Computing large and small stable models

Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046, USA
mirek@cs.uky.edu

Abstract

In this paper, we focus on the problem of existence of and computing *small* and *large* stable models. We show that for every fixed integer k, there is a linear-time algorithm to decide the problem LSM (large stable models problem): does a logic program P have a stable model of size at least $|P|-k$. In contrast, we show that the problem SSM (small stable models problem) to decide whether a logic program P has a stable model of size at most k is much harder. We present two algorithms for this problem but their running time is given by polynomials of order depending on k. We show that the problem SSM is *fixed-parameter intractable* by demonstrating that it is $W[2]$-hard. This result implies that it is unlikely, an algorithm exists to compute stable models of size at most k that would run in time $O(n^c)$, where c is a constant independent of k. We also provide an upper bound on the fixed-parameter complexity of the problem SSM by showing that it belongs to the class $W[3]$.

1 Introduction

The stable model semantics by Gelfond and Lifschitz [10] is one of the two most widely studied semantics for normal logic programs, the other one being the well-founded semantics by Van Gelder, Ross and Schlipf [17]. Among 2-valued semantics, the stable model semantics is commonly regarded as the one providing the correct meaning to the negation operator in logic programming. It coincides with the least model semantics on the class of Horn programs, and with the well-founded semantics and the perfect model semantics on the class of stratified programs [1]. In addition, the stable model semantics is closely related to the notion of a default extension by Reiter [12, 4]. Logic programming with stable model semantics has applications in knowledge representation, planning and reasoning about action. It was also recently proposed as a computational paradigm especially well suited for solving combinatorial optimization and constraint satisfaction problems [14, 15].

The problem with the stable model semantics is that, even in the propositional case, reasoning with logic programs under the stable model semantics is computationally hard. It is well-known that deciding whether a finite propositional logic program has a stable model is NP-complete [13]. Consequently, it is not at all clear that logic programming with the stable model semantics can serve as a practical computational tool.

This issue can be resolved by implementing systems computing stable models and by experimentally studying the performance of these systems. Several such projects are now under way. Niemelä and Simons [16] developed a system, *smodels*, for computing stable models of finite function symbol-free logic programs and reported very promising performance results. For some classes of programs, *smodels* decides the existence of a stable model in a matter of seconds even if an input program consists of tens of thousands of clauses. Encouraging results on using *smodels* to solve planning problems are reported in [15]. Another well-advanced system is DeReS [6], designed to compute extensions of arbitrary propositional default theories but being especially effective for default theories encoding propositional logic programs with good relaxed stratification. Finally, systems capable of reasoning with disjunctive logic programs were described in [9] and [2].

However, faster implementations will ultimately depend on better understanding of the algorithmic aspects of reasoning with logic programs under the stable model semantics. In this paper, we investigate the complexity of deciding whether a finite propositional logic program has stable models of some restricted sizes. Specifically, we study the following two problems ($|P|$ stands for the number of rules in a logic program P):

LSM (Large stable models) Given a finite propositional logic program P and an integer k, decide whether there is a stable model of P of size at least $|P| - k$.

SSM (Small stable models) Given a finite propositional logic program P and an integer k, decide whether there is a stable model of P of size no more than k.

Inputs to the problems LSM and SSM are pairs (P, k), where P is a finite propositional logic program and k is a non-negative integer. Problems of this type are referred to as *parametrized* decision problems. By fixing a parameter, a parameterized decision problem gives rise to its *fixed-parameter* version. In the case of problems LSM and SSM, by fixing k we obtain the following two fixed-parameter problems (k is now no longer a part of input):

$LSM(k)$ Given a finite propositional logic program P, decide whether P has a stable model of size at least $|P| - k$.

$SSM(k)$ Given a finite propositional logic program P, decide whether P has a stable model of size at most k.

The problems LSM and SSM are NP-complete. It follows directly from the NP-completeness of the problem of existence of stable models [13]. But fixing k makes a difference! Clearly, the fixed-parameter problems $SSM(k)$ and $LSM(k)$ can be solved in polynomial time. Indeed, consider a finite propositional logic program P with the set of atoms $At(P)$. Then, there are $O(|At(P)|^k)$ subsets of $At(P)$ of cardinality at most k. For each such subset M, it can be checked in time linear in the size of P (the total number of all occurrences of atoms in P; in the paper we will denote this number by $size(P)$) whether M is a stable model of P. Thus, one can decide whether P has a stable model of size at most k in time $O(size(P) \times |At(P)|^k)$.

Similarly, there are only $O(|P|^k)$ subsets of P of size at least $|P| - k$. Each such subset is a candidate for the set of generating rules of a stable model of size at least $|P| - k$ (and smaller subsets, clearly, are not). Given such a subset R, one can check in time $O(size(P))$ whether R generates a stable model for P. Thus, it follows that there is an algorithm that decides in time $O(size(P) \times |P|^k)$ whether a logic program P has a stable model of size at least $|P| - k$.

