

# Computing Stable Models of Logic Programs Using Metropolis Algorithm

Alex Brik and Jeffrey B. Remmel

Department of Mathematics, University of California, San Diego, CA 92093-0112

**Abstract.** This article introduces a novel Monte Carlo type algorithm, which we call the Metropolized Forward Chaining (MFC) algorithm, to find a stable model of a general propositional logic program  $P$  if  $P$  has a stable model or to find a maximal subprogram  $P'$  of  $P$  and stable model  $M'$  of  $P'$  if  $P$  does not have a stable model. Our algorithm combines the forward chaining algorithm of Marek, Nerode, and Remmel with the Metropolis algorithm. To demonstrate the feasibility of MFC, we conducted computer experiments on randomly generated logic programs and on logic programs to find (2,6) van der Waerden's certificates.

## 1 Introduction

In this paper, we develop a Monte Carlo type algorithm to solve the following two problems.

1. Given a finite propositional logic program  $P$  which has a stable model, find a stable model  $M$  of  $P$ .
2. Given a finite propositional logic program  $P$  which has no stable model, find a maximal program  $P' \subseteq P$  which has a stable model and find a stable model  $M'$  of  $P'$ .

Part of the motivation of this paper was based on the success of the Metropolis algorithm, which was introduced by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller [19], on tasks which are far from the statistical physics applications for which it was originally designed such as applications to cryptography problems, see [4]. Thus we asked whether the Metropolis algorithm could be used to search for stable models of a normal logic program. A key element that is required of any proposed search procedure that can be combined with the Metropolis algorithm is that we must have a simple measure which gives us information about how far a given proposed solution, which fails to be a stable model, is from a stable model. For example, in SAT solvers, one tries to minimize the number of unsatisfied clauses. There are several possibilities for such measures in the search for stable models which we will discuss in more detail at the end of the paper. However, there was one search procedure introduced by Marek, Nerode, and Remmel [18] called forward chaining where there is an unambiguous measure. Thus we combined forward chaining with the Metropolis Algorithm to

produce an algorithm which we call the Metropolized Forward Chaining (MFC) algorithm that accomplishes (1) and (2) described above. Nevertheless, we think that it is important that other search procedures and other measures that can be combined with the Metropolis algorithm be explored. Thus our Metropolized Forward Chaining algorithm should be viewed as just one possible candidate for the use of Metropolis type algorithms in Answer Set Programming (ASP). Also, there are other stochastic search procedures for stable models such as *Pbmodels* due to Liu and Truszczyński [13–15] which uses pseudo-Boolean constraint solvers to compute stable models of logic programs with weight constraints.

Marek, Nerode and Remmel [18] developed a forward chaining algorithm to solve both problems (1) and (2). The basic idea of the forward chaining algorithm was to start with a normal logic program  $P$  and divide  $P$  into its monotonic part  $mon(P)$ , consisting of the set of Horn clauses in  $P$ , and its non-monotonic part  $nmon(P)$ , consisting of those clauses in  $P$  which contain negations in its body. Then for any well ordering  $\prec$  of  $nmon(P)$ , the forward chaining algorithm produces a set of clauses  $A^\prec$  such that  $mon(P) \subseteq A^\prec \subseteq P$  and a stable model  $D^\prec$  of  $A^\prec$ . Marek, Nerode, and Remmel [18] showed that for each stable model  $M$  of  $P$ , there is a well ordering  $\prec_M$  of  $nmon(P)$  such that  $A^{\prec_M} = P$  and  $D^{\prec_M} = M$ . Thus every stable model can be computed by the forward chaining algorithm relative to a suitable well ordering of the nonmonotonic clauses in  $P$ . Hence, in the search for a stable model of  $P$ , we would like to minimize  $|P - A^{\prec_M}|$  so that  $|P - A^{\prec_M}|$  provides us with a natural measure of how far we are from a stable model. In the case where  $P$  has no stable models, it can be shown that an  $A^{\prec_M}$  which minimizes  $|P - A^{\prec_M}|$  is a maximal size subprogram of  $P$  which does have a stable model. We should note that for a normal logic program  $P$  which does not have a stable model there are other approaches to finding a maximal subprogram  $P'$  which possesses a stable model, see [20] and [1].

The Metropolis algorithm is a widely applicable procedure for drawing samples from a specified distribution on large finite set. That is, let  $X$  be a finite set and  $\pi(x)$  be a probability distribution on  $X$ . The algorithm requires that one specifies a connected, aperiodic Markov chain  $K(x, y)$  on  $X$  called the proposal chain.  $K(x, y)$  need not be symmetric but it must be the case that  $K(x, y) > 0$  if and only if  $K(y, x) > 0$ . The chain  $K$  is then modified by an auxiliary coin tossing procedure to create a new chain  $M$  with stationary distribution  $\pi$ . That is, if the chain is currently at  $x$ , one chooses  $y$  from  $K(x, y)$ . Then one defines the acceptance ratio by

$$G(x, y) = \frac{\pi(y)K(y, x)}{\pi(x)K(x, y)} \quad (1)$$

If  $G(x, y) \geq 1$ , then the chain moves to  $y$ . If  $G(x, y) < 1$ , then we flip a coin with probability of heads equal to  $G(x, y)$ . If the coin comes up heads, then we move to  $y$  and, if the coin comes up tails, we stay at  $x$ . See the article by Diaconis and Saloff-Coste [5] for a survey about what is known about the Metropolis algorithm.

By combining forward chaining with a Metropolis algorithm sampling, we develop a new Monte Carlo type algorithm to compute stable models. In our

case, the space  $X$  will be set of permutations  $\sigma$  of  $nmon(P)$  and our choice of  $G(\sigma, \tau)$  will favor those  $\sigma$  such that  $|P - A^{\prec\sigma}|$  is small. Here and below  $\prec_\sigma$  is a well ordering of  $nmon(P)$  induced by  $\sigma$ . The formal definition of  $A^{\prec}$  will be given in the section 2. Informally  $A^{\prec}$  is the maximal consistent subprogram corresponding to  $\prec$ . Thus, any run of the MFC algorithm will tend to move toward  $\sigma$  such that  $|P - A^{\prec\sigma}|$  is small. When  $|P - A^{\prec\sigma}| = 0$ , we will have found a stable model. Even when  $P$  has no stable models, we will move toward  $\sigma$ 's where the program  $A^{\prec\sigma}$ , which always has a stable model, is as large as possible.

Before proceeding, we pause to make some comments on why we picked the forward chaining procedure and why we picked the Metropolis algorithm to search for stable models.

The choice of the Metropolis algorithm as the basis for MFC is motivated to a large degree by the fact that the Metropolis algorithm has been studied by various researchers for over 50 years, and some of its properties are well known. Thus, for instance we know that with the Metropolis algorithm the state space will eventually be sampled with a distribution close to a specified target distribution. Its sophisticated extensions such as SAMC [10] have its own proven properties. In addition, there already exist literature on the analysis of the applications of the Metropolis algorithm, see a survey paper [5]. Nevertheless, the analysis is still difficult and we will not attempt to carry out such an analysis in this paper. In the future, we hope to be able to analyze MFC at least for some special cases.

