Computing Stable Models of Logic Programs Using Metropolis Type Algorithms

Alex Brik, Jeffrey B. Remmel

Department of Mathematics, University of California San Diego, USA
La Jolla, CA 92029-0112

Abstract. We study a novel Monte Carlo type algorithm, which we call the Metropolized Forward Chaining (MFC) algorithm, to find a stable model of a normal 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 logic programs to find (2,6) van der Waerden's certificates. The paper also discusses the use of the Stochastic Approximation Monte Carlo (SAMC) algorithm instead of the Metropolis algorithm in MFC.

1 Introduction

The main goal of this paper is to show how one can combine the forward chaining algorithm of Marek, Nerode, and Remmel [19] with the Metropolis Algorithm to produce an algorithm which we call the Metropolized Forward Chaining (MFC) algorithm that will accomplish the following tasks.[1]
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'.

As discussed in [21], finding maximal subprograms that have stable models is important for certain extensions of ASP for programs where arbitrary set constraints are used to model both hard and soft preferences. In such situations, one may not be able to satisfy all soft preferences so that stable models may not exist that satisfy all preferences. However, if we drop certain soft preferences, we may be able to find subprograms that do have stable models.

The Metropolis algorithm is a widely applicable procedure for drawing samples from a specified distribution on a large finite set. The Metropolis algorithm was introduced by Metropolis et. al. [23] in 1953 for applications in statistical

[1] The preliminary but unpublished version of this paper was presented at NonMon@30 conference. The present version of the paper includes additional research on the use of SAMC algorithm.

physics. It was later generalized to the Metropolis-Hastings algorithm [6]. The applications of the Metropolis-Hastings algorithm are widespread in chemistry, physics, biology, statistics, computer science, group theory, cryptography [3] and linguistics [4]. A key element that is required of any proposed search procedure for stable models that can be combined with the Metropolis algorithm is that it must have a simple measure which gives information about how far a given proposed solution, that fails to be a stable model, is from a stable model. 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, in the search procedure introduced by Marek, Nerode, and Remmel [19], called forward chaining algorithm, there is an unambiguous measure.

One potential problem in all applications of the Metropolis algorithm is the so-called local trap problem. That is, it may be the case that after entering a region of high density, the simulation will tend to stay in that region and will be unlikely to move through a region of low density to another region of high density, rendering the method ineffective, see [13]. There are numerous proposals for modifications of the Metropolis algorithm that aim to fix this problem. Liang in [11] lists some of them. We have run computer experiments on a version of MFC where we have replaced the Metropolis algorithm by Stochastic Approximation Monte Carlo (SAMC) algorithm [12], [1] and we will discuss our SAMC version of MFC in section 3. Our computer experiments indicate that as problem's difficulty level increases MFC performs better with SAMC than with the Metropolis algorithm.

In all these variations of the Metropolis algorithm, the user has a choice of how to set various parameters associated with algorithm. 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 input parameters. Thus we will report some results of our computer experiments on certain instances of problems (1) and (2) which indicate the increase in performance that can occur by properly tuning such parameters.

We think that Metropolis type algorithms can provide a useful alternative to standard solvers in the search for stable models. Our methods had some successes in that MFC can find stable models for certain programs where tested versions of leading ASP solvers *smodels* [26] and *clasp* [8] cannot. However, there is still considerable work that needs to be done to understand how best to use Metropolis type algorithms to search for stable models. Thus, we think of this paper as representing an initial attempt in developing applications of Metropolis type algorithms for ASP and it is our hope that this paper will motivate others to research how to refine and improve such applications. We should note that others have used randomized algorithms to search for stable models in ASP. For example, *Pbmodels* due to Liu and Truszczyński [15–17] which uses pseudo-Boolean constraint solvers to compute stable models of logic programs with weight constraints.

The outline of this paper is as follows. In section 2, we shall describe the preliminaries. In section 3, we shall describe Metropolized Forward Chaining

(MFC) - our Metropolis type algorithm. In section 4, we shall present results of preliminary computer experiments on how the efficiency of the MFC algorithm depends on the choice of parameters. In section 5, we shall discuss some possible modifications of input parameters that should be explored further as well as other possible approaches to use Metropolis type algorithms to find stable models.

2 Preliminaries

ASP. Answer Set Programming (ASP) is logic programming with the stable model or answer set semantics. ASP systems are ideal logic-based systems to reason about a variety of types of data and integrate quantitative and qualitative reasoning. Whether a finite propositional logic program has a stable model is NP-complete ([5], [22]). Furthermore, any NP-search problem can be (uniformly) reduced to the problem of finding a stable model of a finite propositional logic program [20].

