Extremal problems in logic programming and stable model computation

Paweł Cholewiński and Mirosław Truszczyński
Computer Science Department
University of Kentucky
Lexington, KY 40506-0046
{pawel|mirek}@cs.engr.uky.edu

Abstract

We study the following problem: given a class of (disjunctive) logic programs $\mathcal{C}$, determine the maximum number of stable models (answer sets) of a program from $\mathcal{C}$. We establish the maximum for the class of all logic programs with at most n clauses, and for the class of all logic programs of size at most n. We also characterize the programs for which the maxima are attained. We obtain similar results for the class of all disjunctive logic programs with at most n clauses, each of length at most m, and for the class of all disjunctive logic programs of size at most n. Our results on logic programs have direct implication for the design of algorithms to compute stable models. Several such algorithms, similar in spirit to the Davis-Putnam procedure, are described in the paper.

1 Introduction

In this paper we study extremal problems appearing in the context of finite propositional logic programs. Specifically, we consider the following problem: given a class of logic programs $\mathcal{C}$, determine the maximum number of stable models a program in $\mathcal{C}$ may have. Extremal problems have been studied in other disciplines, especially in combinatorics and graph theory [1]. However, no such results for logic programming have been known so far.

We will consider finite propositional disjunctive logic programs built of clauses of the form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \mathbf{not}(c_1), \ldots, \mathbf{not}(c_n),$$

where a_i, b_i and c_i are atoms. In an effort to establish a semantics for disjunctive logic programming, Gelfond and Lifschitz [5] introduced the notion of an *answer set* of a disjunctive program. It is well-known that for *normal* logic programs (each clause has exactly one atom in the head), answer sets coincide with *stable* models [4, 5]. We will denote the set of answer sets of a disjunctive program P (stable models, if P is normal) by $ST(P)$ and we will set

$$s(P) = |ST(P)|.$$

Given a class $\mathcal{C}$ of disjunctive programs, our goal will be to determine the value of

$$\max\{|ST(P)|: P \in \mathcal{C}\}.$$

We will also study the structure of *extremal* programs in $\mathcal{C}$, that is, those programs in $\mathcal{C}$ for which the maximum is attained.

We will focus our considerations on the following classes of programs:

1. $\mathcal{DP}_{n,m}$ — the class of disjunctive programs with at most n clauses and with the length of each clause bounded by m,
2. $\mathcal{LP}_n$ — the class of normal logic programs with at most n clauses.

We will establish the values

$$s(n) = \max\{|ST(P)|: P \in \mathcal{LP}_n\}$$

and

$$d(n,m) = \max\{|ST(P)|: P \in \mathcal{DP}_{n,m}\}.$$

We will show that $s(n) = \Theta(3^{n/3})$ (an exact formula will be given) and $d(n,m) = m^n$, and we will characterize the corresponding *extremal* programs. We will also study related classes of programs: the class of logic programs of size (understood as the total number of atom occurrences) at most n, the class of logic programs with n clauses and with at most one literal in the body of each clause and, finally, the class of disjunctive logic programs of size at most n.

The motivation for this work comes from several sources. First of all, this work has been motivated by our efforts to develop fast algorithms for computing stable models of logic programs. It turns out that bounding the number of stable models and search for extremal logic programs is intimately connected to some recursive algorithms for computing stable models. Two basic lemmas derived in Section 2 imply both the bounds on the number of stable models, and a whole spectrum of algorithms to compute stable models. The lemmas imply the worst-case bounds on the size of the search space traversed by those algorithms. The algorithms show striking analogies to the Davis-Putnam procedure for testing satisfiability of CNF formulas. One of these algorithms is similar to the algorithm recently described and studied in [7, 8]. There are also some analogies with the algorithms described in [9, 3].

Additional motivation comes from considerations of expressive power of logic programming and of representability issues. Both concepts help understand the scope of applicability of logic programming as a knowledge representation tool. Disjunctive logic programs with answer set semantics (logic programs with stable model semantics) can be viewed as encodings of families of sets, namely, of the families of their stable models. A family of sets $\mathcal{F}$ is *representable* if there is a (disjunctive) logic program P such that

$$ST(P) = \mathcal{F}.$$

Important issues are: (1) to find properties of representable families of sets, (2) given a representable family of sets $\mathcal{F}$, to find possibly concise logic program representations of $\mathcal{F}$. Related problems in default logic have been studied in [6].