MFC is easily parallelized, as each processor can perform its own work independently from the other processors. Thus viewing runs on different processors as independent experiments (which is a reasonable assumption) leads to the following simple calculation: if the probability of failure in  $k$  iterations is  $p$  for 1 processor, then the probability of failure for  $s$  processors in  $k$  iterations (each) is  $p^s$ . Even if  $p$  is close to 1.0, for large  $s$ ,  $p^s$  is small.

A major drawback of the Metropolis algorithm is that the constructed Markov chain can mix very slowly and may be trapped indefinitely in a local mode, rendering the method ineffective, see [12]. However, more sophisticated algorithms such as Dynamic Weighting [12] or SAMC [10] which extend the Metropolis algorithm have been shown to overcome this problem in many cases.

We picked the forward chaining algorithm for several reasons. First, it has the ability to not only search for stable models but also find maximal subprograms which have stable models of programs that do not have stable models. Both require that we attempt to minimize  $|P - A^{\prec}|$  which is a target readily adaptable to the Metropolis algorithm. Second, the Metropolis algorithm requires a proposal chain  $K(x, y)$  and a target distribution  $\pi(x)$  that satisfy certain properties. Our choice of the proposal chains and the target distributions that we use in this paper was dictated by the following three considerations.

- (i) The need to satisfy the necessary constraints on the proposal chain required by the Metropolis algorithm.
- (ii) The ease of implementation.

(iii) The possibility of analyzing the rate of convergence.

In the case of MFC algorithm, one can easily prove that the proposal chains and the target distributions we use satisfy the conditions required by the Metropolis algorithm. Finally, with the forward chaining algorithm, the target distributions for the Metropolis algorithm is defined on the symmetric group, the group of all permutations of given finite set. This allows us to use distributions that are similar to ones that have been used in [4] and [5]. In particular, example of section 3.1 in [4] bears a resemblance to the proposal chains and the target distributions in this paper. Thus our choice of the state space, the proposal chains, and the target distributions was in part motivated by their similarity to this example which should help us to use existing techniques to analyze the rate of convergence of the algorithm.

Nevertheless, we should point out that there are some obvious drawbacks of the forward chaining algorithm which does not make it the best choice for all programs. The first is that to find a stable model of a finite program  $P$ , one might have to examine  $|nmon(P)|!$  orderings or permutations of  $nmon(P)$ . For certain classes of normal logic programs, examining all such orderings is clearly inefficient. For example, if one is writing a program to search for a set  $S$  of some fixed set  $T$  which has a certain property  $Q$ , one might include atoms  $t$  and  $\bar{t}$  for each  $t \in T$  and add clauses of the form

1.  $t \leftarrow \neg \bar{t}$  for all  $t \in T$ ,
2.  $\bar{t} \leftarrow \neg t$  for all  $t \in T$ , and
3.  $a \leftarrow t, \bar{t}, \neg a$  where  $a$  is some new atom.

In this case, the idea is that a stable model  $M$  specifies the set  $S_M$  by declaring  $S_M = \{t \in T : t \in M\}$ . Thus the clauses (1), (2), and (3) are designed to ensure that for a stable model  $M$ , either  $t \in M$  or  $\bar{t} \in M$  but not both  $t, \bar{t} \in M$ . In such a situation, the order of clauses of type (1) and (2) may be not relevant so that we do not want to explore all  $(3|T|)!$  orderings of such clauses. One would rather just search the  $2^{|T|}$  subsets. One can get around this problem by considering a limited class of orderings of  $nmon(P)$ . Thus studying restrictions of the search space of the MFC algorithm is clearly an important area for further research. For example, restrictions of the set of permutations have been suggested by Cholewinski and Truszczyński [2] for the Select-ordering-and-check method, which for the prerequisite-free programs is essentially Forward-Chaining. However the implementation of MFC on a restricted state space requires additional research to prove that it satisfies the restrictions imposed by the Metropolis algorithm which is beyond the scope of this introductory paper. Thus we shall not pursue such issues in the paper.

The outline of this paper is as follows. In section 2, we shall describe the forward chaining algorithm. In section 3, we shall describe Metropolized Forward Chaining (MFC) - our Metropolis type algorithm which will depend on several parameters. In section 4, we shall present some results of computer experiments on how the efficiency of the MFC algorithm depends on the choice of parameters. In general, there are relatively few exact results on the performance of Metropolis

type algorithms and hence such algorithms are tuned by modifying the choice of  $K(x, y)$  and  $\pi(x)$ . Thus, in section 5, we shall discuss some possible modifications of  $K(x, y)$  and  $\pi(x)$  that should be explored further as well as other possible approaches to use Metropolis type algorithms to find stable models.

## 2 The Forward Chaining Algorithm.

In this section, we shall describe the forward chaining algorithm of Marek, Nerode, and Remmel [18] for normal propositional logic programs. A *normal propositional logic program*  $P$  consists of clauses of the form

$$C = a \leftarrow a_1, \dots, a_m, \neg b_1, \dots, \neg b_n \quad (2)$$

where  $a, a_1, \dots, a_m, b_1, \dots, b_n$  are atoms. Here  $a_1, \dots, a_m$  are called the *premises* of clause  $C$ ,  $b_1, \dots, b_n$  are called the *constraints* of clause  $C$ , and  $a$  is called the *conclusion* of clause  $C$ . For any clause  $C$  as in (2), we shall write  $prem(C) = \{a_1, \dots, a_m\}$ ,  $cons(C) = \{b_1, \dots, b_n\}$ , and  $c(C) = a$ . Either  $prem(C)$ ,  $cons(C)$ , or both may be empty.  $C$  is said to be a Horn clause if  $cons(C)$  is empty. We let  $mon(P)$  denote the set of all Horn clauses of  $P$  and  $nmon(P) = P \setminus mon(P)$ . The elements of  $nmon(P)$  will be called *nonmonotonic* clauses. Let  $H(P)$  denote the Herbrand base of  $P$ . A subset  $M \subseteq H(P)$  is called a **model** of a clause  $C$  if whenever  $prem(C) \subseteq M$  and  $cons(C) \cap M = \emptyset$  of  $C$ , then  $c(C) \in M$ .  $M$  is a model of a program  $P$  if it is a model of every clause  $C \in P$ . The reduct of  $P$  with respect to  $M$  denoted  $P^M$  is obtained by removing every clause  $C$  such that  $cons(C) \cap M \neq \emptyset$  and then removing the constraints from all the remaining clauses.  $M$  is called a **stable model** of  $P$  if  $M$  is the least model of  $P^M$ .

The forward chaining algorithm can be applied to any normal propositional logic program  $P$  of arbitrary cardinality. Given  $P$  and a well-ordering  $\prec$  of  $nmon(P)$ , the forward chaining algorithm outputs a subset  $D^\prec$  of  $H(P)$ .  $D^\prec$  will not always be a stable model of  $P$ , but it will be a stable model of certain subprogram  $A^\prec$  of  $P$  which will be computed by the forward chaining algorithm.  $A^\prec$  will be a maximal set of clauses for which  $D^\prec$  is a stable model.