While both algorithms are polynomial in the size of the program, their asymptotic complexity is expressed by the product of the size of a program and of a polynomial of order k in the number of atoms (or rules) of the program. Even for small values of k, say for $k \geq 4$, the functions $size(P) \times |At(P)|^k$ and $size(P) \times |P|^k$ grow very fast with $size(P)$, $|At(P)|$ and $|P|$, and render the corresponding algorithms infeasible.

An important question is whether algorithms for problems $SSM(k)$ and $LSM(k)$ exist whose order is significantly lower than k, preferably, a constant independent of k. The study of this question is the main goal of our paper. A general framework for such investigations was proposed by Downey and Fellows [7, 8]. They introduced the concepts of *fixed-parameter tractability* and *fixed-parameter intractability* that are defined in terms of a certain hierarchy of complexity classes known as the *W hierarchy*.

In the paper, we show that the problem LSM is fixed-parameter tractable and demonstrate an algorithm that for every fixed k decides the problem $LSM(k)$ in linear time — a significant improvement over the straightforward algorithm presented earlier.

On the other hand, we demonstrate that the problem SSM is much harder. We outline an algorithm to decide the problems $SSM(k)$, $k \geq 1$, that is asymptotically faster than the simple algorithm described above but the improvement is rather insignificant. Our algorithm runs in time $O(size(P) \times |At(P)|^{k-1})$, an improvement only by the factor of $|At(P)|$. The difficulty in finding a substantially better algorithm is not coincidental. We provide evidence that the problem SSM is *fixed-parameter intractable* and, thus, it is unlikely that there is an algorithm to decide the problems $SSM(k)$ whose running time would be given by a polynomial of order independent of k.

The study of fixed-parameter tractability of problems occuring in the area of nonmonotonic reasoning is a relatively new research topic. Another paper that pursues this direction is [11].

2 Fixed-parameter intractability

This section recalls basic ideas of the work of Downey and Fellows on fixed-parameter intractability. The reader is referred to [7, 8] for a detailed treatment of this subject.

Informally, a *parametrized* decision problem is a decision problem whose inputs are pairs of items, one of which is referred to as a *parameter*. The graph colorability problem is an example of a parametrized problem. The inputs are pairs (G, k), where G is an undirected graph and k is a non-negative integer. The problem is to decide whether G can be colored with at most k colors. The problems SSM and LSM are also examples of parametrized decision problems. Formally, a *parametrized* decision problem is a set $L \subseteq \Sigma^* \times \Sigma^*$, where Σ^* is a fixed alphabet.

By selecting a concrete value $y \in \Sigma^*$ of the parameter, a parametrized decision problem L gives rise to an associated *fixed-parameter* problem $L_y = \{x : (x, y) \in L\}$. For instance, by fixing the value of k to 3, we get a fixed-parameter version of the colorability problem, known as 3-colorability. Inputs to the 3-colorability problem are graphs and the question is to decide whether an input graph can be colored with 3 colors. Clearly, the problems $SSM(k)$ and $LSM(k)$ are fixed-parameter versions of the problems SSM and LSM, respectively.

The interest in the fixed-parameter problems stems from the fact that they are often computationally easier than the corresponding parametrized problems. For instance, the problems SSM and LSM are NP-complete yet, as we saw earlier, their parametrized versions $SSM(k)$ and $LSM(k)$ can be solved in polynomial time. A word of caution is in order here. It is not always the case that fixed-parameter problems are easier. For instance, the 3-colorability problem is still NP-complete.

As we already pointed out, the fact that a problem admits a polynomial-time solution does not necessarily mean that practical algorithms to solve it exist. An algorithm that runs in time $O(n^{15})$, where n is the size of the input, is hardly more practical than an algorithm with an exponential running time (and may even be a worse choice in practice). The algorithms we presented so far to argue that the problems $SSM(k)$, $LSM(k)$ are in P rely on searching through the space of n^k possible solutions (where n is the number of atoms or rules of a program). Thus, these algorithms are not practical (except for the very smallest values of k). The key question is how fast those polynomial-time solvable fixed-parameter problems can really be solved. Or, in other words, can one significantly improve over the brute-force approach?

A technique to deal with such questions is provided by the fixed-parameter intractability theory of Downey and Fellows [7]. A parametrized problem $L \subseteq \Sigma^* \times \Sigma^*$ is *fixed-parameter tractable* if there exist a constant p, an integer function f and an algorithm A such that A determines whether $(x, y) \in L$ in time $f(|y|)|x|^p$ ($|z|$ stands for the length of a string $z \in \Sigma^*$). The class of fixed-parameter tractable problems will be denoted by FPT. Clearly, if a parametrized problem L is in FPT, each of the associated fixed-parameter problems L_y is solvable in polynomial time by an algorithm whose exponent does not depend on the value of the parameter y. It is known (see [7]) that, for instance, the vertex cover problem is in FPT.