It is well-known [5] that every representable family of sets must be an antichain. Our study of extremal problems in logic programming provide additional conditions. Namely, every family of sets representable by a program from $\mathcal{DP}_{n,m}$ must have size bounded by m^n and every family of sets representable by a logic program from $\mathcal{LP}_n$ must have size bounded by $3^{n/3}$. The best bound known previously for families of sets representable by logic programs from $\mathcal{LP}_n$ was $\approx 0.8 \times 2^n/\sqrt{n}$.

In addition, the results of this paper allow some comparison of the expressive power of different classes of programs. For example, there is a disjunctive logic program of size n with $\Theta(2^{n/2})$ answer sets while the largest cardinality of a family of sets representable by a logic program of size n is only $\Theta(2^{n/4})$. This might be interpreted as evidence of stronger expressive power of disjunctive logic programs.

2 Normal logic programs

In this section we study extremal problems for normal (non-disjunctive) logic programs. We will determine the value of the function $s(n)$ and we will provide a characterization of all programs in the class $\mathcal{LP}_n$ which have $s(n)$ stable models. No bounds on the length of a clause are needed in this case. It is well known that each stable model of a program P is a subset of the set of heads of P. Consequently, $s(n) \leq 2^n$. This bound can easily be improved. Stable models of a program form an antichain. Since the size of the largest antichain in the algebra of subsets of an n-element set is $\binom{n}{\lfloor n/2 \rfloor} \approx 0.8 \times 2^n/\sqrt{n}$, clearly, $s(n) \leq 0.8 \times 2^n/\sqrt{n}$. We will still improve on this bound by showing that $s(n) = \Theta(3^{n/3}) \approx \Theta(2^{0.538n}) << 0.8 \times 2^n/\sqrt{n}$. We obtain similar results for the class $\mathcal{LP}_n^2$ of logic programs with n clauses each of which has at most one literal in the body, and for the class $\mathcal{LP}_n'$ of all logic programs with at most n atom occurrences.

Our approach is based on the following versions of the notion of reduct. In the next section, these reducts will be used to design algorithms for computing stable models.

Definition 2.1 *Let P be a logic program and q be an atom which occurs in P. The* positive reduct of P with respect to q, *denoted as $P(q^+)$, is a logic program obtained from P by:*

1. *Removing all clauses with head q.*
2. *Removing all clauses with* **not**(q) *in the body.*
3. *Removing q from the bodies of all remaining clauses.*

The negative reduct of P with respect to q, *denoted by $P(q^-)$, is the logic program obtained from P by:*

1. *Removing all clauses with head q.*
2. *Removing all clauses with q in the body.*
3. *Removing* **not**(q) *from the bodies of all remaining clauses.*

Intuitively speaking, $P(q^+)$ and $P(q^-)$ are the programs implied by P and sufficient to determine all those stable models of P that contain q ($P(q^+)$), and all those stable models of P that do not contain q ($P(q^-)$). Formally, we have the following lemma.

Lemma 2.1 *Let P be a logic program and q be an atom in P. If M is a stable model of P then*

1. *if $q \in M$ then $M \setminus \{q\}$ is a stable model of $P(q^+)$,*
2. *if $q \notin M$ then M is a stable model of $P(q^-)$.*

Observe that the implication in the statement of the lemma cannot be reversed. Due to nonmonotonicity of stable model semantics, not every stable model of the reduct ($P(q^+)$ or $P(q^-)$) gives rise to a stable model of P.

As a corollary, we obtain a recursive bound on the number of stable models in P (recall that $s(P)$ stands for the cardinality of the family of stable models of P).

Corollary 2.2 *For any logic program P and any atom q in P*

$$s(P) \leq s(P(q^+)) + s(P(q^-)). \tag{1}$$

It is also clear that these last two results imply a recursive algorithm to compute stable models. These applications of Lemma 2.1 and Corollary 2.2 will be discussed in the next section.

Similarly, we will define now two reducts implied by P and a clause r from P: $P(r^+)$ and $P(r^-)$.