Before we can define the forward chaining algorithm, we need to define the monotonic closure operator  $cl_{mon}$  associated with  $P$ . For any set  $S \subseteq H(P)$ , we define the one step provability operator relative to  $mon(P)$  by setting  $T_{mon(P)}(S)$  to be the union of the set  $S$  and the set of all  $a \in H(P)$  such that there exists a clause  $C = a \leftarrow a_1, \dots, a_m \in mon(P)$  where  $a_1, \dots, a_m \in S$ . Then, we define the monotonic closure of  $S$  relative of  $mon(P)$ ,  $cl_{mon}(S)$ , by  $cl_{mon}(S) = T_{mon(P)} \uparrow \omega(S)$ .

### Forward Chaining Algorithm.

Let  $P$  be a normal propositional logic program and let  $\prec$  be a well-ordering of  $nmon(P)$ . We define two sequences  $\langle D_\xi \rangle_{\xi \in \alpha^+}$  and  $\langle R_\xi \rangle_{\xi \in \alpha^+}$  of subsets of  $H(P)$  where  $\alpha^+$  is the least cardinal greater than the ordinal  $\alpha$  determined by the well ordering  $\prec$ . The set  $D_\xi$  is the set of atoms *derived* by stage  $\xi$  and  $R_\xi$  is

the set of atoms *rejected* by the stage  $\xi$ . We say that a clause  $C$  is *applicable* at stage  $\xi + 1$  if (i)  $\text{prem}(C) \subseteq D_\xi^\prec$ , (ii)  $(\{c(C)\} \cup \text{cons}(C)) \cap D_\xi^\prec = \emptyset$ , and (iii)  $\text{cl}_{\text{mon}}(D_\xi^\prec \cup \{c(C)\}) \cap (\text{cons}(C) \cup R_\xi^\prec) = \emptyset$ .

1. At stage 0, let  $D_0^\prec = \text{cl}_{\text{mon}}(\emptyset)$ ,  $R_0^\prec = \emptyset$ .
2. At stage  $\beta + 1$ , look for an applicable clause at stage  $\beta + 1$ . If there is no applicable clause at stage  $\beta + 1$ , then we set  $D_{\beta+1}^\prec = D_\beta^\prec$  and  $R_{\beta+1}^\prec = R_\beta^\prec$ . Otherwise we let  $C = C_{\beta+1}$  be the  $\prec$ -first applicable clause at stage  $\beta + 1$  and set  $D_{\beta+1}^\prec = \text{cl}_{\text{mon}}(D_\beta^\prec \cup \{c(C)\})$  and  $R_{\beta+1}^\prec = R_\beta^\prec \cup \text{cons}(C)$ .
3. If  $\xi$  is a limit ordinal, then at stage  $\xi$ , we let  $D_\xi^\prec = \bigcup_{\gamma < \xi} D_\gamma^\prec$  and  $R_\xi^\prec = \bigcup_{\gamma < \xi} R_\gamma^\prec$ .

Let

$$D^\prec = D_{\alpha^+}^\prec = \bigcup_{\gamma < \alpha^+} D_\gamma^\prec \quad \text{and} \quad R^\prec = R_{\alpha^+}^\prec = \bigcup_{\gamma < \alpha^+} R_\gamma^\prec. \quad (3)$$

We say a clause  $C$  is *inconsistent relative to  $P$  and  $\prec$* , if  $\text{prem}(C) \subseteq D^\prec$ ,  $(\{c(C)\} \cup \text{cons}(C)) \cap D^\prec = \emptyset$ , but  $\text{cl}_{\text{mon}}(D^\prec \cup \{c(C)\}) \cap (\text{cons}(C) \cup R^\prec) \neq \emptyset$ . We then let  $I^\prec = \{C : C \text{ is inconsistent}\}$  and  $A^\prec = P \setminus I^\prec$ .

Then Marek, Nerode, and Rimmel [18] proved the following results.

**Theorem 1.** *Let  $P$  be a normal propositional logic program.*

1. *For any well-ordering  $\prec$  of  $\text{nmon}(P)$ ,  $D^\prec$  is a stable model of  $A^\prec$ . Hence if  $I^\prec = \emptyset$ , then  $D^\prec$  is a stable model of  $P$ .*
2. *If  $M$  is a stable model of  $P$ , then there exists a well-ordering  $\prec$  of  $\text{nmon}(P)$  such that  $D^\prec = M$ . In fact, for every well-ordering  $\prec$  such that*

$$\text{NG}(M, P) = \{C \in \text{nmon}(P) : \text{prem}(C) \subseteq M, \text{cons}(C) \cap M = \emptyset\} \quad (4)$$

*forms an initial segment of  $\prec$ ,  $D^\prec = M$ .*

We are interested in only finite programs so that a well ordering of  $\text{nmon}(P)$  is determined by taking a permutation  $\sigma$  of  $\text{nmon}(P)$ .

*Example 1.* Let  $H = \{a, b, c, d, e, f\}$  and let  $P$  consist of the following clauses: (1)  $a \leftarrow$ , (2)  $b \leftarrow c$ , (3)  $c \leftarrow a, \neg d$ , (4)  $d \leftarrow b, \neg c$ , (5)  $e \leftarrow c, \neg f$ , (6)  $f \leftarrow c, \neg e$ . Here,  $\text{mon}(P)$  consists of clauses (1) and (2), whereas  $\text{nmon}(P)$  consists of clauses (3), (4), (5), and (6). Let  $\prec$  be the ordering of  $\text{nmon}(P)$  where (3)  $\prec$  (4)  $\prec$  (5)  $\prec$  (6). Then the construction of sets  $D_n^\prec$  and  $R_n^\prec$  is as follows.

**Stage 0**  $D_0^\prec = \text{cl}_{\text{mon}}(\emptyset) = \{a\}$ ,  $R_0^\prec = \emptyset$ .

**Stage 1**  $C_1 = (3)$ ,  $D_1^\prec = \text{cl}_{\text{mon}}(\{a\} \cup \{c\}) = \{a, b, c\}$ ,  $R_1^\prec = \{d\}$ .

**Stage 2**  $C_2 = (5)$ ,  $D_2^\prec = \{a, b, c, e\}$ , and  $R_2^\prec = \{d, f\}$ .

**Stage 3** At this stage our construction stabilizes.

It is easy to see that  $I^\prec = \emptyset$  so  $D^\prec = D_2^\prec$  is a stable model of  $P$ .

Now, let  $\prec'$  be an ordering of  $\text{nmon}(P)$  where (4)  $\prec'$  (3)  $\prec'$  (6)  $\prec'$  (5). Then one can easily check that  $D^{\prec'} = \{a, b, c, f\}$ ,  $R^{\prec'} = \{d, e\}$ , and  $I^{\prec'} = \emptyset$  so  $D^{\prec'}$  is a stable model of  $P$ . These are the only stable models of  $P$  and one can check that the forward chaining construction will produce one of these two stable models for any well-ordering of  $\text{nmon}(P)$ .  $\square$

### 3 The Metropolized Forward Chaining Algorithm

In this section, we shall formally define our MFC algorithm. First, we shall briefly describe the basics of the Metropolis algorithm. Our presentation is based on the description of Markov Chains and the Metropolis algorithm found in [4], [9], and [11]. We shall write  $P(T)$  for the probability of  $T$  and  $P(A|B)$  for the conditional probability of  $A$  given  $B$ .