A *normal propositional logic program* P consists of clauses of the form

$$C = a \leftarrow a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n,$$

where $a, a_1, \ldots, a_m, b_1, \ldots, b_n$ are atoms and $\neg$ is a non-classical negation operator. Here a_1,...,a_n are called the *premises* of clause C, $b_1, \ldots, b_m$ are called the *constraints* of clause C, and a is called the *conclusion* of clause C. For any clause C, we shall write $prem(C) = \{a_1, \ldots, a_n\}$, $cons(C) = \{b_1, \ldots, b_m\}$, 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. 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 a 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.

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 $C = a \leftarrow a_1, \ldots, a_m \in mon(P)$ where

$a_1, \ldots, a_m \in S$. Then, we define the monotonic closure of S relative of $mon(P)$, $cl_{mon}(S)$, by $cl_{mon}(S) = T_{mon(P)}(S) \uparrow \omega = \bigcup_{k \geq 0} T^k_{mon(P)}(S)$.

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 α^+ is the least cardinal greater than the ordinal α determined by the well ordering $\prec$. The set D_ξ is the set of atoms *derived* by stage ξ and R_ξ is the set of atoms *rejected* by the stage ξ. We say that a clause C is *applicable* at stage $\xi + 1$ if (i) $prem(C) \subseteq D^{\prec}_{\xi}$, (ii) $(\{c(C)\} \cup cons(C)) \cap D^{\prec}_{\xi} = \emptyset$, and (iii) $cl_{mon}(D^{\prec}_{\xi} \cup \{c(C)\}) \cap (cons(C) \cup R^{\prec}_{\xi}) = \emptyset$.

1. At stage 0, let $D^{\prec}_0 = cl_{mon}(\emptyset)$, $R^{\prec}_0 = \emptyset$.
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 $D^{\prec}_{\beta+1} = D^{\prec}_{\beta}$ and $R^{\prec}_{\beta+1} = R^{\prec}_{\beta}$. Otherwise we let $C = C_{\beta+1}$ be the $\prec$-first applicable clause at stage $\beta + 1$ and set $D^{\prec}_{\beta+1} = cl_{mon}(D^{\prec}_{\beta} \cup \{c(C)\})$ and $R^{\prec}_{\beta+1} = R^{\prec}_{\beta} \cup cons(C)$.
3. If ξ is a limit ordinal, then at stage ξ, we let $D^{\prec}_{\xi} = \bigcup_{\gamma < \xi} D^{\prec}_{\gamma}$ and $R^{\prec}_{\xi} = \bigcup_{\gamma < \xi} R^{\prec}_{\gamma}$.

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

Then Marek, Nerode, and Remmel [19] proved the following results.

Theorem 1. *Let P be a normal propositional logic program.*
1. For any well-ordering $\prec$ of $nmon(P)$, $D^{\prec}$ 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 $nmon(P)$ such that $D^{\prec} = M$. In fact, for every well-ordering $\prec$ such that $NG(M, P) = \{C \in nmon(P) : prem(C) \subseteq M, cons(C) \cap M = \emptyset\}$ forms an initial segment of $\prec$, $D^{\prec} = M$.

We are interested only in finite programs so that a well ordering of $nmon(P)$ is determined by specifying a permutation σ of $nmon(P)$.

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, $mon(P)$ consists of clauses (1) and (2), whereas $nmon(P)$ consists of clauses (3), (4), (5), and (6).

Let $\prec$ be the ordering of $nmon(P)$ where (3) $\prec$ (4) $\prec$ (5) $\prec$ (6). Then the construction of sets $D^{\prec}_n$ and $R^{\prec}_n$ is as follows.
Stage 0 $D^{\prec}_0 = cl_{mon}(\emptyset) = \{a\}$, $R^{\prec}_0 = \emptyset$.
Stage 1 $C_1 = (3)$, $D^{\prec}_1 = cl_{mon}(\{a\} \cup \{c\}) = \{a, b, c\}$, $R^{\prec}_1 = \{d\}$.
Stage 2 $C_2 = (5)$, $D^{\prec}_2 = \{a, b, c, e\}$, and $R^{\prec}_2 = \{d, f\}$.
Stage 3 At this stage our construction stabilizes.
It is easy to see that $I^{\prec} = \emptyset$ so $D^{\prec} = D^{\prec}_2$ is a stable model of P.

The Metropolis-Hastings Algorithm. The presentation in this section is based on the description of Markov Chains and the Metropolis algorithm found in [3], [9], and [14].