Definition 2.2 *Let $r = q \leftarrow a_1, \ldots, a_k, \mathbf{not}(b_1), \ldots, \mathbf{not}(b_l)$ be a clause of a logic program P. The* positive reduct of P with respect to r, *denoted as $P(r^+)$, is a logic program obtained from P by:*

1. *Removing all clauses with head in $p \in \{q, a_1, \ldots, a_k, b_1, \ldots, b_l\}$.*
2. *Removing all clauses with at least one of* **not**(q), **not**(a_1), **not**(a_k), $b_1, \ldots, b_l$ *in the body.*
3. *Removing all of $q, a_1, \ldots, a_k,$* **not**$(b_1), \ldots,$ **not**(b_l) *from the bodies of all remaining clauses.*

The negative reduct of P with respect to r, *denoted as $P(r^-)$, is a logic program obtained from P by deleting r, that is $P(r^-) = P \setminus \{r\}$.*

Let us recall that a logic program clause r is *generating* for a set of atoms S if every atom occurring positively in the body of r is in S and every atom occurring negated in r is not in S. Using the concept of a generating clause,

the intuition behind the definitions of $P(r^+)$ and $P(r^-)$ is as follows. The reduct $P(r^+)$ allows us to compute all those stable models of P for which r is a generating clause, while the reduct $P(r^-)$ allows us to compute all those stable models of P for which r is not generating. More formally, we have the following lemma.

Lemma 2.3 *Let $r = q \leftarrow a_1, \ldots, a_k, \mathbf{not}(b_1), \ldots, \mathbf{not}(b_l)$ be a clause of a logic program P. If M is a stable model of P then*

2. if $\{a_1, \ldots, a_k\} \subseteq M$ and $\{b_1, \ldots, b_l\} \cap M = \emptyset$ then $M \setminus \{q, a_1, \ldots, a_k\}$ is a stable model of $P(r^+)$,

2. otherwise M is a stable model of $P(r^-)$.

Also in the case of this lemma, the implication in its statement cannot be replaced by equivalence, due to the nonmonotonic nature of the stable model semantics. That is, not every stable model of the reduct ($P(r^+)$ or $P(r^-)$) gives rise to a stable model of P.

As before, we have a corollary providing a recursive bound on the number of stable models. A corresponding algorithmic implications will be discussed in Section 3.

Corollary 2.4 *For any logic program P and any clause r of P*

$$s(P) \leq s(P(r^+)) + s(P(r^-)). \tag{2}$$

We will now introduce the class of canonical logic programs and determine for them the number of their stable models . We will use canonical programs to characterize extremal logic programs in the class $\mathcal{LP}_n$.

Definition 2.3 *Let $A = \{a_1, a_2, \ldots, a_k\}$ be a set of atoms. By $c(a_i)$ we denote the clause*

$$c(a_i) = a_i \leftarrow \mathbf{not}(a_1), \ldots, \mathbf{not}(a_{i-1}), \mathbf{not}(a_{i+1}), \ldots, \mathbf{not}(a_k).$$

A canonical *logic program over A, denoted by $CP[A]$, is the logic program containing exactly k clauses $c(a_1), \ldots, c(a_k)$, that is*

$$CP[A] = \bigcup_{i=1}^{k} \{c(a_i)\}.$$

Intuitively, the program $CP[A]$ "works" by selecting exactly one atom from A. Formally, $CP[A]$ has exactly k stable models of the form $M_i = \{a_i\}$, for $i = 1, \ldots, k$.

Definition 2.4 *Let P be a logic program and A be the set of atoms which appear in P. Program P is a* $2,3,4$-program *if A can be partitioned into pairwise disjoint sets $A_1, \ldots, A_l$ such that $2 \leq |A_i| \leq 4$ for $i = 1, \ldots, l$, and*

$$P = \bigcup_{i=1}^{l} CP[A_i].$$

Roughly speaking, a $2,3,4$-program is a program which arises as a union of independent canonical programs of sizes 2, 3 or 4. A $2,3,4$-program is stratified in the sense of [2] and the canonical programs are its strata. Stable models of a $2,3,4$-program can be obtained by selecting (arbitrarily) stable models for each stratum independently and, then, forming their unions. By the *signature* of a $2,3,4$-program P we mean the triple $\langle \lambda_2, \lambda_3, \lambda_4 \rangle$, where λ_i, $i = 2,3,4$, is the number of canonical programs over an i-element set appearing in P.

Up to isomorphism, a $2,3,4$-program is uniquely determined by its signature. Other basic properties of $2,3,4$-programs are gathered in the following proposition.

Proposition 2.5 *Let P be a $2,3,4$-program with n clauses and the signature $\langle \lambda_2, \lambda_3, \lambda_4 \rangle$. Then:*

1. $n = 2\lambda_2 + 3\lambda_3 + 4\lambda_4$,
2. $s(P) = 2^{\lambda_2} 3^{\lambda_3} 4^{\lambda_4}$.