A sequence of random variables  $x_0, x_1, x_2, \dots$  defined on a finite state space  $X$  is called a Markov chain if it satisfies the Markov property:

$$(\forall t \geq 0)(P(x_{t+1} = y | x_t = x, \dots, x_0 = z) = P(x_{t+1} = y | x_t = x)). \quad (5)$$

Therefore,  $P(x_{t+1} = y | x_t = x) = P(x_{s+1} = y | x_s = x)$  for all  $t, s \geq 0$ . Hence we can record such probabilities as a transition function

$$M(x, y) = P(x_1 = y | x_0 = x). \quad (6)$$

It then follows that for all  $n \geq 1$ ,  $P(x_n = y | x_0 = x) = M^n(x, y)$ . We refer to the outcomes  $x_0 = x, x_1 = y, x_2 = z, \dots$  as a run of the chain starting at  $x$ .

A *probability distribution* on  $X$  is a function  $\pi : X \rightarrow [0, 1]$  such that  $\sum_{x \in X} \pi(x) = 1$ . We say that  $\pi$  is a *stationary distribution* for  $M$  if for all  $x \in X$ ,

$$\sum_{y \in X} \pi(y) M(y, x) = \pi(x).$$

If  $\forall x, y \in X \exists n_0 \forall n \geq n_0 (M^n(x, y) > 0)$ , then we say that the Markov chain  $M$  is *connected* or *irreducible*. A *period* of state  $x \in X$  written  $d(x)$  is the greatest common divisor of all integers  $n \geq 1$  for which  $M^n(x, x) > 0$ . (If  $M^n(x, x) = 0$  for all  $n \geq 1$ , define  $d(x) = 0$ ). A Markov chain in which each state has period one is called *aperiodic*.

**Theorem 2.** (*The Fundamental Theorem of Markov Chains*) *Let  $X$  be a finite set and  $M(x, y)$  be an irreducible, aperiodic Markov chain. Then  $M$  has a unique stationary distribution  $\pi = \pi M$  and for all  $x, y \in X$ ,  $\lim_{n \rightarrow \infty} M^n(x, y) = \pi(y)$ .*

What is now called the Metropolis algorithm was introduced in [19] and it allows one to sample from a distribution  $\pi(x)$  where a direct sampling is not feasible. The basic idea of the Metropolis algorithm is to simulate a Markov chain in the state space of  $x$  so that its stationary distribution is the target distribution  $\pi(x)$ . Hence for runs of  $n$  steps for large enough  $n$ , the algorithm will generate samples with the probabilities close to those defined by  $\pi(x)$ . In particular, the Metropolis algorithm prescribes a transition rule starting with a symmetric Markov chain  $K(x, y)$ . Later Hastings [6] extended the original Metropolis algorithm where one could relax the symmetric requirement on  $K(x, y)$  to the weaker condition that  $K(x, y) > 0$  iff  $K(y, x) > 0$ .

**Metropolis - Hastings Algorithm.**

Given the Markov chain  $K(x, y)$  and the probability distribution  $\pi(x)$ , the Metropolis-Hastings algorithm defines a new Markov chain  $M(x, y)$  as follows. Let  $G(x, y) = \pi(y) K(y, x) / \pi(x) K(x, y)$ . Then

$$M(x, y) = \begin{cases} K(x, y) & \text{if } G(x, y) \geq 1 \text{ and } y \neq x \\ K(x, y) G(x, y) & \text{if } G(x, y) < 1 \text{ and } y \neq x \\ K(x, y) + \sum_{z:G(x,y)<1} K(x, y) (1 - G(x, z)) & \text{if } y = x \end{cases}.$$

The definition of  $M(x, y)$  is equivalent to the following procedure. Given current state  $x_t$ , draw  $y$  based on the Markov chain  $K(x, y)$  and draw  $U$  from the uniform distribution on  $[0, 1]$ . Then set  $x_{t+1} = y$  if  $U \leq G(x_t, y)$  and set  $x_{t+1} = x_t$  otherwise. The key result about the Metropolis-Hasting algorithm is the following.

**Proposition 1.** *Let  $X$  be a finite set and  $K(x, y)$  be a proposal chain on  $X$  such that  $\forall x, y \in X K(x, y) > 0$  iff  $K(y, x) > 0$ . Let  $\pi(x)$  be a probability distribution on  $X$ . Let  $M(x, y)$  be the Metropolis-Hastings chain as defined above. Then  $M(x, y)$  is an irreducible, aperiodic Markov chain on  $X$  with  $\pi(x) M(x, y) = \pi(y) M(y, x)$  for all  $x, y \in X$ . In particular, for all  $x, y \in X$*

$$\lim_{n \rightarrow \infty} M^n(x, y) = \pi(y). \quad (7)$$

To define the MFC algorithm, fix a finite normal propositional logic program  $P$  and let  $N = |nmonP|$ . Since  $nmon(P)$  is finite, the well orderings of  $nmon(P)$  can be specified by permutations  $\sigma$  of  $nmon(P)$ . So let  $perm(P)$  denote the set of all permutations of  $nmon(P)$ . Thus  $|perm(P)| = N!$ . For any  $\sigma \in perm(P)$ , we let  $D^\sigma$ ,  $R^\sigma$ , and  $I^\sigma$  denote the set of derived atoms, rejected atoms, and inconsistent clauses output by the forward chaining algorithm for  $P$  relative to the well ordering of  $nmon(P)$  given by  $\sigma$ . We let  $r(\sigma) = |I^\sigma|$ . Thus if  $r(\sigma) = 0$ , then  $D^\sigma$  is a stable model of  $P$ . Let  $F_i(P) = \{\sigma \in perm(P) : r(\sigma) = i\}$ .

A subprogram  $P' \subseteq P$  is a *maximal size subprogram* of  $P$  that has a stable model if  $P' = A^\sigma$  for some  $\sigma \in perm(P)$  and  $r(\sigma) = \min_{\tau \in perm(P)} r(\tau)$ .

To apply the Metropolis-Hasting algorithm, we need to specify the state space  $X$ , the Markov chain  $K(x, y)$ , and the sampling distribution  $\pi$ . Our state space for the MFC algorithm will be  $perm(P)$ . Next we fix  $k$  where  $2 \leq k < N$  and specify  $K$  by saying that  $K(\sigma, \tau)$  is the probability that starting with  $\sigma$ , we produce  $\tau$  by picking  $k$  elements  $1 \leq i_1 < \dots < i_k \leq N$  uniformly at random, then picking permutation  $\gamma$  of  $i_1, \dots, i_k$  uniformly at random, and then creating a new permutation by replacing  $\sigma_{i_1}, \dots, \sigma_{i_k}$  in  $\sigma$  by  $\sigma_{\gamma(i_1)}, \dots, \sigma_{\gamma(i_k)}$ . For example, if  $k = 3$ ,  $\sigma = 1\ 4\ 3\ 5\ 2\ 6$ , and we picked  $i_1 = 2$ ,  $i_2 = 4$ , and  $i_3 = 6$  and  $\gamma$  is the permutation  $(6\ 4\ 2)$ , then the new permutation that we would create is  $1\ 6\ 3\ 4\ 2\ 5$ . It is easy to see that for all  $\sigma, \tau \in perm(P)$ ,  $K(\sigma, \tau) = K(\tau, \sigma)$  so that  $K$  specifies a symmetric Markov chain and the acceptance ratio for the Metropolis-Hasting algorithm is just  $G(\sigma, \tau) = \frac{\pi(\sigma)}{\pi(\tau)}$ .

