1 Introduction

Let $S_1 = B^n$ be a finite n-dimensional binary space and let $f(x), x \in S_1$ be a function such that $f(x) \mapsto R \geq 0$. We are interested in the study of algorithms that calculate the maximum of a function by estimating the exact Boltzman distribution π_{MAX} defined by:

$$\pi_{MAX}(x) = \frac{1}{Z_T} e^{\frac{f(x)}{T}} \tag{1}$$

where Z_T is the usual partition function. Along this paper we will also use the inverse temperature $\beta = \frac{1}{T}$ as a parameter of the search algorithms.

A number of times the Boltzmann distribution has been proposed as a search distribution for optimization problems. If points can be generated according to the Boltzmann distribution, then points of high fitness are generated more often than points of low fitness. The central problem is how to efficiently compute this distribution.

There has been a new proposal to use the Boltzmann distribution in *population based search methods* [19]. The proposed algorithm is defined as follows:

BEDA – Boltzmann Estimated Distribution Algorithm

- **STEP 0:** $t \Leftarrow 0$. Generate N points according to the uniform distribution $p(\mathbf{x}, 0) = 2^{-n}$.
- **STEP 1:** With a given $\Delta\beta(t) > 0$, let

$$p^s(\mathbf{x}, t) = \frac{p(\mathbf{x}, t) e^{\Delta\beta(t) f(\mathbf{x})}}{\sum_y p(\mathbf{y}, t) e^{\Delta\beta(t) f(\mathbf{y})}}.$$

- **STEP 2:** Generate N new points according to the distribution $p(\mathbf{x}, t+1) = p^s(\mathbf{x}, t)$.
- **STEP 3:** $t \Leftarrow t + 1$.
- **STEP 4:** If stopping criterion not met go to STEP 1

BEDA is a conceptional algorithm, because the calculation of the distribution requires to compute the sum of exponentially many terms. The following convergence theorem is easily proven.

Theorem 1 (Convergence). *Let $\Delta\beta(t)$ be an annealing schedule, i.e. for every t increase the inverse temperature β by $\Delta\beta(t)$. Then for BEDA the distribution at time t is given by*

$$p(\mathbf{x}, t) = \frac{e^{\beta(t) f(\mathbf{x})}}{Z_f(\beta(t))} \tag{2}$$

with the inverse temperature

$$\beta(t) = \sum_{\tau=1}^{t} \Delta\beta(\tau). \tag{3}$$

Let $\mathcal{M}$ be the set of global optima. If $\beta(t) \to \infty$, then

$$\lim_{t\to\infty} p(\mathbf{x}, t) = \begin{cases} 1/|\mathcal{M}| & x \in \mathcal{M} \\ 0 & else \end{cases} \tag{4}$$

There have been a number of methods proposed how to compute efficiently the Boltzmann distribution using a population of points. [18, 19].

In this paper we will investigate other techniques to approximate the Boltzmann distribution. These techniques are based on *single searchers*.

The main problem associated which calculating π_{MAX} is to calculate the normalizing constant Z_T. In order to calculate Z_T the evaluation of all the points of the search space is required. This is impossible for high dimensional problems. Therefore a number of algorithms have been proposed for the problem of sampling from π_{MAX}.

The outline of the paper is as follows. First we review important algorithms based on approximate Markov Chains Monte Carlo (MCMC) methods. In section 3 an exact MCMC algorithm is presented, its advantages over classical MCMC methods are discussed. In 4 we concentrate on the use MCMC in optimization. A number of optimization algorithms based on exact and inexact MCMC methods are introduced. Section 5 shows the numerical results obtained in the application of the algorithms considered in the paper. The conclusions of our paper are presented in section 6

There is a vast and always growing literature in Monte Carlo Markov Chain algorithms. For our introduction to the field we have made use of different previous reviews [1, 2, 4, 14, 16] that we recommend to readers interested in more details. Nevertheless, as our main objective is the connections between MCMC and the research in optimization our research is different from previous approaches.