As a direct corollary to Proposition 2.5, we obtain a result describing $2,3,4$-programs with n clauses and maximum possible number of stable models. For $k \geq 1$, let us define $A(k)$ to be the unique (up to isomorphism) $2,3,4$-program with the signature $\langle 0, k, 0 \rangle$, and $C(k)$ and $C'(k)$ to be the unique (up to isomorphism) $2,3,4$-programs with the signatures $\langle 2, k-1, 0 \rangle$ and $\langle 0, k-1, 1 \rangle$, respectively. Finally, for $k \geq 0$, let us define $B(k)$ to be the unique (up to isomorphism) $2,3,4$-program with the signature $\langle 1, k, 0 \rangle$.

Corollary 2.6 *Let P be a $2,3,4$-program with n clauses and maximum number of stable models. Then,*

1. *if $n = 3k$ for some $k \geq 1$, $P = A(k)$,*
2. *if $n = 3k+1$ for some $k \geq 1$, $P = C(k)$ or $C'(k)$,*
3. *if $n = 3k+2$ for some $k \geq 0$, $P = B(k)$.*

Consequently, the maximum number of stable models of an $2,3,4$-programs with n clauses is given by

$$s_0(n) = \begin{cases} 3 * 3^{\lfloor n/3 \rfloor - 1} & \text{for } n \equiv 0 \ mod\ 3 \\ 4 * 3^{\lfloor n/3 \rfloor - 1} & \text{for } n \equiv 1 \ mod\ 3 \\ 6 * 3^{\lfloor n/3 \rfloor - 1} & \text{for } n \equiv 2 \ mod\ 3 \end{cases}$$

Corollary 2.6 implies that

$$s(n) \geq s_0(n) = \Theta(3^{n/3}). \tag{3}$$

We will now show that, in fact, $s(n) = s_0(n)$. Moreover, we will also determine the class of all extremal programs. We will call an atom q occurring in P *redundant* if q is not the head of a clause in P.

Let P be a logic program. By $\overline{P}$ we denote the logic program obtained from P by removing all negated occurrences of redundant atoms. Let $\mathcal{E}_n$ be

the class of all programs P such that

1. $\overline{P}$ is $A(k)$, if $n = 3k$ $(k \geq 1)$, or
2. $\overline{P}$ is $B(k)$, if $n = 3k + 2$ $(k \geq 0)$, or
3. $\overline{P}$ is $C(k)$ or $C'(k)$, if $n = 3k + 1$ $(k \geq 1)$.

Theorem 2.7 *If P is an extremal logic program with $n \geq 2$ clauses, then P has $s_0(n)$ stable models. That is, for any $n \geq 2$*

$$s(n) = s_0(n).$$

In addition, the extremal programs in $\mathcal{LP}_n$ are exactly the programs in $\mathcal{E}_n$.

Theorem 2.7 can be proved by induction on n. The next lemma establishes the basis for the induction.

Lemma 2.8 *Let P be an extremal program with n clauses. Then, for some atoms $a_1, \ldots, a_n$:*

1. *if $n = 2$, $\overline{P} = CP[\{a_1, a_2\}]$ $(= B(0))$,*
2. *if $n = 3$, $\overline{P} = CP[\{a_1, a_2, a_3\}]$ $(= A(1))$,*
3. *if $n = 4$, $\overline{P} = CP[\{a_1, a_2, a_3, a_4\}]$ $(= C'(1))$, or $\overline{P} = CP[\{a_1, a_2\}] \cup CP[\{a_3, a_4\}]$ $(= C(1))$.*

The induction step, as well as the main steps of the argument, are provided by the following lemma. Its proof relies on Lemmas 2.1 and 2.3 that establish recursive dependencies between the number of stable models of P and of its reducts. The details will be given in the full version of the paper.

Lemma 2.9 *Let $n \geq 5$. Assume that every extremal program with $n' < n$ clauses, and with no negated occurrences of redundant atoms is a $2,3,4$-program. If P is an extremal program with $n \geq 5$ clauses and no negated occurrences of redundant atoms, then:*

1. *P contains no two clauses with the same head;*
2. *P contains no atoms that appear only positively in P;*
3. *P contains no clauses of the form $q \leftarrow p$;*
4. *P is a $2,3,4$-program;*

Lemmas 2.8 and 2.9 imply that if P is an extremal program for the class $\mathcal{LP}_n$ and if P has no negated occurrences of redundant atoms, then P is a $2,3,4$-program. Consequently, by Corollary 2.6, it follows that P is either $A(k)$, $C(k)$ or $C'(k)$, or $B(k)$, depending on whether $n = 3k$, $3k + 1$ or $3k + 2$. In addition, we have that $s(n) = s_0(n) = \Theta(3^{n/3})$. Hence, Theorem 2.7 follows.