Specifying a reasonable distribution  $\pi$  is more problematic. Ideally, we want a distribution  $\pi$  such that  $\pi(\sigma)$  depends only  $r(\sigma)$  and that the probability of  $F_i(P) = \{\sigma : r(\sigma) = i\}$  is non-zero at least for  $i$  in some initial segment, say  $0 \leq i \leq n$  for some  $n < N$ . We also want to favor those  $\sigma$  where  $r(\sigma)$  is small. We shall assume that there exists  $C_1, C_2, \theta, m$  with  $0 < C_1 \leq C_2, 0 < \theta < 1, m \geq 1$  independent of  $N$  such that

- (a) for  $j = 0, 1, \dots, n, C_1\theta^{-j^m} \leq |F_j(P)| \leq C_2\theta^{-j^m}$  and
- (b) for  $j = n + 1, \dots, N, |F_j(P)| \leq C_2\theta^{-j^m}$ .

This is basically saying that  $|F_j(P)|$  is growing exponentially for  $j = 1, \dots, n$ , and is bounded by exponential for  $j = n + 1, \dots, N$ . For example when  $j = 1$ , we are saying that  $|F_{i+1}(P)| \approx \theta^{-1}|F_i(P)|$  for  $i < n$ . Of course, this is not true for all programs  $P$ . That is, there are programs where  $F_0(P) = N!$  which means that the forward chaining algorithm always produces a stable model no matter what ordering we pick for  $nmon(P)$ . For example, Marek, Nerode, and Remmel [16] described a class of programs called FC-normal programs for which this is true. FC-normal logic programs are a generalization of Reiter's [21] normal default theories. There are also programs  $P$  for which  $F_N(P) = N!$ . Such a program must have the following property. For every clause  $C = a \leftarrow b_1, \dots, b_n, \neg c_1, \dots, \neg c_m$  in  $nmon(P)$ , we have that  $cl_{mon}(a) \cap \{c_1, \dots, c_m\} \neq \emptyset$ . Clearly such programs can have no stable models. Thus our assumptions certainly do not apply to all programs  $P$ . Instead, our goal was to provide a reasonable set of assumptions that will allow us to define a distribution function  $\pi(x)$  where the convergence to orderings  $\sigma$  where  $r(\sigma)$  is small is relatively rapid. There are many other choices of distribution functions which seem reasonable and we shall briefly address this issue in the conclusions of the paper. However, for the moment, we shall assume (a) and (b) and then define a probability distribution  $\pi_N$  on  $perm(P)$  by setting

$$\pi_N(\sigma) = \frac{\theta^{j^m}}{\sum_{s=0}^{N-1} \theta^{s^m} |F_s(P)|}$$

if  $\sigma \in F_j(P)$  where  $0 \leq j \leq N - 1$  and  $\pi_N(\sigma) = 0$  if  $\sigma \in F_N(P)$ .

We can then prove the following.

**Theorem 3.** For  $j = 0, 1, \dots, n, \lim_{N \rightarrow \infty} \pi_N(F_j(P))N \geq \frac{C_1}{C_2} > 0$ .

Moreover, if  $\sigma_i \in F_i$  and  $\sigma_s \in F_s$  where  $0 \leq i, s \leq N - 1$  then  $\frac{\pi_N(\sigma_i)}{\pi_N(\sigma_s)} = \theta^{i^m - s^m}$  so that if  $i \leq s$  then  $\frac{\pi_N(\sigma_i)}{\pi_N(\sigma_s)} \geq 1$ .

This given, at any given step in the MFC algorithm, we do the following.

*Given an ordering  $\sigma^{(t)} = \sigma \in perm(P)$ , we compute  $D^\sigma, R^\sigma$  and  $I^\sigma$  by applying the forward chaining algorithm. We then pick a new ordering  $\tau$  according to the chain  $K(x, y)$ . That is, we pick  $k$  elements of  $\sigma$  uniformly at random and pick a random permutation of those  $k$  elements and let  $\tau$  be the permutation that results by reordering the  $k$  chosen elements of  $\sigma$  according to this random*

permutation. Then we compute  $D^\tau$ ,  $R^\tau$  and  $I^\tau$  by applying the forward chaining algorithm. Then we draw  $U$  from the uniform distribution on  $[0,1]$  and set

$$\sigma^{(t+1)} = \begin{cases} \tau & \text{if } U \leq \frac{\pi_N(\tau)}{\pi_N(\sigma)} \\ \sigma & \text{if otherwise.} \end{cases}$$

Note by Theorem 3, if  $r(\tau) < r(\sigma) \leq N - 1$ , then we will replace  $\sigma$  by  $\tau$  so that we will try to move to orderings which minimize the number of inconsistent rules.

In general, for a fixed sample space  $X$ , there are two items that govern the behavior of the Metropolis algorithm. Namely, the Markov chain  $K(x, y)$  and the probability distribution function  $\pi(x)$ . In our particular case, we have three parameters that govern the behavior of the MFC algorithm. First there is the choice of  $k$  which is the number of elements of  $\sigma \in \text{perm}(P)$  that we pick when we move to the next permutation of  $\text{non}(P)$  which determines the proposal chain  $K(x, y)$ . Second, there are the parameters  $\theta$  and  $m$  which arise in the distribution  $\pi_N[\theta, m](\sigma) = \theta^{r(\sigma)^m} / Z$  where  $Z = \sum_{s=0}^{N-1} \theta^{s^m} |F_s(P)|$ . In the next section, we shall report on some numerical experiments on how varying these parameters effects the performance of the MFC algorithm.

## 4 Numerical Experiments

In this section, we will briefly report on some numerical experiments that we carried out on the MFC algorithm. We considered two types of general logic programs. First we consider a certain class of random programs which are based on four parameters  $\gamma$ ,  $U$ ,  $K$ ,  $S$ . That is, we constructed a random program according the following procedure:

For  $i = 1$  to  $\gamma$ ,

- (1) choose the number of premises  $k_i$  at random from the set  $\{0, 1, 2, \dots, K\}$ ,
- (2) choose the number of constraints  $s_i$  at random from the set  $\{0, 1, 2, \dots, S\}$ ,
- (3) choose  $k_i$  elements  $a_1^i, a_2^i, \dots, a_{k_i}^i$  at random from the Herbrand base  $U$ ,
- (4) choose  $s_i$  elements  $b_1^i, b_2^i, \dots, b_{s_i}^i$  at random from  $U$ ,
- (5) choose  $a^i$  at random from  $U$ , and finally
- (6) add clause  $a^i \leftarrow a_1^i, a_2^i, \dots, a_{k_i}^i, \neg b_1^i, \dots, \neg b_{s_i}^i$  to the random program.