2 Approximation of the distribution by stochastic sampling

Stochastic sampling algorithms (also called stochastic simulation or Monte Carlo algorithms) are a family of algorithms used to approximate a distribution π and to estimate expectations derived from π. From now on we will assume that π is our Boltzman distribution π_{MAX}. Stochastic sampling algorithms include methods that only generate independent samples from the desired distribution (e.g. rejection sampling) and other methods where this is not necessarily the case like Markov Chain Monte Carlo methods. MCMC methods base the sampling of a distribution on a Markov chain.

A Markov chain is specified by:

- A sequence of discrete random variables $X^{(0)}, X^{(1)}, ...$
- A marginal distribution for the initial state $X^{(0)}$

- Transition probabilities for state $X^{(t+1)}$ to follow state $X^{(t)}$, $P(x^{(t+1)}|x^{(t)})$,

This determines the joint distribution $X^{(0)}, X^{(1)}, ...$ by making the Markov assumption that $X^{(t)}$ is conditionally independent of $X^{(t-k)}$ for $k > 1$ given $X^{(t-1)}$.

Stationary Markov chains have transition probabilities that do not depend on time, which can be represented by a transition matrix M. $M_{x,x'}$ denotes the probability of a transition to state x' given that the system is in state x. The state probabilities at time t can be represented as a row vector p_t. Then $p_t = M^t p_0$. Given a Markov chain with transition probability matrix M a distribution π is invariant (or stationary) if $\pi = \pi M$. This is called the general balance condition with respect to π, and it means that if $M(x, x')$ denotes the probability of a transition from x to x' under M, we require that

$$\sum_{x \in S1} \pi(x) M(x, x') = \pi(x') \tag{5}$$

for all $x' \in S_1$. If $p_t = \pi$ for some t, then the Markov chain has reached the invariant distribution by time t and will continue in that distribution forever. An ergodic Markov chain has an invariant distribution that is reached asymptotically no matter what the initial distribution p_0 is, i.e.

$$\lim_{t \to \infty} p_t = \pi \tag{6}$$

MCMC methods comprise different algorithms able to devise an ergodic Markov chain such that the steady-state distribution of the chain is π. To approximate the probability, the chain can be started in any point $x \in S_1$, and after a sufficiently long time (say for k steps) take each state of the chain as a sample of the desired probability π. The number of simulation steps T necessary before the Markov chain is close to its stationary distribution is called the mixing time or mixing rate of the chain. The MCMC literature refers to the initial time the Markov chain is run until it is assumed to be closed enough to stationarity as the burn-in period.

Classical MCMC algorithms can determine only an approximation of π, this approximation depends on the burn-in period. In general it is a hard task to estimate a priori the burn-in period such that $\|\pi - \sigma\| < \epsilon$, where σ is the approximation of π, and ϵ is the acceptance error. When bounds for the burn-in period can be calculated they are usually not tight enough for being of practical use. Another problem related with this is how to determine, from the observation of the MCMC methods' output if convergence has been already reached (convergence diagnostics). We will show later that a recently introduced class of MCMC methods that make an exact approximation of π provides solutions for both of these problems.

2.1 Non exact MCMC algorithms

MCMC algorithms make use of an ergodic Markov chain with stationary distribution π. We present now some of the best known non exact MCMC algorithms.

2.1.1 Metropolis algorithm

In the Metropolis method a proposal density function $Q(x)$ is used which depends on the current state $x^{(t)}$. The proposal density $Q(x'; x^{(t)})$ can be any fixed density. A tentative new state x' is generated from the proposal density $Q(x'; x^{(t)})$. To decide whether to accept the new state, we compute the quantity:

$$a = \frac{\pi^*(x')Q(x^{(t)}; x')}{\pi^*(x^{(t)})Q(x'; x^{(t)})} \tag{7}$$

in our case $\pi^*(x) = e^{\frac{f(x)}{T}}$, and it is assumed that $\pi^*(x)$ can be evaluated for any x. If $a \geq 1$ then the new state is accepted, otherwise it is accepted with probability a.