There is substantial evidence supporting a conjecture that some parametrized problems whose fixed-parameter versions belong to P are not fixed-parameter tractable. To study and compare complexity of parametrized problems Downey and Fellows proposed the following notion of reducibility[1]. A parametrized problem L can be *reduced* to a parametrized problem L' if there exist a constant p, an integer function q and an algorithm A that to each instance (x, y) of L assigns an instance (x', y') of L' such that

1. x' depends upon x and y and y' depends upon y only,
2. A runs in time $O(q(|y|)|x|^p)$,
3. $(x, y) \in L$ if and only if $(x', y') \in L'$.

Downey and Fellows also defined a hierarchy of complexity classes called the *W hierarchy*:

$$\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \mathrm{W}[3] \ldots \tag{1}$$

The classes W[t] can be described in terms of problems that are complete for them (a problem D is *complete* for a complexity class $\mathcal{E}$ if $D \in \mathcal{E}$ and every problem in $\mathcal{E}$ can be reduced to D). Let us call a boolean formula *t-normalized* if it is of the form of products-of-sums-of-products ... of literals, with t being the number of products-of, sums-of expressions in this definition. For example, 2-normalized formulas are products of sums of literals. Thus, the class of 2-normalized formulas is precisely the class of CNF formulas. Define the *weighted t-normalized satisfiability problem* as:

$WS(t)$ Given a t-normalized formula φ and an integer k, decide whether there is a model of φ with at most k atoms (or, alternatively, decide whether there is a satisfying valuation for φ which assigns the logical value **true** to at most k atoms).

It is believed that the problems $WS(t)$, $t \geq 2$, are *not* fixed-parameter tractable and that for different values of t they are of different difficulty. Downey and Fellows show that for $t \geq 2$, the problems $WS(t)$ are complete for the class W[t]. They also show that a restricted version of the problem $WS(2)$:

$WS_3(2)$ Given a 3CNF formula φ and an integer k (parameter), decide whether there is a model of φ with at most k atoms

[1] The definition given here is sufficient for the needs of this paper. To obtain structural theorems a subtler definition is needed. This topic goes beyond the scope of the present paper. The reader is referred to [7] for more details.

is complete for the class $W[1]$. Downey and Fellows conjecture that all the implications in (1) are proper[2]. In particular, they conjecture that problems complete for the classes W[t], $t \geq 1$, are not fixed-parameter tractable.

In the paper, we relate the problem SSM to the problems $WS(2)$ and $WS(3)$ to place the problem SSM in the W hierarchy, to obtain estimates of its complexity and to argue for its fixed-parameter intractability.

3 Large stable models

In this section we will show an algorithm for the parametrized problem LSM that runs in time $O(2^{k+k^2} \times size(P))$, where (P, k) is an input instance. This result implies that the problem LSM is fixed-parameter tractable and that for every fixed k there is a linear-time algorithm for the problem $LSM(k)$.

We start by introducing some basic notation. Given a logic program rule r, we define $h(r)$ to be the head of the rule r and $b(r)$ to be the set of atoms appearing in the body of r. We denote by $b^+(r)$ the set of atoms that appear positively in the body of r and by $b^-(r)$ the set of atoms that appear negated in the body of r. For a logic program P, by $H(P)$ we denote the set atoms of P that appear as heads of rules from P. Finally, given a logic program P and a set of atoms M, by P_M we denote the Gelfond-Lifschitz reduct of P with respect to M.

Given a logic program P, denote by P^* the logic program obtained from P by eliminating from the bodies of the rules in P all literals $\mathbf{not}(a)$, where a is not the head of any rule from P. The following well-known result states the key property of the program P^*.

Lemma 3.1 *A set of atoms M is a stable model of a logic program P if and only if M is a stable model of P^*.*

Since $|P| = |P^*|$, Lemma 3.1 implies that the problem LSM has a positive answer for (P, k) if and only if it has a positive answer for (P^*, k). Moreover, it is easy to see that P^* can be constructed from P in time linear in the size of P. Thus, when looking for algorithms to decide the problem LSM we may restrict our attention to programs in which every atom appearing negated in the body of a rule appears also as the head of a rule. We will denote the class of such logic programs by $\mathcal{C}$.

By P^k let us denote the program consisting of those rules r in P for which $|b^-(r)| \leq k$. We have the following lemma.

Lemma 3.2 *Let P be a logic program in $\mathcal{C}$ and let $M \subseteq H(P)$ be a set of atoms such that $|M| \geq |P| - k$.*

1. *M is a stable model of P if and only if M is a stable model of P^k*

[2] If true, this conjecture would imply that in the context of fixed-parameter tractability there is a difference between the complexity of weighted satisfiability for 3CNF and CNF formulas.