The general bound of Theorem 2.7 can still be slightly improved (lowered) if the class of programs is further restricted. Since there are extremal programs for the whole class $\mathcal{LP}_n$ with no more than 2 literals in the body of each clause, the only reasonable restriction is to limit the number of literal

occurrences in the body to at most 1. The class of programs with n clauses and satisfying this restriction will be denoted by $\mathcal{LP}^2_n$.

Denote by $P(k)$ a $2,3,4$-program with signature $\langle k,0,0\rangle$. Clearly, $P(k) \in \mathcal{LP}^2_n$. We have the following result.

Theorem 2.10 *For every program $P \in \mathcal{LP}^2_n$, $s(P) \leq 2^{\lfloor n/2 \rfloor}$. Moreover, there are programs in $\mathcal{LP}^2_n$ for which this bound is attained. Program $P(k)$ is a unique (up to isomorphism) extremal program with $n = 2k$ clauses, and every extremal program with $n = 2k+1$ clauses can be obtained by adding one more clause to $P(k)$ of one of the following forms: $p \leftarrow a$, $a \leftarrow$, and $a \leftarrow \mathbf{not}(b)$, where p is an arbitrary atom (may or may not occur in $P(k)$), and a and b are atoms not occurring in $P(k)$.*

Next, we will consider the class $\mathcal{LP}'_n$ of all logic programs with the total size (number of literal occurrences in the bodies and heads) at most n. Let $s'(n)$ be defined as the maximum number of stable models for a program in $\mathcal{LP}'_n$. We have the following result.

Theorem 2.11 $s'(n) = \Theta(2^{n/4})$.

Finally, let us observe that every antichain $\mathcal{F}$ of sets of atoms is representable by a logic program.

Theorem 2.12 *For every antichain $\mathcal{F}$ of finite sets there is a logic program P such that $ST(P) = \mathcal{F}$. Moreover, there exists such P with at most $\sum_{B\in\mathcal{F}} |B|$ clauses and total size at most $|\mathcal{F}| \times \sum_{B\in\mathcal{F}} |B|$.*

On one hand this theorem states that logic programs can encode any antichain $\mathcal{F}$. On the other, the encoding that is guaranteed by this result is quite large (in fact, larger than the explicit encoding of $\mathcal{F}$). In the same time, our earlier results show that often substantial compression can be achieved. In particular, there are antichains of the total size of $\Theta(n3^{n/3})$ that can be encoded by logic programs of size $\Theta(n)$. More in-depth understanding of applicability of logic programming as a tool to concisely represent antichains of sets remains an open area of investigation.

3 Applications in stable model computation

In this section we will describe algorithms for computing stable models of logic programs. These algorithms are recursive and are implied by Lemmas 2.1 and 2.3. They select an atom (or a clause, in the case of Lemma 2.3) and compute the corresponding reducts. According to Lemmas 2.1 and 2.3, stable models of P can be reconstructed from stable models of the reducts. However, it is not, in general, the case that every stable model of a reduct implies a stable model of P (see the comments after Lemma 2.3). Therefore, all candidates for stable models for P, that are produced out of the stable

```
STABLE_MODELS_A(P)
Input: a finite logic program P;
Returns: family Q of all stable models of P;

IMPLIED_SET(P, M, P_0);
if (|P_0| = 0) then return {M}
else
    Q := ∅;
    q := SELECT_ATOM(P_0);

    P_1 := P_0(q^+);
    L := STABLE_MODELS_A(P_1);
    for all N ∈ L do if IS_STABLE(P_0, {q} ∪ N)
        then Q := Q ∪ {M ∪ {q} ∪ N};

    P_2 := P_0(q^-);
    L := STABLE_MODELS_A(P_2);
    for all N ∈ L do if IS_STABLE(P_0, N) then Q := Q ∪ {M ∪ N};

    return Q;
```

Figure 1: Algorithm for computing stable models by splitting on atoms.

models of the reduct, must be tested for stability for P. To this end, an auxiliary procedure IS_STABLE is used. Calling IS_STABLE for a set of atoms M and a logic program P returns **true** if M is a stable model of P, and it returns **false**, otherwise.

In our algorithms we use yet another auxiliary procedure IMPLIED_SET. This procedure takes one input parameter, a logic program P, and outputs a set of atoms M and a logic program P_0 (modified P) with the following properties:

1. M is a subset of every stable model of P, and
2. stable models of P are exactly the unions of M and stable models of P_0.