To compute the acceptance probability we need to be able to compute the probability ratios $\frac{\pi^*(x')}{\pi^*(x^{(t)})}$ and $\frac{Q(x^{(t)};x')}{Q(x';x^{(t)})}$. If the proposal density is a simple symmetrical density then the latter factor is unity, and the Metropolis method simply involves comparing the values of the targeted density at the two points. The general algorithm for asymmetric Q, given above, is also called the Metropolis-Hastings algorithm.

2.1.2 Gibbs Sampling algorithm

The most frequently used among non exact MCMC algorithms is the Gibbs sampler [8], also known as the heat bath algorithm for the binary and discrete case. It can be viewed as a Metropolis method in which the proposal distribution Q is defined in terms of the conditional distributions of $\pi(x)$. It is assumed that whilst $\pi(x)$ is too complex to draw samples from it directly, its conditional distributions $\pi(x_i|x_{-i})$ are tractable to work with.

To draw samples from $\pi(x)$, start with an initial state $x^0 = (x_1^0, ..., x_n^0)$. At each time t of the Gibbs sampler, a location i to be updated is randomly chosen from x^t and the new vector x^{t+1} is selected using the transition rules in (8).

$$x^{t+1} = \phi(x^t, U, W) = \begin{cases} x_j^{t+1} = x_j^t & \text{for } j \neq i \\ x_j^{t+1} = \pi(. \mid (x^t - \{x_i^t\}) & \text{for } j = i \end{cases} \tag{8}$$

A convenient representation of the updating process of the Markov chain is achieved using the deterministic function $\phi(x, U, W)$ where U is drawn uniformly on $S_2 = (0, ..., n)$ and decides the site i to be updated. W is a uniform $(0, 1)$ variable that is used to determine the transition value for x_i^t based on the conditional probabilities. The algorithm is presented in algorithm 1. k refers to the burn-in period and $N_s = k+ \mid A \mid$ where $\mid A \mid$ is the size of the sample set A.

The Gibbs Sampling algorithm is also a building block of the Coupling From The Past (CFTP) or Propp-Wilson [21] algorithm, an exact MCMC technique that will be treated in the next section.

Algorithm 1: **Gibbs Sampler**

1 Set $t \Leftarrow 0$. Generate randomly an initial point x^0 from the state space S_1.
2 While $t < k$ generate $x^{t+1} = \phi(x^t, U, W)$.
3 While $t < N_s$ generate $x^{t+1} = \phi(x^t, U, W)$ and add it to the sample set A.
4 Estimate π using the sample set A.

2.2 Improvements to non-exact MCMC methods

Although the algorithms presented above have been successfully used in many fields they show limitations in their ability to escape from deep local minima of π during the simulation. The expected waiting time for moves from states of high probability to states of lower probability can be very long. In this context arises the dilemma of waiting a long time in a local maximum or to have an incorrect equilibrium distribution. To address this question several improvements have been proposed. A number of methods have been used to attempt to "accelerate" the dynamics of Monte Carlo updates.

We focus our analysis on two classes of algorithms designed to overcome this problem. The first class of techniques uses the temperature of the system or other parameters as an additional random variable during the simulation, and the second employs non-local update moves in the simulation of the Markov chain. References to other techniques used for speeding up Monte Carlo methods can be found in [16].

The first class comprises algorithms like Simulated and Parallel Tempering [17, 9], and Dynamic Weighting [27]. The idea of these algorithms is to use the "temperature" to control the Markov chain simulation in such a way that the chain can spend less time to scape from local optima. The second class comprises algorithms that instead of making a change in a single variable during the simulation propose ways of constructing and flipping clusters or blocks of variables to speed the simulation.

2.2.1 Simulated Tempering

To sample the distribution π Simulated Tempering (ST)[17] uses a family of distributions $\prod = \pi_i(x), i \in I$ by varying a single parameter, the temperature, in the target distribution π. Distribution π corresponds to the member of this family with the highest temperature of the system (i is the temperature index). Then a new target distribution, $\pi_{st}(x, i) \propto c_i \pi_i(x)$, is defined on the augmented space $(x, i) \in S_1 \times I$. c_i is a constant which purpose is to allow each temperature level to have reasonable chance of being visited. Finally, a MCMC sampler is used to draw samples from π_{st}. The intuition behind ST is that by heating up the distribution repeatedly, the new sampler can escape from a local mode and improve its mixing rate.