2. *If M is a stable model of P^k, then P^k has no more than $k+k^2$ different negated literals appearing in the bodies of its rules.*

The following algorithm for the problem $LSM(k)$ is implied by Lemmas 3.1 and 3.2.

1. Eliminate from the input logic program P all literals $\mathbf{not}(a)$, where a is not the head of any rule from P. Denote the resulting program by Q (that is, $Q = P^*$).

2. Compute the set of rules Q^k consisting of those rules r in Q for which $|b^-(r)| \leq k$.

3. Decide whether Q^k has a stable model M such that $|M| \geq |P| - k$.

By Lemmas 3.1 and 3.2, stable models of Q^k that have at least $|P| - k$ elements are precisely the stable models of P with at least $|P| - k$ elements. Thus, our algorithm is correct.

Notice that steps 1 and 2 can be implemented in time $O(size(P))$. To implement step 3, note that every stable model of the logic program Q^k is determined by a subset of $\bigcup\{b^-(r): r \in Q^k\}$ [5]. By Lemma 3.2, there are no more than 2^{k+k^2} such candidate subsets to consider. Checking for each of them whether it determines a stable model of Q^k can be implemented in time $O(size(Q^k)) = O(size(P))$. Consequently, our algorithm runs in time $O(2^{k+k^2} \times size(P))$.

Theorem 3.3 *The problem LSM is fixed-parameter tractable. Moreover, for each fixed k there is a linear-time algorithm to decide whether a logic program P has a stable model of size at least $|P| - k$.*

4 Computing stable models of size at most k

As already mentioned, there is a straightforward algorithm to decide the problem $SSM(k)$ that runs in time $O(size(P) \times n^k)$, where $n = |At(P)|$. This algorithm can be somewhat improved. In this section we will outline an algorithm for the problem $SSM(k)$ that runs in time $O(size(P) \times n^{k-1})$. We will provide a detailed description in the case $k = 2$ and comment on how to extend this algorithm to the case of an arbitrary k.

We say that a logic program P is *proper* if it satisfies the following three conditions:

(P1) for every rule $r \in P$, $h(r) \notin b^+(r)$

(P2) for every rule $r \in P$, $b^+(r) \cap b^-(r) = \emptyset$

(P3) $\bigcup\{b^-(r): r \in P\} \subseteq H(P)$ (that is, $P = P^*$).

Given a logic program P, its *proper core* is a logic program obtained from P by removing from P every clause that violates conditions (P1) or (P2) and by enforcing (P3). The following lemma is straightforward.

Lemma 4.1 *A set of atoms M is a stable model of a logic program P if and only if it is a stable model of its proper core.*

Clearly, a proper core of a program P can be constructed in time linear in the size of P. Hence, Lemma 4.1 allows us to restrict our discussion of algorithms to decide the problem $SSM(k)$ to the class of proper logic programs.

Let P be a logic program. By $P(k)$ we will denote the program obtained from P by removing from it each clause with more than k atoms appearing positively in its body. In our discussion below we will use the following result.

Lemma 4.2 *Let P be a proper logic program and let M be a set of atoms such that $|M| \leq k$. Then M is a stable model of P if and only if M is a stable model of $P(k)$.*

We will now present an algorithm to decide the problem $SSM(1)$. Define $P_0 = P(0)$ and $P_1 = P(1) \setminus P(0)$. In other words, P_i, $i = 0, 1$, consists of those rules in P that have exactly i different atoms occurring positively in the body. Next, for each atom a define:

$H_0(a) =$ the number of rules r in P_0 with $h(r) = a$ and $a \notin b^-(r)$
$H_1(a) =$ the number of rules r in P_1 with $a \in b^+(r)$ (since $r \in P_1$, there are no other positive atoms in the body of r)
$H_2(a) =$ the number of rules r in P_0 with $h(r) \neq a$ and $a \notin b^-(r)$.

We have the following lemma.

Lemma 4.3 *Let P be a proper logic program. The set $\{a\}$ is a stable model of $P(1)$ if and only if $H_0(a) \geq 1$ and $H_1(a) = H_2(a) = 0$.*

Clearly, the tables H_0, H_1 and H_2 can be computed in time $O(size(P))$. Since it takes linear time to decide whether the empty set is a stable model of a program, Lemmas 4.2 and 4.3 imply a linear-time algorithm to decide whether a logic program P has a stable model of size at most 1.

We will next describe an algorithm to decide whether a logic program has a stable model of size at most 2. Consider a proper logic program P. As before, define $P_0 = P(0)$ and $P_1 = P(1) \setminus P(0)$. In addition, define $P_2 = P(2) \setminus P(1)$.