The second set of programs that we consider are designed to find so-called certificates for van der Waerden numbers, see [7]. That is, in 1927 the Dutch mathematician van der Waerden [22] proved that for given numbers  $r$  and  $k$ , there exists a smallest number  $n$  - the van der Waerden number  $W(r, k)$  - such that each set partition of the set  $\{1, 2, \dots, n\}$  into  $r$  parts contains at least one subset with an arithmetic progression of at least length  $k$ . For example, it is known that  $W(2, 3) = 9$ ,  $W(2, 4) = 35$ ,  $W(2, 5) = 178$ , and  $W(2, 6) = 1132$ , see [8]. A way to show that  $W(r, k) > n$  is to find a  $(r, k)$  van der Warden certificate of size  $n$  which is a set partition of  $\{1, 2, \dots, n\}$  into  $r$  parts, none of which contain an arithmetic progression of length  $\geq k$ .

We carried out numerical experiments on the following simple program whose stable models correspond to partitions of  $\{1, \dots, S\}$  into 2 parts that have no arithmetic progression of length 6. The Herbrand base of the program  $P(S, 2, 6) = P$  is the set  $H(P) = \{i, \bar{i} : i = 1, \dots, S\}$ . First for all  $k \in \{1, \dots, S\}$  we add the clauses

$$k \leftarrow \bar{k} \text{ and } \bar{k} \leftarrow \neg k.$$

Then for each arithmetic progression  $k, k+p, k+2p, \dots, k+5p \in \{1, \dots, S\}$ , we add the clauses

$$\begin{aligned} k \leftarrow \neg k, \neg k + p, \neg k + 2p, \neg k + 3p, \neg k + 4p, \neg k + 5p \text{ and} \\ \bar{k} \leftarrow \bar{\neg k}, \bar{\neg k + p}, \bar{\neg k + 2p}, \bar{\neg k + 3p}, \bar{\neg k + 4p}, \bar{\neg k + 5p}. \end{aligned}$$

It is not difficult to show that the stable models  $U$  of  $P$  are determined by a set partition  $(M, \bar{M})$  of  $\{1, \dots, S\}$  where  $U = M \cup \{\bar{k} : k \in \bar{M}\}$  and neither  $M$  nor  $\bar{M}$  contain an arithmetic progression of length 6. In this case  $\text{non}(P(S, 2, 6)) = P(S, 2, 6)$  and one can show that for any given  $\sigma$  in  $\text{perm}(P(S, 2, 6))$ ,  $D^\sigma = M \cup \{\bar{k} : k \in \bar{M}\}$  for some set partition  $(M, \bar{M})$  of  $\{1, \dots, S\}$  and  $r(\sigma) = |I^\sigma|$  is equal to the number of arithmetic progressions of length 6 which appear in either  $M$  or  $\bar{M}$ . We also consider a similar program  $P(S, 2, 4)$  to find (2,4)-van der Waerden certificates.

To demonstrate the feasibility of MFC we have conducted numerical experiments that can be divided into four categories:

1. Finding the stable models of the random logic programs.
2. Finding the stable models of a prerequisite free program for (2,6) van der Waerden certificates of size 150. The experiments in this category are a preliminary study of how parameter choices effect the performance of MFC.
3. Finding the stable models of prerequisite free programs for (2,6) van der Waerden certificates of size 160, 170, 180, 190. The experiments are to study how changes in problem size effect the performance of MFC.
4. Finding maximal size subprograms of the program for (2,4) van der Waerden certificates of size 35 which have stable models. In this case, since we know that  $W(2,4) = 35$ , we know that there are no stable models of the program. However, we found permutations  $\sigma$  such that  $r(\sigma) = 1$  so that there are set partitions  $(M, \bar{M})$  of  $\{1, \dots, 35\}$  for which there is exactly one arithmetic progression of length 4 which appears in one of  $M$  or  $\bar{M}$ . Thus the experiments in this case are to study how parameter changes effect the performance of MFC in finding maximal size subprograms that have stable models.

## Experiments on a Random Program

A random program with 2000 rules,  $U = \{1, 2, \dots, 100\}$ ,  $K = 5$  and  $S = 5$  was generated. The program had 1665 nonmonotonic rules.

To denote the experiments, we will write  $(K(s), \pi_N[\theta, m])$  to indicate that we use permutations of size  $s$  to define  $K(x, y)$  and a distribution  $\pi(\sigma) = \theta^{r(\sigma)^m} / Z$ .

	$\pi [0.01, 1]$	$\pi [0.1, 2]$	$\pi [\frac{2}{N}, 1]$	$\pi [\frac{10}{N}, 1]$	$\pi [\frac{1}{\lfloor \sqrt{N} \rfloor}, 1]$
average	12596	12190	11525	12796	10207
SD	13410	16523	10932	8980	9548
min	1375	1949	1864	963	509
max	59267	56391	39534	37358	32431

**Table 1.** data for the experiments on a random program

In this case, we only ran  $(K(2), \pi_N[\theta, m])$  experiments so that our only choice of parameters was  $\theta$  and  $m$ . For each chosen set of parameters, we ran 20 experiments starting with the same program  $P$  and the same initial ordering of the  $nmon(P)$ . Since the MFC algorithm makes random choices, the number of iterations required before the MFC algorithm finds a stable model will vary from run to run. The results are summarized in table 1 where *average* gives the average number of iterations that the MFC algorithm required to find a stable model in the 20 experiments, *SD* refers the standard deviation for number of iterations that the MFC algorithm required to find a stable model, *min* is the minimum number of iterations required to find a stable model, and *max* is the maximum number of iterations required to find a stable model.

### Experiments on Finding (2,6) van der Waerden Certificates of Size 150

The numerical experiments were conducted in finding the stable models of the program that encodes (2,6) van der Waerden certificates of length 150. The purpose of the experiments was to study how the performance of MFC changes with the variation of the parameters. For each set of the parameters 20 experiments were conducted. The experiments would start from a fixed initial well ordering. The program has  $N = 4649$  nonmonotonic clauses.

The results of the experiments are summarized in the tables 2, 3, 4, 5. Each table lists in its columns values for a parameter  $k$  for the proposal chain  $K(k)$ . Each table lists in its rows the values of the parameter  $n$  corresponding to the probability distribution  $\pi[\frac{1}{n}, 1]$ . Our Java implementation of MFC on 2.50GHz Intel CPU typically performed 350000 iterations per minute in these experiments. The best performance was shown at 195189 iterations with  $k = 25$  and  $n = N/100$ . The data in table 2 show that the optimal values for  $k$  and  $\theta$  are:  $15 < k < 45$  and  $N/140 < \theta < N/20$ .

### Growth of the Number of Iterations with the Increase in Size of a Certificate

The plot 1 demonstrates how the log average number of iterations averaged over the experiments with  $k = 30$  change with the increase in the size of the certificate. The plot of the log average number of iterations averaged over the experiments with  $n = N/100$  exhibits a similar pattern.