A sequence of random variables $x_0,\ x_1,\ x_2,\ ...$ defined on a finite state space X is called a Markov chain if it satisfies the Markov property: $\forall t \geq 0\ (P\left(x_{t+1} = y \middle|\ x_t = x, ..., x_0 = z\right) = P\left(x_{t+1} = y \middle|\ x_t = x\right))$.

In this paper we will consider only Markov chains with the additional property: $P\left(x_{t+1} = y \middle|\ x_t = x\right) = P\left(x_{s+1} = y \middle|\ x_s = x\right)$ for all $t, s \geq 0$. Hence, we can record such probabilities as a transition function $M\left(x, y\right) = P\left(x_1 = y \middle|\ x_0 = x\right)$. We let $M\left(\cdot,\cdot\right)$ denote the matrix which records this transition function and let $M^n\left(\cdot,\cdot\right)$ denote the n-th power of the matrix $M(\cdot,\cdot)$ for any $n \geq 1$. It then follows that for all $n \geq 1$, $P\left(x_n = y \middle|\ x_0 = x\right) = M^n\left(x, y\right)$.

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

Given a Markov chain $K\left(\cdot,\cdot\right)$, called the proposal chain, and a probability distribution $\pi\left(\cdot\right)$, let $G\left(x, y\right) = \pi\left(y\right) K\left(y, x\right) / \pi\left(x\right) K\left(x, y\right)$. The Metropolis-Hastings algorithm defines a new Markov chain $M\left(\cdot,\cdot\right)$, where the probability $M\left(x, y\right)$ is equal to the probability of drawing $x_{t+1} = y$ given $x_t = x$ using the following procedure:

1. given current state $x_t = x$, draw y based on the Markov chain $K\left(x,\cdot\right)$;
2. draw U from the uniform distribution on $[0, 1]$;
3. set $x_{t+1} = y$ if $U \leq G\left(x_t, y\right)$ 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 $K\left(\cdot,\cdot\right)$ be a proposal chain on X such that $\forall x, y \in X\ K\left(x, y\right) > 0$ iff $K\left(y, x\right) > 0$. Let $\pi\left(\cdot\right)$ be a probability distribution on X. Let $M\left(\cdot,\cdot\right)$ be the Metropolis-Hastings chain as defined above. Then $\pi\left(x\right) M\left(x, y\right) = \pi\left(y\right) M\left(y, x\right)$ for all x, y. In particular, for all x, $y \in X$ $\lim\limits_{n \to \infty} M^n\left(x, y\right) = \pi\left(y\right)$.*

The relevance of this result to the task of finding stable models of normal propositional logic programs is that it implies that after sampling from M for sufficiently many steps, the probability of being at y is $\pi\left(y\right)$ regardless of the starting state. If $\pi\left(y\right)$ is defined to be relatively large whenever y corresponds to a stable model then for a normal propositional logic program P which has a stable model, samples generated from M will eventually include those corresponding to the stable models and moreover the sampling from M will be biased towards those corresponding to the stable models of P.

Stochastic Approximation Monte Carlo (SAMC). SAMC is variant of the Metropolis algorithm designed to help overcome the local trap problem. Let $\pi\left(x\right) = c\psi\left(x\right)$, $x \in X$, denote the target probability distribution, where X is the sample space and c is an unknown constant. We will assume that X is finite, although such an assumption is not made in [11]. Let $E_0, ..., E_d$ denote a partition of X into subregions and let $\omega_i = \sum_{x \in E_i} \psi(x)$. SAMC attempts to draw samples from the distribution $\pi_\omega(x) \propto \sum_{i=0}^{d} \frac{p_i \psi(x)}{\omega_i} I(x \in E_i)$, where $I(\cdot)$ is an indicator function and $\mathbf{p} = (p_0, ..., p_d)$ is the desired sampling distribution of the subregions $E_0, \ldots, E_d$. If $\omega_0, ..., \omega_d$ can be well estimated, then in SAMC, sampling from $\pi_\omega(\cdot)$ will result in a "random walk" in the space of subregions with each subregion being sampled with a frequency proportional to p_i. Hence, the local trap problem can be overcome (provided that the sample space is partitioned appropriately).

Let ξ_{ti} denote the working estimate of $\log(\omega_i/p_i)$ obtained at iteration t, $\xi_t = (\xi_{t0}, ..., \xi_{td})$, and $\{\gamma_t\}$ denote a gain factor sequence such that
(*) $\lim\limits_{t \to \infty} \gamma_t = 0$, $\sum_{t=1}^{\infty} \gamma_t = \infty$, and $\sum\limits_{t=1}^{\infty} \gamma_t^\eta < \infty$ for some $\eta \in (1, 2)$.
Let $J(x)$ denote the index of the subregion to which x belongs. Let $K(\cdot, \cdot)$ be a proposal distribution as in Metropolis algorithm. Then one iteration of SAMC can be described as follows.