For every two different atoms a and b in P define:

$G_0(a, b) =$ the number of rules r in P_0 with $h(r) = a$, $a \notin b^-(r)$ and $b \notin b^-(r)$

$G_1(a,b) =$ the number of rules r in P_1 such that $h(r) = b$, $b^+(r) = \{a\}$, $a \notin b^-(r)$ and $b \notin b^-(r)$
$G_2(a,b) =$ the number of rules r in P_2 with $b^+(r) = \{a,b\}$
$G_3(a,b) =$ the number of rules r in P_1 with $b^+(r) = \{a\}$, $h(r) \neq b$ and $b \notin b^-(r)$
$G_4(a,b) =$ the number of rules r in P_0 with $h(r) \notin \{a,b\}$, $a \notin b^-(r)$ and $b \notin b^-(r)$.

We have the following lemma.

Lemma 4.4 *Let P be a proper logic program. Then, the set $\{a,b\}$ is a stable model for P if and only if $G_2(a,b) = G_3(a,b) = G_4(a,b) = 0$ and at least one of the following three conditions holds:*

1. $G_0(a,b) \geq 1$ *and* $G_0(b,a) \geq 1$
2. $G_0(a,b) \geq 1$ *and* $G_1(a,b) \geq 1$
3. $G_0(b,a) \geq 1$ *and* $G_1(b,a) \geq 1$.

Observe that each of the arrays G_i, $0 \leq i \leq 4$, can be computed in time $O(n \times size(P))$. Thus, Lemmas 4.2 and 4.4 imply the following algorithm to decide the problem $SSM(2)$:

1. Decide whether $SSM(1)$ holds. If so, output YES and stop (as shown earlier, this task takes $O(size(P))$ steps)

2. Otherwise, use the algorithm implied by Lemma 4.4 to decide whether P has stable models of size 2 (that is, compute tables G_i and check the condition of Lemma 4.4 for each set of two atoms). If so, output YES and otherwise output NO.

Let $|At(P)| = n$. Since there are $O(n^2)$ two-element subsets of $At(P)$ and since $n^2 = O(n \times size(P))$, our algorithm can be implemented to run in time $O(n \times size(P))$, where n is the number of atoms occurring in P. However, the algorithm requires that several $n \times n$ arrays be maintained.

The algorithms presented in this section can be extended to the case of an arbitrary k. We will only present a very general outline here. The details are rather complex and are omitted. First, observe that by Lemmas 4.1 and 4.2, it is enough to describe the algorithm for proper logic programs with at most k positive atoms in the body. Hence, consider such a program P. As in the case of $k = 2$, we first compute programs $P_0 = P(0)$, and $P_i = P(i) \setminus P(i-1)$, $1 \leq i \leq k$. Next, we establish a lemma, corresponding to Lemmas 4.3 and 4.4 that we used in the cases $k = 1$ and $k = 2$, characterizing stable models of size at most k in terms of the numbers of rules in the programs P_i satisfying certain properties. These numbers can be arranged in no more than $f(k)$ tables (for some function f) of dimensions no more than k. One can show that these tables can be computed in time $O(size(P) \times n^{k-1})$ and that the whole algorithm can also be implemented to run in time $O(size(P) \times n^{k-1})$.

5 Complexity of the problem SSM

The algorithm outlined in the previous section is not quite satisfactory. First, the detailed description is quite complex, Second, it poses high space requirements that are of the order $\Theta(n^k)$. A natural question to ask is: are there significantly better algorithms for the problems $SSM(k)$?

In this section we address this question by studying the complexity of the problem SSM. Our goal is to show that the problem is difficult in the sense of the W hierarchy. To this end we will show that the problem $WS(2)$ can be reduced to the problem SSM, that is, that the problem SSM is $W[2]$-hard. Given the overwhelming evidence of fixed-parameter intractability of problems that are $W[2]$-hard [7], it is unlikely that algorithms for problems $SSM(k)$ exist whose asymptotic behavior would be given by a polynomial of order independent of k. To better delineate the location of the problem SSM in the W hierarchy we also provide an upper bound on its hardness by showing that it belongs to the class $W[3]$.

We will start by showing that the problem $SSM(k)$ is reducible (in the sense of the definition from Section 2) to the problem $WS(3)$. To this end, we describe an encoding of a logic program P by means of a collection $T(P)$ of 3-normalized formulas so that P has a stable model of size at most k if and only if $T(P)$ has a model with no more than $(k+1)(k^2+2k)$ atoms. In the general setting of the class NP, an explicit encoding of the problem of existence of stable models in terms of propositional satisfiability was described in [3]. Our encoding, while different in key details, uses some ideas from that paper.