	2	5	10	20	25	30	35	40	50	100
N	2260772	846708	612053	298555	282552	315571	403392	443851	445125	2093764
N/10	2753384	790587	349564	239942	233991	263141	304789	324856	279459	2419806
N/40	2322840	502795	319578	245632	321114	249683	206649	312397	305957	1298756
N/100	2621310	859811	340289	224666	195189	269188	302300	288519	287471	1246021
N/140	2731825	1147510	369233	295081	424040	332511	356360	553428	419744	1364054
N/200	6843363	2282997	1306961	808980	584248	837274	803650	824788	1312579	5012982

**Table 2.** average number of iterations for  $(K(k), \pi_N[1/n, 1])$ ,  $N = 4649$ , fixed initial state

	2	5	10	20	25	30	35	40	50	100
N	1998982	976440	662526	244302	344523	218712	423850	484295	472330	2200163
N/10	3354477	1020842	259343	175030	198223	190267	287767	333524	157601	4085229
N/40	1851592	366574	225926	188136	246211	184535	169616	362242	255926	1073170
N/100	2547684	532864	180970	130351	128943	139431	243511	195866	206650	943956
N/140	1719185	990320	310028	168601	282443	287981	250897	444768	297726	750645
N/200	3778056	1146486	913365	750605	581623	780580	529109	610343	1076844	2797124

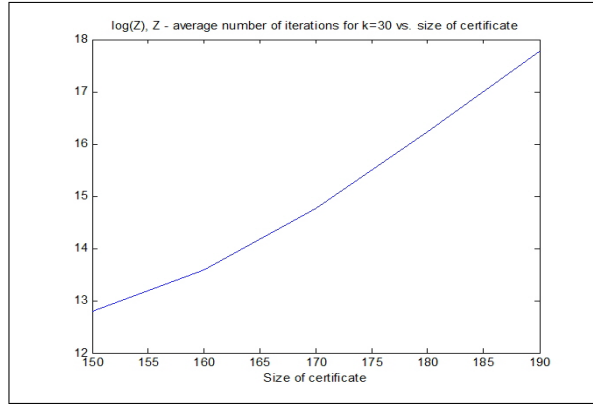
**Table 3.** standard deviations for  $(K(k), \pi_N[1/n, 1])$ ,  $N = 4649$ , fixed initial state

	2	5	10	20	25	30	35	40	50	100
N	506385	118501	50712	49468	37730	34489	41371	56338	45255	616847
N/10	483420	175872	96267	44153	33011	76586	71975	26238	61550	239696
N/20	313633	96272	91111	21375	27958	50516	55463	56545	22680	127951
N/40	351395	94940	39521	63989	73666	34308	25133	41684	40637	125913
N/100	435109	119075	108941	33322	45684	62589	70827	58571	52579	225842
N/140	543911	356067	79106	76935	65592	48675	108874	60732	63907	254902
N/200	1323977	398789	141737	92529	53392	96012	205381	82647	96110	519133

**Table 4.** minimum number of iterations fore  $(K(k), \pi_N[1/n, 1])$ ,  $N = 4649$ , fixed initial state

	2	5	10	20	25	30	35	40	50	100
N	7021692	2097870	2161059	864643	1601663	860841	1560798	1765221	1990824	9915381
N/10	14282858	4459788	1023114	677936	827151	772605	1259603	1462271	550639	18176615
N/20	5198495	2079553	823841	920805	1317666	1416657	1529561	1213912	1131990	2087963
N/40	7481536	1510295	947360	741165	928673	641405	714943	1587899	1037749	4541180
N/100	12051247	2163366	666407	481071	501031	625234	864906	701321	783670	3835183
N/140	7129850	4859652	1439368	690670	1191898	1205546	848044	1773200	1128906	2964538
N/200	15149973	5212479	3409289	3131556	2331566	2985698	1983699	2096518	4132950	10929384

**Table 5.** maximum number of iterations for  $(K(k), \pi_N[1/n, 1])$ ,  $N = 4649$ , fixed initial state



**Fig. 1.**  $\log(Z)$  where  $Z$  is the average number of iterations for  $k=30$

	2	3	5	10	15	20	25	30	35	40	45	50	60	70	100
$N/2$	0.23	0.33	0.25	0.35	0.38	0.25	0.25	0.25	0.25	0.33	0.33	0.33	0.35	0.25	0.3

**Table 6.** fraction of runs that found  $\sigma \in F$  with  $r(\sigma) = 1$  in 1,000,000,000 iterations.

The plot 1 shows the growth in the number of iterations with the increase in size of the experiments that is typical for the experiments that we have conducted when the parameters  $k, \theta$  are fixed and  $m = 1$ .

### Finding Maximal Submodels

Since  $W(2, 4) = 35$ , see [3], there does not exist a  $(2, 4)$  van der Waerden's certificate of size 35. Thus our program  $P(35, 2, 4) = P$  does not have stable models. It turns out that there are permutations  $\sigma$  of  $nmon(P)$  such that  $r(\sigma) = 1$ . In this case, we can show that the corresponding model  $D^\sigma$  is of the form  $M \cup \{\bar{k} : k \in \bar{M}\}$  where  $(M, \bar{M})$  is a set partition of  $\{1, \dots, 35\}$  such that there is exactly one arithmetic progression of length 4 that is contained in  $M$  or  $\bar{M}$ .

For each set of parameters, 40 experiments were conducted where we stopped when we found a  $\sigma$  such that  $r(\sigma) = 1$  or when  $10^9$  iterations were performed, whichever occurred first. The fraction of the number of experiments that ended when  $\sigma$  such that  $r(\sigma) = 1$  was found is shown in the table 6.

## 5 Conclusion

In this paper we have introduced a Monte Carlo type algorithm for solving two problems:

1. Given a finite propositional logic program  $P$  which has a stable model, find a stable model  $M$  of  $P$ .

2. Given a finite propositional logic program  $P$  which has no stable model, find a maximal program  $P' \subseteq P$  which has a stable model and find a stable model  $M'$  of  $P'$ .

To demonstrate the feasibility of MFC, we have conducted numerical experiments on random programs and on the programs defining van der Waerden certificates. The experiments have demonstrated that the MFC algorithm can successfully solve both problems in certain cases.

A major drawback of the Metropolis algorithm is that the constructed Markov chain can mix very slowly and may be trapped indefinitely in a local mode, rendering the method ineffective, see [12]. The numerical experiments that we have conducted led us to observe this drawback effecting the performance of MFC algorithm with certain parameters. In our case, there are a number of other proposals for the chain  $K(x, y)$  which could possibly improve the performance. For example, instead of choosing a subset  $k$  of  $1, \dots, |nmon(P)|$  at random and then choosing a random permutation of those  $k$  elements, one could examine all  $k!$  permutations of the  $k$  chosen elements and then pick the best one, i.e. the ordering  $\tau$  for which  $r(\tau)$  is the smallest. This proposal would lead to a non-symmetric Markov chain and thus we would need the full Metropolis-Hasting algorithm. There are also alternative distributions  $\pi(x)$  that one should explore. For instance one could consider the following probability distribution function as an alternative to the one discussed in the paper: for  $\sigma \in F_j$   $\pi(\sigma) = \frac{(n-j)!}{Z}$  for some well chosen  $n$  with  $0 < n < N$ . There are other ways to modify the algorithm. For example, Liu, Liang, and Wong [12] introduced a modification called Dynamic Weighting and showed that adding Dynamic Weighting improves the rate of convergence of the Metropolis algorithm in many cases. Some limited experiments with SAMC algorithm [10] have shown that when it is used as a basis for MFC instead of the Metropolis algorithm the sampling does not get trapped in a local mode, and the performance of the MFC with SAMC improves compared to MFC performance with the Metropolis algorithm. We plan to explore such alternatives in the future.