1. Simulate a sample x_t by a single update with the target distribution $\pi_{\xi_t}(x) \propto \sum\limits_{i=0}^{d} \frac{\psi(x)}{e^{\xi_{ti}}} I(x \in E_i)$. That is, (a) generate y according to the proposal distribution $K(x_t, y)$, (b) calculate the ratio $r = e^{\left(\xi_{tJ(x_t)} - \xi_{tJ(y)}\right)} \frac{\psi(y)}{\psi(x_t)} \frac{K(y, x_t)}{K(x_t, y)}$, and (c) accept y with probability $\min(1, r)$. If it is accepted, set $x_{t+1} = y$; otherwise, set $x_{t+1} = x_t$.

2. Set $\xi^* = \xi_t + \gamma_{t+1}(e_t - p)$ where $e_t = (e_{t,0}, ..., e_{t,d})$ and $e_{t,i} = 1$ if $x_t \in E_i$ and 0 otherwise. If $\xi^* \in \varXi$, set $\xi_{t+1} = \xi^*$; otherwise, set $\xi_{t+1} = \xi^* + \mathbf{c}^*$, where $\mathbf{c}^* = (c^*, ..., c^*)$ can be an arbitrary vector which satisfies $\xi^* + \mathbf{c}^* \in \varXi$. Here $\varXi$ is a compact region. For example, we can set $\varXi = [-B, B]^d$ for a large B.

In [12] Liang et al. showed that under mild conditions
$\theta_{ti} \to C + \log\left(\sum\limits_{x \in E_i} \psi(x)\right) - \log(p_i + h)$ if $E_i \neq \emptyset$ and $\theta_{ti} \to -\infty$ if $E_i = \emptyset$ as $t \to \infty$ where $h = \sum\limits_{j \in \{i : E_i = \emptyset\}} \frac{p_j}{d - d_0}$, d_0 is the number of the empty subregions, and C is an arbitrary constant. A subregion E_i is called empty if $\sum\limits_{x \in E_i} \psi(x) = 0$. This result states that asymptotically, the working estimates θ_{ti} of $\log(\omega_i/p_i)$ are equal to $\log(\omega_i/p_i)$ after an adjustment for the empty regions. The relevance of this result to the problem of finding stable models or to the problem of finding maximal submodels is that it implies that if we partition the sample space into subregions so that there are subregions that contain only stable models, or there are subregions that contain only maximal submodels, then, assuming that these subregions are not empty, they will be visited by SAMC simulation.

3 The Metropolized Forward Chaining Algorithm

In this section, we shall formally define our MFC algorithm. 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 σ 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^σ, R^σ, and I^σ 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 σ. We let $r(\sigma) = |I^\sigma|$. Thus if $r(\sigma) = 0$, then D^σ is a stable model of P. Let $F_i(P) = \{\sigma \in perm(P) : r(\sigma) = i\}$.

We say 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 $r\,(\sigma) = \min\limits_{\tau \in perm(P)} r\,(\tau)$.

To apply the Metropolis-Hasting algorithm, we need to specify the state space X, the Markov chain $K(\cdot,\cdot)$, and the sampling distribution $\pi\,(\cdot)$. 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 σ, we produce τ by picking k elements $1 \leq i_1 < \cdots < i_k \leq N$ uniformly at random, then picking permutation γ of $i_1, \ldots, i_k$ uniformly at random, and then creating a new permutation by replacing $\sigma_{i_1}, \ldots, \sigma_{i_k}$ in σ by $\sigma_{\gamma(i_1)}, \ldots, \sigma_{\gamma(i_k)}$. 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 $G(\sigma,\tau) = \frac{\pi(\tau)}{\pi(\sigma)}$.