Let us consider an integer k and a logic program P. For each atom q in P let us introduce new atoms $c(q)$, $c(q,i)$, $1 \leq i \leq k+1$, and $c^-(q,i)$, $2 \leq i \leq k+1$. Intuitively, atom $c(q)$ represents the fact that in the process of computing the least model of the reduct of P with respect to some set of atoms, atom q is computed no later than during the iteration $k+1$ of the van Emden-Kowalski operator. Similarly, atom $c(q,i)$ represents the fact that in the same process atom q is computed exactly in the iteration i of the van Emden-Kowalski operator. Finally, atom $c^-(q,i)$, expresses the fact that q is computed *before* the iteration i of the van Emden-Kowalski operator. The formulas $F_1(q,i)$, $2 \leq i \leq k+1$, and $F_2(q)$ describe some basic relationships between atoms $c(q)$, $c(q,i)$ and $c^-(q,i)$ that we will require to hold:

$$F_1(q,i) = c^-(q,i) \Leftrightarrow c(q,1) \vee \ldots \vee c(q,i-1),$$

$$F_2(q) = c(q) \Leftrightarrow c(q,1) \vee \ldots \vee c(q,k+1).$$

Let r be a rule in P with $h(r) = q$, say

$$r = q \leftarrow a_1, \ldots, a_m, \mathbf{not}(b_1), \ldots, \mathbf{not}(b_n).$$

Define a formula $F_3(r,i)$, $2 \leq i \leq k+1$, by

$$F_3(r,i) = c^-(a_1,i) \wedge \ldots \wedge c^-(a_m,i) \wedge \neg c(b_1) \wedge \ldots \wedge \neg c(b_n) \wedge \neg c^-(q,i).$$

Define also $F_3(r, 1) = \textbf{false}$ if $m \geq 1$ and

$$F_3(r, 1) = \neg c(b_1) \wedge \ldots \wedge \neg c(b_k),$$

otherwise. Speaking informally, formula $F_3(r, i)$ asserts that q is computed by means of rule r in the iteration i of the least model computation process and that it has not been computed earlier.

Let $r_1, \ldots, r_t$ be all rules in P with atom q in the head. Define a formula $F_4(q, i)$, $1 \leq i \leq k+1$, by

$$F_4(q, i) = c(q, i) \Leftrightarrow F_3(r_1, i) \vee \ldots \vee F_3(r_t, i).$$

Intuitively, the formula $F_4(q, i)$ expresses the definition of $c(q, i)$ (recall that $c(q, i)$ stands for the following statement: when computing the least model of the reduct of P, atom q is first computed in the iteration i).

We will now define the theory $T_0(P)$ that encodes the problem of existence of small stable models. Put

$$\begin{aligned} T_0(P) \quad = \quad & \{F_1(q, i) \colon q \in At(P),\ 1 \leq i \leq k+1\} \cup \{F_2(q) \colon q \in At(P)\} \cup \\ & \{F_4(q, i) \colon q \in At(P),\ 1 \leq i \leq k+1\}. \end{aligned}$$

We will now establish some useful properties of the theory $T_0(P)$. First, consider a set U of atoms that is a model of $T_0(P)$. Define

$$M = \{q \in At(P) \colon c(q) \in U\}.$$

Lemma 5.1 *Let $q \in M$. Then there is a unique integer i, $1 \leq i \leq k+1$, such that $c(q, i) \in U$.*

For every atom $q \in M$ define i_q to be the integer whose existence and uniqueness is guaranteed by Lemma 5.1. Define $i_U = \max\{i_q \colon q \in M\}$. Next, for each i, $1 \leq i \leq i_U$ define

$$M_i = \{q \in M \colon i_q = i\}.$$

Lemma 5.2 *For every i, $1 \leq i \leq i_U$, $M_i \neq \emptyset$.*

Lemma 5.2 implies that if $|M| \leq k$, then $i_U \leq k$.

Lemma 5.3 *Assume that $|M| \leq k$. Then M is a stable model of P.*

Consider now a stable model M of the program P and assume that $|M| \leq k$. Clearly, $M = \bigcup_{i=1} T^i_{P_M}(\emptyset)$. For each atom $q \in M$ define s_q to be the least integer s such that $q \in T^s_{P_M}(\emptyset)$. Clearly, $s_q \geq 1$. Moreover, since $|M| \leq k$, it follows that for each $q \in M$, $s_q \leq k$. Now, define

$$U_M = \{c(q), c(q, s_q) \colon q \in M\} \cup \{c^-(q, i) \colon q \in M,\ s_q < i \leq k+1\}$$

Lemma 5.4 *The set of atoms U_M is a model of $T_0(P)$.*

Lemmas 5.1 - 5.4 add up to a proof of the following result.