A related algorithm is the Parallel Tempering (PT) [9] where instead of augmenting the space to include the temperature as a variable, a joint probability distribution on the product space of the $|I|$ distributions is defined, each of the distributions is characterized by its own temperature and parallel MCMC schemes are run on each space. Instead of

doing temperature transitions like in ST an "index swapping" operation between spaces is incorporated.

In dynamic weighting MCMC [27] a dynamic weighting variable is introduced for controlling the Markov chain simulation. This fact allows the system to make transitions against a steep probability barrier without a proportionally long waiting time. To account for the bias caused by such transitions, an importance weight is computed and it is saved along with the sample state vector. At equilibrium, estimates for expectations are obtained by importance-weighted average of the sampled values, rather than the simple average as in the Metropolis method.

2.2.2 Blocked Gibbs Sampling methods for approximation

Another way of speeding the mixing rate of the Markov chain is to use non local update MCMC that change the value of a block of variables instead to update only one site of the vector. One member of this class is the clustering-flipping algorithm proposed by Wolff [26] with application to the simulation of spin models. In this algorithm clusters are built up by starting from a randomly-chosen seed spin and, with a given probability, adding further spins if they are adjacent and point to the same direction that the spins already in the cluster. This algorithm has shown good results for the simulation of spin models , nevertheless, the way in which clusters are constructed in algorithms like that of Wolff, are very dependent to the particular characteristics of the model used. These algorithms do not seem to be of direct application to other problems.

Research done [15, 23] in the use of MCMC methods for Bayesian computation has identified that the performance of one-at-a time MCMC samplers deteriorate for problems with highly correlated components. A remedy for this problem has been to "block" the variables into groups that are then updated simultaneously using a Gibbs or Metropolis Hasting step [3]. This technique is called "blocking" and when it is used with Gibbs Sampling the algorithm is usually called blocked Gibbs Sampling. There exist different versions of this class of technique, for instance some implementations [24, 5] employ a randomization of the block size during the simulation.

In this paper we present results in the use of the Blocked Gibbs Sampling for the problem of approximating π and finding the optimum of $f(x)$. In our implementation the block size is fixed along the simulation.

Algorithm 2: **Blocked Gibbs Sampler**

1. Set $t \Leftarrow 0$. Generate randomly an initial point x^0 from the state space S_1.
2. Select a set Ng of s different uniformly selected variables of X.
3. While $t < k$ generate $x^{t+1} = \phi(x^t, U, W, Ng)$.
4. While $t < N_s$ generate $x^{t+1} = \phi(x^t, U, W, Ng)$ and add it to the sample set A.
5. Estimate π using the sample set A.

$$x^{t+1} = \phi(x^t, U, W, Ng) = \begin{cases} x_j^{t+1} = x_j^t & : \quad for\ j \notin Ng \\ x_{Ng}^{t+1} = x'_{Ng} & if \quad W \leq \frac{e^{\frac{f(x')}{T}}}{e^{\frac{f(x')}{T}} + e^{\frac{f(x)}{T}}} \\ x_{Ng}^{t+1} = x_{Ng}^t & otherwise \end{cases} \tag{9}$$

In algorithm 2 we present our blocked Gibbs Sampling. The algorithm receives as a parameter the block size s, that in our case is understood as the maximum number of variables that will change their values together. s also determines a neighborhood of the solution x comprised by those vectors that have a Hamiltonian distance to x equal or less than s. The extension of the burn-in period k, and the maximum number of steps N_s are inputs of the algorithm too.

The transition rule for the state, given the set of variables to be updated (Ng) is shown in equation (9). The determination of the proposal comprises two steps. To determine the set of variables to be updated, and to select the new values for this set of variables. Available information about the structure of the problem can be used at both steps. In the first step we can select with higher probability s variables known to be related. In the second step also the assignments of values for the variables can be biased taking into account previous information.