In specifying a distribution π we want $\pi(\sigma)$ to depend only on $r(\sigma)$ and we want the probability of $F_i(P) = \{\sigma : r(\sigma) = i\}$ to be 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 σ where $r(\sigma)$ is small. We shall assume that there exists C_1, C_2, θ, m with $0 < C_1 \leq C_2$, $0 < \theta < 1$, $m \geq 1$ independent of N such that
(a) for $j = 0, 1, ..., n$, $C_1\theta^{-j^m} \leq |F_j(P)| \leq C_2\theta^{-j^m}$ and
(b) for $j = n+1, ..., N$, $|F_j(P)| \leq C_2\theta^{-j^m}$.
This is basically saying that $|F_j(P)|$ is growing exponentially for $j = 1, \ldots, n$, and is bounded by an exponential for $j = n+1, \ldots, N$. 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 [18] described a class of programs called FC-normal programs for which this is true. FC-normal logic programs are a generalization of Reiter's [25] 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 $C = a \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m$ in $nmon(P)$, we have that $cl_{mon}(a) \cap \{c_1, \ldots, c_m\} \neq \emptyset$. Clearly such programs can have no stable models. Thus our assumptions certainly do not apply to all programs P. Instead, our goal is to provide a reasonable set of assumptions that will allow us to define a distribution function $\pi(\sigma)$ where the convergence to orderings σ 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 π_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 2. *For $j = 0, 1, ..., n$, $\lim_{N\to\infty} \pi_N(F_j(P))\, N \geq \frac{C_1}{C_2} > 0$.*
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 $\frac{\pi_N(\sigma_i)}{\pi_N(\sigma_s)} \geq 1$. (Here for any set X, $\pi_N(X) = \sum_{x\in X} \pi_N(x)$).

The relevance of the theorem is that if we are sampling from π, then, for large enough N, the expected number of samples to encounter a stable model increases at most linearly with N - the size of the nonmonotonic part of P.

This given, at any step in the MFC algorithm, we do the following.
Given an ordering $\sigma^{(t)} = \sigma \in perm(P)$, we compute D^σ, R^σ and I^σ by applying the forward chaining algorithm. We then pick a new ordering τ according to the chain $K(\sigma, \cdot)$. That is, we pick k elements of σ uniformly at random and pick a random permutation of those k elements and let τ be the permutation that results by reordering the k chosen elements of σ according to this random permutation. Then we compute D^τ, R^τ and I^τ by applying the forward chaining algorithm. Then we draw U from the uniform distribution on [0,1] and set $\sigma^{(t+1)} = \tau$ if $U \leq \frac{\pi_N(\tau)}{\pi_N(\sigma)}$ and set $\sigma^{(t+1)} = \sigma$ otherwise.

We have three parameters that govern the behavior of the MFC algorithm: the parameter k which is the number of elements of $\sigma \in perm(P)$ that we pick when we move to the next permutation of $nmon(P)$ which determines the proposal chain $K(\cdot, \cdot)$ and the parameters θ 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.

Using SAMC in MFC

To adapt SAMC to MFC, the following must be specified: (1) a proposal chain, $K(\cdot, \cdot)$, (2) a target probability distribution $\pi(\cdot)$, (3) a partition function $J(\cdot)$ that for each x in a sample space returns the index $J(x)$ of the subregion that contains x, (4) a desired sampling distribution $\mathbf{p}$, and (5) a sequence $\{\gamma_t\}_{t=1}^{\infty}$ satisfying (*).

In our experiments we have used the proposal chain $K[k]$ and probability distribution $\pi[\theta, m]$ described in the previous sections. A natural choice for a partition function is $r(\sigma)$ which for a candidate σ returns the number of inconsistent clauses corresponding to σ. Then all the candidates corresponding to stable models of the program P, if such stable models exist, or all the candidates corresponding to the maximal submodels will be contained in a unique subregion. Moreover such a subregion does not contain any other candidates. Nevertheless, our computer experiments indicate that the performance of MFC with SAMC improves if the level sets of $r(\sigma)$ are further partitioned. The basis for such a partitioning is a secondary partition function $\sigma \to i(\sigma)$, that for each permutation of $\operatorname{nmon}(P)$ produces a nonnegative integer $i(\sigma)$ - the index of σ. We require that $i(\sigma)$ lies in $\{\alpha(|\sigma|), ..., \beta(|\sigma|)\}$, for some positive integers $\alpha(|\sigma|) \leq \beta(|\sigma|)$ dependent on the size of permutations. For any statement A, we let $I(A)$ be the indicator function that A is true, i.e., $I(A) = 1$ if A is true and $I(A) = 0$ if A is false.

Suppose that we divide each level set of $r(\sigma)$ into S subregions for some $S \geq 1$. Given a permutation index function $i(\cdot)$, define $J_S(\cdot)$ by

$$J_S(\sigma) = r(\sigma) \cdot S + \lfloor (i(\sigma) - \alpha(|\sigma|))/(\lfloor \frac{\beta(|\sigma|) - \alpha(|\sigma|)}{S} \rfloor + 1) \rfloor .$$

We use the following function $i(\sigma) = \sum_{s=1}^{|\sigma|} s \cdot I(\text{clause } s \text{ is inconsistent})$.