Theorem 5.5 *Let k be a non-negative integer and let P be a logic program. The program P has a stable model of size at most k if and only if the theory $T_0(P)$ has a model U such that $|\{q \in At(P) \colon c(q) \in U\}| \leq k$.*

We will now modify the theory $T_0(P)$ to construct a theory $T(P)$ that will demonstrate that the problem $SSM(k)$ can be reduced to the problem $WS(3)$. First, for each atom $q \in At(P)$, introduce k^2+2k new atoms $d(q,i)$, $1 \leq i \leq k^2+2k$, and define

$$C_0(q) = \{\neg c(q) \vee d(q,i) \colon 1 \leq i \leq k^2+2k\} \cup \{c(q) \vee \neg d(q,i) \colon 1 \leq i \leq k^2+2k\}.$$

Next, define

$$\begin{aligned} C_1(q,i) \quad = \quad & \{\neg c^-(q,i) \vee c(q,1) \vee \ldots \vee c(q,i-1)\} \cup \\ & \{\neg c(q,j) \vee c^-(q,i) \colon 1 \leq j \leq i-1\}, \end{aligned}$$

$$C_2(q) = \{\neg c(q) \vee c(q,1) \vee \ldots \vee c(q,k+1)\} \cup \{\neg c(q,j) \vee c(q) \colon 1 \leq j \leq k+1\},$$

and

$$C_4(q,i) = \{\neg c(q,i) \vee F_3(r_1,i) \vee \ldots \vee F_3(r_t,i)\} \cup \{\neg F_3(r_j,i) \vee c(q,i) \colon 1 \leq j \leq m\},$$

where $\{r_1,\ldots,r_t\}$ is the set of all rules in P with q in the head.

Finally, define

$$\begin{aligned} T(P) \quad = \quad & \{C_0(q) \colon q \in At(P)\} \cup \{C_1(q,i) \colon q \in At(P), 1 \leq i \leq k+1\} \cup \\ & \{C_2(q) \colon q \in At(P)\} \cup \{C_4(q,i) \colon q \in At(P), 1 \leq i \leq k+1\} \end{aligned}$$

It is easy to see that the set of clauses $C_1(q,i)$ is equivalent to the formula $F_1(q,i)$, the set of clauses $C_2(q)$ is equivalent to the formula $F_2(q)$, and the set $C_4(q,i)$ of disjunctions of conjunctions of literals is equivalent to the formula $F_4(q,i)$. Thus, the union of the last three sets of clauses in the definition of $T(P)$ is logically equivalent to the theory $T_0(P)$. It follows that $U \subseteq \{c(q) \colon q \in At\} \cup \{c(q,i) \colon q \in M,\ 1 \leq i \leq k+1\} \cup \{c^-(q,i) \colon q \in M,\ 2 \leq i \leq k+1\}$ is a model of $T_0(P)$ if and only if $U \cup \{d(q,i) \colon c(q) \in U, 1 \leq i \leq k^2+2k\}$ is a model of $T(P)$. Moreover, every model of $T(P)$ is of the form $U \cup \{d(q,i) \colon c(q) \in U, 1 \leq i \leq k^2+2k\}$, where $U \subseteq \{c(q) \colon q \in At\} \cup \{c(q,i) \colon q \in M,\ 1 \leq i \leq k+1\} \cup \{c^-(q,i) \colon q \in M,\ 2 \leq i \leq k+1\}$ is a model of $T_0(P)$.

The role of the clauses in the sets $C_0(q)$ is to decrease the effect of the atoms $c^-(q,i)$ and $c(q,i)$ on the size of models of $T(P)$. Consequently, given a model U of $T_0(P)$, we can derive a bound on $|\{q \in At(P) \colon c(q) \in U\}|$ from a bound on the size of the model of $T(P)$ corresponding to U. Specifically, one can show that $T_0(P)$ has a model U with $|\{q \in At(P) \colon c(q) \in U\}| \leq k$ if and only if $T(P)$ has a model of size at most $(k+1)(k^2+2k)$. Thus, by

Theorem 5.5, one can show that P has a stable model of size at most k if and only if $T(P)$ has a model of size at most $(k+1)(k^2+2k)$. In other words, the problem SSM can be reduced to the problem $WS(3)$ (note that $T(P)$ consists of 3-normalized formulas).

Theorem 5.6 *The problem $SSM(k) \in W[3]$.*

Next, we will show that the problem $WS(2)$ can be reduced to the problem SSM. Let $C = \{c_1, \ldots, c_m\}$ be a collection of clauses. Let $A = \{x_1, \ldots, x_n\}$ be the set of atoms appearing in clauses in C. For each atom $x \in A$, introduce k new atoms $x(i)$, $1 \leq i \leq k$. By S_i, $1 \leq i \leq k$, we denote the logic program consisting of the following n clauses:

$$x_1(i) \leftarrow \mathbf{not}(x_2(i)), \ldots, \mathbf{not}(x_n(i))$$
$$\ldots$$
$$x_n(i) \leftarrow \mathbf{not}(x_1(i)), \ldots, \mathbf{not}(x_{n-1}(i))$$

Define $S = \bigcup_{i=1}^{k} S_i$. Clearly, each stable model of S is of the form $\{x_{j_1}(1), \ldots, x_{j_k}(k)\}$, where $1 \leq j_p \leq n$ for $p = 1, \ldots, k$. Sets of this form can be viewed as representations of nonempty subsets of the set A that have no more than k elements. This representation is not one-to-one, that is, some subsets have multiple representations.