There are several specific choices for the procedure IMPLIED_SET. A trivial option is to return $M = \emptyset$ and $P_0 = P$. Another possibility is to output as M the set of atoms true in the well-founded semantics and as P_0 the residual program for P (see [9, 3]). However, in general, there are many other, intermediate, ways to compute M and P_0 in polynomial time so that conditions (1) and (2) above are satisfied.

We will now describe the algorithms. We adopt the following notation. For a logic program clause r, by $head(r)$ we denote the head of r and by $positivebody(r)$, the set of atoms occurring positively in the body of r.

```
STABLE_MODELS_R(P)
Input: a finite logic program P;
Returns: family Q of all stable models of P;

IMPLIED_SET(P, M, P_0);
if (|P_0| = 0) then return {M}
else
    Q := ∅;
    r := SELECT_CLAUSE(P_0);

    P_1 := P_0(r^+);
    L := STABLE_MODELS_R(P_1);
    for all N ∈ L do if IS_STABLE(P_0, N ∪ positivebody(r) ∪ {head(r)})
        then Q := Q ∪ {M ∪ N ∪ positivebody(r) ∪ {head(r)}};

    P_2 := P_0(r^-);
    L := STABLE_MODELS_R(P_2);
    for all N ∈ L do if IS_STABLE(P_0, N) then Q := Q ∪ {M ∪ N};

    return Q;
```

Figure 2: Algorithm for computing stable models by splitting on clauses.

First we will discuss an algorithm based on splitting the original program (that is, computing the reducts) with respect to a selected atom. The correctness of this method is guaranteed by Lemma 2.1. We call this algorithm STABLE_MODELS_A.

In this algorithm, to compute stable models for an input program P we first simplify it to a program P_0 by executing the procedure IMPLIED_SET. A set of atoms M contained in all stable models of P is also computed. Due to our requirements on the IMPLIED_SET procedure, at this point, to compute all models of P, we need to compute all models of P_0 and expand each by M. To this end, we select an atom occurring in P_0, say q, by calling a procedure SELECT_ATOM. Then, we compute the reducts $P_0(q^+)$ and $P_0(q^-)$. For both reducts we compute their stable models. Each of these stable models gives rise to a set of atoms $\{q\} \cup N$ (in the case of stable models for $P_0(q^+)$) or N (in the case of stable models for $P_0(q^-)$). Each of these sets is a candidate for a stable model for P_0. Calls to the procedure IS_STABLE determine those that are. These sets, expanded by M, are returned as the stable models of P. We present the pseudocode for this algorithm in Figure 1.

The second algorithm, STABLE_MODELS_R, is similar. It is based on Lemma 2.3. That is, instead of trying to find stable models of P among the sets of atoms implied by the stable models of $P(q^+)$ and $P(q^-)$, we search

for stable models of P using stable models of $P(r^+)$ and $P(r^-)$, where r is a *clause* of P. The correctness of this approach follows by Lemma 2.3. The pseudocode is given in Figure 2.

Algorithms STABLE_MODELS_A and STABLE_MODELS_R can be merged together into a hybrid method, which we call STABLE_MODELS_H (Figure 3). Here, in each recursive call to STABLE_MODELS_H we start by deciding whether the splitting (reduct computation) will be performed with respect to an atom or to a clause. This is done by invoking the function SELECT_MODE("atom","clause"). Depending on the outcome, the algorithm follows the approach of either STABLE_MODELS_A or STABLE_MODELS_R. That is, either an atom or a clause is selected, the corresponding reducts are computed and recursive calls to STABLE_MODELS_H are made.

All three algorithms provide a convenient framework for experimentation with different heuristics for pruning the search space of all subsets of the set of atoms. In general, the performance of these algorithms depends heavily on how the selection routines SELECT_ATOM, SELECT_CLAUSE and SELECT_MODE are implemented. Although any selection strategy yields a correct algorithm, some approaches are more efficient than others. In particular, the proof of Theorem 2.7 implies selecting techniques for the algorithm STABLE_MODELS_H guaranteeing that the algorihm terminates after the total of at most $O(3^{n/3})$ recursive calls.

Let us also observe that the recursive dependencies given in Lemmas 2.1 and 2.3 indicate that in order to keep the search space (number of recursive calls) small, selection heuristics should attempt to keep the total size of $P(q^+) \cup P(q^-)$ or $P(r^+) \cup P(r^-)$ as small as possible.