In [12] the authors suggest using a biased desired sampling distribution. We thus construct a desired sampling distribution as follows: let $\rho \in (0,1]$ be a fixed constant. For a region E, let $r(E)$ be the number of inconsistent clauses corresponding to any element of E, here we assume that $E \subseteq \{\sigma | \, r(\sigma) = j\}$ for some j. Then the weight p_E of E is $p_E = \frac{\rho^j}{z}$ where $z = \sum_{E \text{ - is a subregion}} \rho^{r(E)}$. In [12], the authors also suggest that one uses the sequence $\{\gamma_t\}_{t=1}^{\infty}$ where $\gamma_t = \frac{t_0}{\max(t, t_0)}$ for some specified $t_0 > 1$ which is what we did in our computer experiments.

Thus our SAMC adaptation of MFC depends on the following parameters: (1) $k \geq 2$ - the parameter for the proposal chain $K[k]$, which is the number of elements of σ that we permute, (2) $\theta \in (0,1)$, $m \geq 1$ - the parameters for the target probability distribution $\pi(\sigma) = \frac{\theta^{r(\sigma)^m}}{Z}$, (3) the secondary partition function $i(\sigma)$ and $S \geq 1$ - the number of subregions of a level set of $r(\sigma)$, (4) $\rho \in (0,1]$ - the bias constant for the desired sampling distribution, and (5) $t_0 > 1$ - a constant that determines the sequence $\{\gamma_t\}_{t=1}^{\infty}$.

4 Numerical Experiments

In this section, we will briefly report on some numerical experiments.

The 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 [27] proved that for given numbers q and t, there exists a smallest number n - the van der Waerden number $W(q,t)$ - such that for all $m \geq n$, each set partition of the set $\{1, 2, ..., m\}$ into q parts contains at least one subset with an arithmetic progression of at least length t. For example, it is known that $W(2,3) = 9$, $W(2,4) = 35$, $W(2,5) = 178$, and $W(2,6) = 1132$, see [10]. A way to show that $W(q,t) > n$ is to find a (q,t) van der Warden certificate of size n which is a set partition of $\{1, 2, ..., n\}$ into q parts, none of which contain an arithmetic progression of length $\geq t$. This problem is a useful benchmark for the initial study of our algorithm. This is because for $n < 1132$ a $(2,6)$ van der

Waerden certificates of size n exist and hence the failure of an algorithm to find such a certificate cannot be attributed to the absence of a solution.

We carried out numerical experiments on the following simple program whose stable models correspond to partitions of $\{1, \ldots, S\}$ 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 $H(P) = \{i, \bar{i} : i = 1, \ldots, S\}$. First for all $t \in \{1, ..., S\}$ we add the clauses

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

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

$$t \leftarrow \neg t, \neg t+p, \neg t+2p, \neg t+3p, \neg t+4p, \neg t+5p \text{ and}$$
$$\bar{t} \leftarrow \neg \bar{t}, \neg \overline{t+p}, \neg \overline{t+2p}, \neg \overline{t+3p}, \neg \overline{t+4p}, \neg \overline{t+5p}.$$

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

To demonstrate the feasibility of MFC we have conducted four types of numerical experiments.

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 1, 2. 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 $\pi\left[\frac{1}{n}, 1\right]$. 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 1 show that the optimal values for k and θ are: $15 < k < 45$ and $N/140 < \theta < N/20$. More experiments are required to reach a conclusion regarding general applicability of these ranges.

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

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

We have conducted the experiments to study the change in the number of iterations averaged over the experiments with $k = 30$, $n = N/100$, $m = 1$ with the change in the size of the certificate. The experiments were conducted for sizes 150, 160, 170, 180, 190, 200. The experiments have demonstrated that the log of the number of iterations is a quadratic function of the size of a certificate.

Finding Maximal Submodels

Since $W(2, 4) = 35$, see [2], 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 σ of $nmon(P)$ such that $r(\sigma) = 1$. In this case, we can show that the corresponding model D^σ is of the form $M \bigcup \{\overline{k} : k \in \overline{M}\}$ where $(M, \overline{M})$ is a set partition of $\{1, \ldots, 35\}$ such that there is exactly one arithmetic progression of length 4 that is contained in M or $\overline{M}$.

For each set of parameters, 40 experiments were conducted where we stopped when we found a σ such that $r(\sigma) = 1$. The average number of iterations for various sets of parameters are shown in the table 3.

