 1683, Springer, Berlin, 1999, pp. 264–273.
[28] P. Jeavons, D. Cohen, M. Gyssens, Closure properties of constraints, J. ACM 44 (4) (1997) 527–548.
[29] Ph.G. Kolaitis, M.Y. Vardi, Conjunctive-query containment and constraint satisfaction, in: Proc. of Symposium on Principles of Database Systems (PODS'98), 1998, pp. 205–213.
[30] D. Maier, The Theory of Relational Databases, Computer Science Press, Rockville, MD, 1986.
[31] W. Marek, M. Truszczyński, Nonmonotonic Logics—Context-Dependent Reasoning, Springer, Berlin, 1993.
[32] K. Menger, Zur allgemeinen Kurventheorie, Fund. Math. 10 (1927) 96–115.
[33] I. Niemelä, P. Simons, T. Soininen, Stable model semantics of weight constraint rules, in: M. Gelfond, N. Leone, G. Pfeifer (Eds.), Proc. 5th International Conference on Logic Programming Nonmonotonic Reasoning (LPNMR'99), El Paso, TX, Lecture Notes in Computer Science, Vol. 1730, Springer, Berlin, 1999, pp. 317–331.
[34] I. Niemelä, P. Simons, Smodels—An implementation of the stable model and well-founded semantics for normal LP, in: J. Dix, U. Furbach, A. Nerode (Eds.), Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97), Dagstuhl, Germany, Lecture Notes in Computer Science, Vol. 1265, Springer, Berlin, 1997, pp. 421–430.
[35] C.H. Papadimitriou, M. Yannakakis, On the complexity of database queries, in: Proc. of Symp. on Principles of Database Systems (PODS'97), Tucson, AZ, 1997, pp. 12–19.
[36] J. Pearson, P.G. Jeavons, A survey of tractable constraint satisfaction problems, CSD-TR-97-15, Royal Holloway, Univ. of London, 1997.
[37] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1) (1987) 57–95.

[38] N. Robertson, P.D. Seymour, Graph minors II. Algorithmic aspects of tree-width, J. Algorithms 7 (1986) 309–322.
[39] M. Truszczyński, Computing large and small stable models, in: Proc. 16th International Conference on Logic Programming (ICLP'99), Las Cruces, NM, 1999, pp. 169–183.
[40] M. Vardi, Complexity of relational query languages, in: Proc. 14th ACM Symposium on Theory of Computing (STOC'82), San Francisco, CA, 1982, pp. 137–146.
[41] M. Yannakakis, Algorithms for acyclic database schemes, in: C. Zaniolo, C. Delobel (Eds.), Proc. Internat. Conference on Very Large Data Bases (VLDB'81), Cannes, France, 1981, pp. 82–94.