Next, define P_1 to be the program consisting of the clauses

$$x_j \leftarrow x_j(i), \quad j = 1, \ldots, n, \quad i = 1, 2, \ldots, k.$$

Stable models of the program $S \cup P_1$ are of the form $\{x_{j_1}(1), \ldots, x_{j_k}(k)\} \cup M$, where M is a nonempty subset of A such that $|M| \leq k$ and $x_{j_1}, \ldots, x_{j_k}$ enumerate (possibly with repetitions) all elements of M.

Finally, for each clause

$$c = a_1 \vee \ldots \vee a_s \vee \neg b_1 \vee \ldots \vee \neg b_t$$

from C define a logic program clause $p(c)$:

$$p(c) = f \leftarrow b_1, \ldots, b_t, \mathbf{not}(a_1), \ldots, \mathbf{not}(a_s), \mathbf{not}(f)$$

where f is yet another new atom. Define $P_2 = \{p(c) \colon c \in C\}$ and $P^C = S \cup P_1 \cup P_2$.

Theorem 5.7 *A set of clauses C has a nonempty model with no more than k elements if and only if the program P^C has a stable model with no more than $2k$ elements.*

Now the reducibility of the problem $WS(k)$ to the problem $SSM(2k)$ is evident. Given a collection of clauses C, to check whether it has a model of size at most k, we first check whether the empty set of atoms is a model of C. If so, we return the answer YES and terminate the algorithm. Otherwise, we construct the program P^C and check whether it has a stable model of size at most $2k$. Consequently, we obtain the following result.

Theorem 5.8 *The problem SSM is* W[2]*-hard.*

6 Open problems and conclusions

There is a natural variation on the problem of computing large stable models: given a logic program P and an integer k (parameter), decide whether P has a stable model of size at least $|At(P)| - k$. We conjecture that this version of the problem LSM is fixed-parameter intractable but have not been able to find a proof, yet.

Another open problem is to resolve whether there is an algorithm for the problem $SSM(k)$ that would run in time $O(n^{\alpha k})$, for some constant $\alpha < 1$.

Finally, our results show that SSM is in $W[3]$ and that it is $W[2]$-hard. Determining the exact location of the problem SSM in the W hierarchy is yet another open problems suggested by our paper.

Acknowledgments

The author thanks Victor Marek and Jennifer Seitzer for useful discussions and comments, and to an anonymous referee for pointing out inaccuracies in proofs of some of the results. This research was supported by the NSF grants IRI-9400568 and IRI-9619233.

References

[1] K. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of deductive databases and logic programming. Papers from the workshop held in Washington, D.C., August 18–22, 1986*, pages 89–142, Palo Alto, CA, 1988. Morgan Kaufmann.

[2] C. Aravindan, J. Dix, and I. Niemelä. DisLoP: Towards a disjunctive logic programming system. In *Logic programming and nonmonotonic reasoning (Dagstuhl, Germany, 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 1997.

[3] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[4] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78(1, (Part B)):85–112, 1991.

[5] A. Bondarenko, F. Toni, and R.A. Kowalski. An assumption-based framework for non-monotonic reasoning. In A. Nerode and L. Pereira, editors, *Logic programming and non-monotonic reasoning (Lisbon, 1993)*, pages 171–189, Cambridge, MA, 1993. MIT Press.

[6] P. Cholewiński, W. Marek, and M. Truszczyński. Default reasoning system deres. In *Proceedings of KR-96*, pages 518–528. Morgan Kaufmann, 1996.

[7] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM J. Comput*, 24:873–921, 1995.

[8] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for w[1]. *Theoretical Computer Science*, 141:109–131, 1995.

[9] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for non-monotonic reasoning. In *Logic programming and nonmonotonic reasoning (Dagstuhl, Germany, 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 1997.

[10] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In R. Kowalski and K. Bowen, editors, *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080, Cambridge, MA, 1988. MIT Press.

[11] G. Gottlob, F. Scarcello, and M. Sideri. Fixed parameter complexity in ai and nonmonotonic reasoning. In *Proceedings of LPNMR'99*, 1999. To appear.

[12] W. Marek and M. Truszczyński. Stable semantics for logic programs and default theories. In E.Lusk and R. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 243–256. MIT Press, 1989.

[13] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[14] W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K.R. Apt, W. Marek, M. Truszczyński, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[15] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In I. Niemelä and T. Schaub, editor, *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79, 1998.

[16] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of JICSLP-96*. MIT Press, 1996.

[17] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.