We have also conducted experiments using $n = N/2$, $m = 1$ for various values of k. However only a fraction (roughly 1/3) of these runs completed in under

n	N/30	N/40	N/50	N/60	N/70	N/80	N/100
k=15	6929312	1199948	368252	256846	132098	357942	5308213

Table 3. average number of iterations for $(K(k), \pi_N[1/n, 1])$, $N = 444$, fixed initial state.

10^9 iterations. The value of k where the largest fractions of runs completed in under 10^9 iterations was $k = 15$.

Experiments using SAMC algorithm.
We have conducted a set of experiments to compare the performance of MFC with the Metropolis algorithm and the performance of MFC with SAMC. Three types of experiments were performed: experiments on finding $(2, 6)$ Van der Waerden certificates of size 180, experiments on finding maximal stable models for the problem of finding $(2, 4)$ Van der Waerden certificates of size 35, and the experiments on finding a largest feasible $(2, 6)$ Van der Waerden certificate, that is, experiments in which we attempt to find as large a certificate as we can, given the limitations of the algorithm and that of the computing resources.

For each experiment type, the experiments for MFC based on the Metropolis algorithm and experiments for MFC based on the SAMC algorithm were performed. For all the experiments we fixed $m = 1$. The rest of the parameters were optimized using a univariate method [24]. The details are skipped due to space limitations.

For the problem of finding $(2, 6)$ Van der Waerden certificates of size 180, we conducted 40 experiments for each set of parameters for both versions of MFC. The results are summarized in the table below:

Algorithm	Optimized parameters	Iterations average
SAMC	$k = 30$, $\theta = \frac{1}{6660}$, $t_0 = 50000$, $\rho = 0.2$, $S = 20$	1896839
Metropolis	$k = 20$, $\theta = \frac{110}{6660}$	2919673

The table shows the improvement of using SAMC instead of the Metropolis algorithm of 35%.

For the second type of experiments - those for computing maximal submodels for the program for finding $(2, 4)$ Van der Waerden certificates of size 35, the best performance with the Metropolis algorithm was the average of 132098 iterations for $\theta = N/70$ and $k = 15$. The best performance showed by SAMC was with the parameter values $\theta = 4N$, $k = 20$, $t_0 = 30000$, $\rho = 0.15$ and $S_2 = 20$ and it was the average of 267451 iterations.

Finding Maximal Feasible Certificates. Using MFC with the Metropolis algorithm we were able to produce a size 240 certificate in 504193366 iterations in under 2 weeks. MFC with SAMC was able to produce a size 300 certificate in 841851708 in under 2 weeks. Both runs were performed on a 288 processor cluster.

The difficulty of the problem is illustrated by the fact that when we have conducted experiments using *smodels 2.26* and *clasp 1.3.3* solvers to find van der Waerden certificates (with the program $P(S, 2, 6)$ translated to *smodels* format) on a single processor machine *smodels* failed to find a certificate of size 210 while running for over 3 weeks and *clasp* failed to find a certificate of size 240 while running for over 2 weeks. This set of experiments is far to small to make any significant conclusions about the relative power of solvers as compared with MFC. However, our results do indicate that the problem of finding $(2, 6)$ van der Waerden certificates of sizes ≥ 210 for our program is not easy for the solvers or MFC.

5 Conclusion

In this paper we have studied a Monte Carlo type algorithm for solving 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 $P' \subseteq P$ that has a stable model and find a stable model M' of P'.

The MFC algorithm combines the Forward Chaining algorithm and the Metropolis algorithm and can be easily parallelized. We have investigated a particular setup for MFC that uses a proposal chain $K\,[k]$ and the target probability distribution $\pi\,[\theta, m]$. There are a number of other proposals for the chain $K(\cdot,\cdot)$ 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 τ 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 $\pi(\cdot)$ 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 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. One can not use this approach to find maximal subprograms that have stable models of a program that does not have a stable model which is an important property for some applications we have in mind. However, one could argue that this approach might be preferable to MFC in that the search space is 2^m where m is the cardinality of the set of conclusions of P as opposed to the search space of MFC which has size $n!$ where $n = |\mathrm{nmon}\,(P)| \geq m$. However this may not necessarily be the case. While MFC is searching through the set of $n!$ well-orderings of $\mathrm{perm}\,(P)$, many of these will produce the same stable models. Thus the question of which approach is preferable requires additional research.

The experiments described in this paper have demonstrated that the MFC algorithm can successfully solve our two problems. Our goal in these experiments was to show that the choice of the target distribution and the number k, of clauses that we permute at each step of the MFC algorithm can have a significant effect on the performance of the MFC algorithm and our experiments clearly indicate that this is the case. This suggests that in any particular application where one plans to run the MFC algorithm multiple times, it is certainly worth the effort to tune such parameters to achieve maximum performance.