The presented algorithms compute all stable models for the input program P. They can be easily modified to handle other tasks associated with logic programming. That is, they can be tailored to compute one stable model, determine whether stable model for P exists, as well as answer whether an atom is true or false in all stable models of P (cautious reasoning), or in one model of P (brave reasoning). All these tasks can be accomplished by adding a suitable stop function and by halting the algorithm as soon as the query can be answered.

The general structure of our algorithms is similar to well-known Davis-Putnam method for satisfiability problem. The IMPLIED_SET procedure corresponds to the, so called, unit-propagation phase of Davis-Putnam algorithm. In this phase necessary and easy-to-compute conclusions of the current state are drawn to reduce the search space. If the answer is still unknown then a guess is needed and two recursive calls are performed to try both possibilities. But there are also differences. First, in our case, splitting can also be done with respect to a clause. The second difference is due to nonmonotonicity of stable semantics for logic programs. When a recursive call in Davis-Putnam procedure returns an answer, this answer is guaranteed to be correct. There is no such guarantee in the case of stable models. Each answer (stable model) returned by a recursive call in our algorithms must

```
STABLE_MODELS_H(P)
Input: a finite logic program P;
Returns: family Q of all stable models of P;

IMPLIED_SET(P, M, P_0);
if (|P_0| = 0) then return {M}
else
    Q := ∅;
    split_mode := SELECT_MODE("atom","clause");

    if (split_mode = "atom") then
       begin
          q := SELECT_ATOM(P_0);
          P_1 := P_0(q^+);
          L := STABLE_MODELS_H(P_1);
          for all N ∈ L do if IS_STABLE(P_0, {q}∪N) then Q := Q∪{M∪{q}∪N};
          P_2 := P_0(q^-);
          L := STABLE_MODELS_H(P_2);
          for all N ∈ L do if IS_STABLE(P_0, N) then Q := Q ∪ {M ∪ N};
       end
    else      (* split_mode = "clause" *)
       begin
          r := SELECT_CLAUSE(P_0);
          P_1 := P_0(r^+);
          L := STABLE_MODELS_H(P_1);
          for all N ∈ L do if IS_STABLE(P_0, N ∪ positivebody(r) ∪ {head(r)})
             then Q := Q ∪ {M ∪ N ∪ positivebody(r) ∪ {head(r)}};
          P_2 := P_0(r^-);
          L := STABLE_MODELS_H(P_2);
          for all N ∈ L do if IS_STABLE(P_0, N) then Q := Q ∪ {M ∪ N};
       end
    return Q;
```

Figure 3: Hybrid algorithm for computing stable models.

be additionally tested (by IS_STABLE procedure) to see whether it is a stable model for the original program.

4 Disjunctive logic programs

In this section, we will focus on the class of disjunctive logic programs $\mathcal{DP}_{n,m}$. For a set of atoms $\{a_1, \ldots, a_m\}$, let us denote by $d(a_1, \ldots, a_m)$ the disjunctive clause of the form

$$a_1 \vee \ldots \vee a_k \leftarrow .$$

By $D(n, m)$, we will denote the disjunctive logic program consisting of n clauses:

$$d(a_{1,1}, \ldots, a_{1,m})$$

...

$$d(a_{n,1}, \ldots, a_{n,m}),$$

with all atoms $a_{i,j}$ — distinct. It is clear that every set of the form

$$\{a_{i,j_i} : i = 1, \ldots, n,\ 1 \leq i_j \leq m\}$$

is an answer set for $D(n,m)$, and that all answer sets for $D(n,m)$ are of this form. Hence,

$$|ST(D(n,m))| = m^n.$$

Consequently, general upper bounds on the number of answer sets for disjunctive programs in such classes that allow clauses of arbitrary length do not exist.

Turning attention to the class $\mathcal{DP}(n,m)$, it is now clear that, since $D(n,m) \in \mathcal{DP}(n,m)$,

$$d(n,m) \geq m^n.$$

The main result of this section shows that, in fact,

$$d(n,m) = m^n$$

and the program $D(n,m)$ is the only (up to isomorphism) extremal program in this class.