There are other ways in which one might apply the Metropolis algorithm to find stable models. For example, instead of using the forward chaining algorithm, one might just consider starting with a subset  $M$  of the Herbrand base and then computing the least model  $M'$  of the Gelfond-Lifschitz transform of  $P$  relative to  $M$ . Then one could compute the cardinality of the symmetric difference of  $M$  and  $M'$ ,  $r(M) = |(M - M') \cup (M' - M)|$ . Then one can move to another subset  $M_1$  by either including new elements in  $M$  or excluding elements from  $M$  or both. The approach, however lacks an important feature of MFC which is that in case that a stable model of a program is not found, a maximal or nearly maximal subprogram can be generated.

However even without considering the modifications of the algorithm itself there are many questions related to the algorithm's performance. In the experiments that we have conducted, we have observed that the change in the parameters  $k$  and  $\theta$  has a significant effect on the performance, and that with the increase in the problem size (i.e. size of the certificate) the change in the optimal

values of the parameters form a trend. More research is needed to understand the exact relationship between the parameters  $k$ ,  $\theta$ ,  $m$  and the performance of MFC.

More research is necessary to conclude how MFC compares with the existing algorithms for finding stable models of logic programs. We have conducted experiments using SMODELS and CLASP solvers to find van der Waerden certificates (with the program  $P(S, 2, 6)$  translated to SMODELS format). For sizes 150, 160, 170, 180, 190, 200 SMODELS clearly outperformed our implementation of MFC. SMODELS found certificates in a small fraction of time that it took MFC to find certificates. However, for size 210, SMODELS failed to find a certificate while running for over 3 weeks, whereas MFC was able to produce certificates of size 210 in 37918180 iterations (under 3 days). The situation is similar with the CLASP solver which found certificates of sizes up to 220 in a small fraction of time that it took MFC to find corresponding certificates. However CLASP failed to find a certificate of size 230 while running for over 2 weeks, whereas MFC was able to produce a size 230 certificate in 251000000 iterations in under 5 days. Of course these experiments are difficult to interpret as they represent a performance comparison for a single problem. Moreover SMODELS and CLASP were run on a single processor machine, whereas MFC was run on a multi-processor cluster.

Regardless of the competitiveness of the MFC algorithm in finding stable models, we feel that its ability to find maximal size subprograms which have stable models when the original program  $P$  does not have stable models will have many applications to planning and negotiations with preferences. That is, it is often impossible to satisfy preferences of everyone. In such a situation, finding maximal subprograms which have stable models can represent a way to satisfy as many preferences as possible. This will be the subject of future work.

**Acknowledgements.** We are grateful to Adriano Garsia for helpful discussion and for providing a summer research assistantship from NSF grant DMS-0800273 for the first author in order to conduct some of the computer experiments.

The second author was partially supported by NSF grant DMS-0654060.

This research was supported in part by NSF MRI Award 0821816 and by other computing resources provided by the Center for Computational Mathematics (<http://ccom.ucsd.edu/>).

## References

1. M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits and S. Woltran. Debugging ASP Programs by Means of ASP, *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, (C. Baral and G. Brewka and J. Schlipf, eds.), LNCS 4483 (2007), 31-43.
2. P. Cholewinski and M. Truszczyński, Minimal number of permutations sufficient to compute all extensions. (<http://www.cs.uky.edu/ai/papers.dir/extns-permutations.pdf>) Unpublished note.

3. V. Chvatal, Some unknown van der Waerden numbers. *Combinatorial Structures and Their Applications* (R. Guy et al., eds.), Gordon and Breach, New York, (1970), pp. 31-33.
4. P. Diaconis, The Markov Chain Monte Carlo Revolution. *Bull. Amer. Math. Soc.*, Nov. (2008).
5. P. Diaconis and L. Saloff-Coste, What do we know about the Metropolis Algorithm?, *J. Comp. and Syst. Science*, **57** (1998), 20-36.
6. W. Hastings, Monte Carlo sampling methods using Markov chains and their applications, *Biometrika*, **57** (1970), 97-109.
7. P.R. Herwig, M. J. H. Heule, P. M. van Lambalgen, and H. van Maaren, A New Method to Construct Lower Bounds for van der Waerden Numbers, *Electronic J. Comb.*, **14** (2007), #R6.
8. M. Kouril and J.L. Paul, The van der Waerden number  $W(2, 6)$  is 1132, *Experimental Mathematics*, **17:1** (2008), 53-61.
9. S. Karlin and H. M. Taylor, A First Course in Stochastic Processes. Second Edition. Academic Press. (1975).
10. F. Liang, C. Liu, and R. J. Carroll, Stochastic Approximation in Monte Carlo Computation. *J. Amer. Statist. Assoc.*, **102** (2007), 305-320.
11. J. S. Liu, Monte Carlo Strategies in Scientific Computing. Springer. (2001).
12. J. S. Liu, F. Liang, and Wing Hung Wong, A Theory for Dynamic Weighting in Monte Carlo Computation. *J. Amer. Statist. Assoc.*, **96** (2001), 561-573.
13. L. Lui and M. Truszczyński, Local-search techniques in propositional logic extended with cardinality atoms, *Proceedings of the 9-th International Conference on Principles and Practice of Constraint Programming (CP-2003)*, (F. Rossi, ed.), LNCS 2833 (2003), 495-509.
14. L. Lui and M. Truszczyński, Local search techniques for Boolean combinations of pseudo-Boolean constraints, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-06)*, AAAI Press (2006), 98-103.
15. L. Lui and M. Truszczyński, Satisfiability testing of Boolean combinations of pseudo-boolean constraints using local search techniques, *Constraints*, **12**(3) (2007), 345-369.
16. W. Marek, A. Nerode, and J. B. Remmel, Context for Belief Revision: FC-Normal Nonmonotonic Rule Systems, *Annals of Pure and Applied Logic*, **67** (1994), pp. 269-324.
17. W. Marek, A. Nerode, and J.B. Remmel, Basic forward chaining construction for logic programs, *Logical Foundations for Computer Science'97, Logic at Yaroslavl*, Lecture Notes in Computer Science 1234, Springer-Verlag (1997) 214-225.
18. W. Marek, A. Nerode, and J.B. Remmel, Logic programs, well orderings, and forward chaining, *Annals of Pure and Applied Logic*, **96** (1999), 231-276.
19. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of State Calculations by Fast Computing Machines, *J. Chem. Phys*, **21** (1953), 1087-1092.
20. E. Oikarinen and M. Jarvisalo, Max-ASP: Maximum Satisfiability of Answer Set Programs, *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009)*, (Esra Erdem, Fangzhen Lin, and Torsten Schaub, eds.), vol. 5753 of Lecture Notes in Artificial Intelligence (2009), 236-249.
21. R. Reiter, A logic for default reasoning, *Artificial Intelligence*, **13** (1980), 81-132.
22. B.L. van der Waerden, Beweis einer Baudetschen Vermutung, *Nieuw Archief voor Wiskunde*, **15** (1927), 212-216.