A major drawback of the Metropolis algorithm is the local trap problem (see section 1). To overcome this problem we investigated a version of MFC where we used the SAMC algorithm in place of the Metropolis algorithm. Our experiments indicate that in certain cases, this significantly improves the performance of the MFC algorithm.

In general, our computer experiments were too limited to draw any significant conclusions about the question of what is the best choice of parameters and variant of the Metropolis algorithm that will yield the optimal performance for the MFC algorithm. We suspect that there is no global answer, but that the choice is problem specific. Nevertheless, our computer experiments indicate that this is interesting area for future research.

More research is necessary to determine how MFC compares with the existing algorithms for finding stable models of logic programs. Regardless of the competitiveness of the MFC algorithm in finding stable models, we feel that its ability to find maximal size subprograms that 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 that have stable models can represent a way to satisfy as many preferences as possible. This will be the subject of future work.

Acknowledgments. 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. Y. F. Atchade, J. S. Liu, The Wang-Landau algorithm in general state spaces: applications and convergence analysis. *Technical Report, Department of Statistics, University of Michigan*, (2007).
2. V. Chvatal, Some unknown van der Waerden numbers. *Combinatorial Structures and Their Applications*, (1970), pp. 31-33.
3. P. Diaconis, The Markov Chain Monte Carlo Revolution. *Bulletin of the American Mathematical Society*, **46** (2009), 179-205.
4. M. Dunn, S. J. Greenhill, S. C. Levinson, R. D. Gray, Evolved Structure of Languages Shows Lineage-Specific Trends in Word-Order Universals. *Nature*, **473**, (2011), 79-82.
5. C. Elkan, A Rational Reconstruction of Nonmonotonic Truth Maintenance Systems. *Artificial Intelligence,* **43** (1990), 219-234.
6. W. Hastings, Monte Carlo sampling methods using Markov chains and their applications. *Biometrica*, **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 Journal of Combinatorics*, **14** (2007), #R6.

8. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, Conflict-driven answer set solving. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, (2007), 386–392.
9. S. Karlin and H. M. Taylor, A First Course in Stochastic Processes. Second Edition. Academic Press. (1975).
10. M. Kouril and J.L. Paul, The van der Waerden number $W(2,6)$ is 1132. *Experimental Mathematics*, **17:1** (2008), 53-61.
11. F. Liang, On the use of stochastic approximation Monte Carlo for Monte Carlo integration. *Statistics and Probability Letters* **79** (2009), 581-587.
12. F. Liang, C. Liu, and R. J. Carroll, Stochastic Approximation in Monte Carlo Computation. *Journal of the American Statistical Association*, **102** (2007), 305-320.
13. J. S. Liu, F. Liang, and Wing Hung Wong, A Theory for Dynamic Weighting in Monte Carlo Computation. *Journal of the American Statistical Association*, **96** (2001), 561-573.
14. J. S. Liu, Monte Carlo Strategies in Scientific Computing. Springer. (2001).
15. L. Liu 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*, LNCS 2833 (2003), 495-509.
16. L. Liu and M. Truszczyński, Local search techniques for Boolean combinations of psuedo-Boolean constraints. *Proceedings of the Twentieth National Conference on Artificial Intelligence*, AAAI Press (2006), 98-103.
17. L. Liu and M. Truszczyński, Statisfiability testing of Boolean combinations of pseudo-boolean constraints using local search techniques. *Constraints*, **12**(3) (2007), 345-369.
18. 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.
19. 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.
20. W. Marek, J.B. Remmel, On the expressibility of stable logic programming. *Theory and Practice of Logic Programming*, **3(4,5)** (2003), 551-567. .
21. W. Marek, J.B. Remmel, Extensions of Answer Set Programming, to appear in a special volume for Nonmon@30.
22. W. Marek, M. Truszczynski, Computing intersection of autoepistemic expansions. *Logic Programming and Non-monotonic Reasoning, Proceedings of the First International Workshop*, (1991), 37-50.
23. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, **21** (1953), 1087-1092.
24. S. S. Rao, Engineering Optimization: Theory and Practice. Fourth Edition. *John Wiley & Sons, Inc.* (2009), p. 315.
25. R. Reiter, A logic for default reasoning. *Artificial Intelligence*, **13** (1980), 81-132.
26. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. **138** (2002), 181–234.
27. B.L. van der Waerden, Beweis einer Baudetschen Vermutung. *Nieuw Archief voor Wiskunde*, **15** (1927), 212-216.