Consider a clause d of the form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \mathbf{not}(c_1), \ldots, \mathbf{not}(c_n).$$

By d^+ we will denote the clause obtained from d by moving all negated atoms to the head. That is, d^+ is of the form:

$$a_1 \vee \ldots \vee a_k \vee c_1 \vee \ldots \vee c_n \leftarrow b_1, \ldots, b_m.$$

Let D be a disjunctive program. Define

$$D^+ = \{d^+ : d \in D\}.$$

Lemma 4.1 *For every disjunctive logic program D, $ST(D) \subseteq ST(D^+)$.*

Lemma 4.1 allows us to restrict our search for disjunctive programs with the largest number of answer sets to those that do not contain negated occurrences of atoms. Let $D \in \mathcal{DP}(n,m)$ be a disjunctive logic program without any occurrences of negated atoms. Let D' be a subset of D consisting of all the clauses with the empty body. Each minimal model for D can be obtained by the following procedure:

1. Pick a minimal model M' of D'
2. Reduce $D \setminus D'$ by removing clauses satisfied by M' as well as atoms in the bodies that are satisfied by M'. Call the resulting program D''.
3. Pick a minimal model M'' of D''.

4. Output $M' \cup M''$ as a minimal model of D.

Clearly, D' and D'' are in $\mathcal{DP}(n', m)$ and $\mathcal{DP}(n'', m)$, respectively, where $|D'| = n'$ and $|D''| = n''$. If $n' < n$ and $n'' < n$ then, by induction, one can show that $|ST(D)| \leq m^n$ and that $DP(n, m)$ is the only (up to isomorphism) program for which the inequality becomes equality. Cases $n' = n$ and $n'' = n$ can be handled directly. The details of the argument will be provided in the full version of the paper. Thus, we have the following theorem.

Theorem 4.2 *For every integers $m \geq 1$ and $n \geq 1$, and for every program $D \in \mathcal{DP}(n, m)$, $|ST(D)| \leq m^n$. Moreover, the program $D(n, m)$ is the only program in the class $\mathcal{DP}(n, m)$ for which the bound of m^n is reached. In particular, $d(n, m) = m^n$.*

Finally, we will consider the class $\mathcal{DP}_n$ of all logic programs with the total size (number of literal occurrences in the bodies and heads) at most n. Let $d'(n)$ be defined as the maximum number of answer sets for a disjunctive program in $\mathcal{DP}_n$. We have the following result.

Theorem 4.3 *For every $n \geq 2$, $d'(n) = \Theta(2^{n/2})$.*

Compared with the estimate from Theorem 2.11 for the function $s'(n)$, the function $d'(n)$ is much larger (it is, roughly the square of $s'(n)$). Consequently, there are antichains representable by disjunctive logic programs with the cardinality of the order of the square of the cardinality of largest antichains representable by logic programs of the same total size. This may be an additional argument for disjunctive logic programs as a knowledge representation mechanism.

5 Conclusions

In this paper, we studied extremal problems appearing in the area of logic programming. Specifically, we were interested in the maximum number of stable models (answer sets) a program (disjunctive program) from a given class may have. We have studied several classes in detail. We determined the maximum number of stable models for logic programs with n clauses, logic programs with n clauses, each of length at most 2, and for logic programs of total size at most n. In some of these cases we also characterized the extremal programs, that is, the programs for which the maxima are attained. Similar results were obtained for disjunctive logic programs. Our results have interesting algorithmic implications. Several algorithms, having a flavor of Davis-Putnam procedure, for computing stable model semantics are presented in the paper.

Extremal problems for logic programming have not been studied so far. This paper shows that they deserve more attention. They are interesting in their own right and have interesting computational and knowledge representation applications.

References

[1] B. Bollobás. *Extremal Graph Theory.* Academic Press, 1978.

[2] P. Cholewiński. Reasoning with stratified default theories. In *Proceedings of LPNMR'95*, Lecture Notes in Computer Science 928, pages 273–286, Berlin, 1995. Springer-Verlag.

[3] W. Chen, T. Swift, and D.S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24:161–200, 1994.

[4] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In *Proceedings of the 5th international symposium on logic programming*, pages 1070–1080, Cambridge, MA, 1988. MIT Press.

[5] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[6] W. Marek, J. Treur and M. Truszczyński. Representability by default theories. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics*, pages 105–108, 1996.

[7] I. Niemelä. Towards efficient default reasoning. In *Proceedings of IJCAI-95*, pages 312–318. Morgan Kaufmann, 1995.

[8] I. Niemelä and P. Simmons. Evaluating an algorithm for default reasoning. In *Proceedings of the IJCAI-95 Workshop on Applications and Implementations of Nonmonotomic Reasonigs Systems*, pages 66-72, 1995.

[9] V.S. Subrahmanian, D. Nau and C. Vago. WFS + branch bound = stable models. *IEEE Transactions on Knowledge and Data Engineering*, 7:362–377, 1995.