If Ng were randomly selected it could be the case that the number of sized s blocks of related variables be very small compared to the total number of combinations of subsets of variables with size s. In this case the possibility of randomly choosing s related variables vanishes. Then in our experiments we consider still another version of the blocked Gibbs Sampling where the s variables to be chosen will be related (they appear together in the definition set of an additive function) with a probability p_u. This implies to change only the second step of the algorithm shown in algorithm 2. What we expect is that increasing the probability of selecting a block of s related variables the mixing time will decrease even faster.

We have used two different ways of choosing the new values for the current block of variables Ng. These two ways define different forms of making the transitions in the neighborhood of solutions. We will refer to them as $Neig1$ and $Neig2$. In $Neig1$ a uniformly random value x'_{Ng} is selected among the $2^s - 1$ possible values different from the current one. $Neig2$ has been implemented as in [20], the probability of taking a new value x'_{Ng} depends on the Hamiltonian weight of the final solution. We will go back to this question later in the paper.

To compare all the versions of the algorithms a measure of the error of the approximation $\sigma(t)$ at time t for the Markov chain simulation is needed. We will employ the total variation distance, commonly used in the MCMC literature to study the mixing rate of Markov chains. This measure is defined as:

$$\| \sigma_t - \pi \| = \frac{1}{2} \sum_{x \in S_1} | \sigma_t(x) - \pi(x) | \tag{10}$$

The error decays exponentially with time, asymptotically according to:

$$error = ae^{-\frac{t}{c}} \tag{11}$$

where a and c are constants that depend on the Markov chain specification.

3 Exact MCMC algorithms

Non exact MCMC have the drawback that it is very difficult to find bounds for the burn-in period and to determine when convergence has been achieved. Exact MCMC solve this problem. To get an intuitive idea of how these algorithms work let us consider the case when a path of a Markov chain begins from each possible state of the space. Lets say that two of these stochastic processes are coupled if their paths coincide after a random time, the coupling time. It is clear that if the same transition function and random parameters are subsequently used for both processes whose paths have coincided, then subsequent states of the two paths will also coincide. As time passes almost certainly all the paths will coincide, and after some time, the simulation will be sampling states from only one path. It is reasonable to expect that given that the Markov chain converges to the target distribution, the samples from such a unique path are precisely samples from π.

3.1 Coupling From The Past or Propp-Wilson algorithm

If paths meet or merge we say that they have coalesced. The notion of coalescence is essential for the understanding of the Propp-Wilson or Coupling From The Past (CFTP) algorithm. Consider that $N = |S_1|$ copies of a Markov chain are run from every possible initial state. Let the same updating function and sequence of random numbers be used for all the chains. If all the chains have reached the same state we say we have coalescence. From the step where coalescence has happened and onward all the chains will follow the same sample path. Propp and Wilson have shown [21] that when coalescence of all the paths has been achieved the effect of the initial state has worn off and the current state can be taken as a sample from the exact distribution. Nevertheless, running all the chains from an initial state zero to achieve coalescence on an instant t of the future could yield to obtain biased samples. We present a simple example taken from [7], interest reader could find other examples of this effect in [21, 11].

Consider the following transition matrix on the state space $0, 1$. From state 0 the chain moves to 0 or to 1, with probability $\frac{1}{2}$ each. From state 1 the chain moves deterministically to 0. The stationary distribution is given by $\pi(0) = \frac{2}{3}, \pi(1) = \frac{1}{3}$. However, with probability one the two copies of the chain will first coalesce at state 0. The solution to this bias problem in the Propp-Wilson algorithm is to run the chains from the past into the present instead of from the present into the future. To formally introduce the algorithm we employ the notation used in [4].

There is a family of functions $f_\phi : \phi \in \Theta$ from S_1 to S_1, f_ϕ in the transition function, a random map which specifies a transition for each state according to the transition kernel of the chain. If all realizations of the random map are monotone functions then

we call it a monotone transition rule. There is also a probability μ on Θ, so that π is the stationary distribution of the forward chain on S_1. In other words, for each $y \in S_1$,

$$\sum_{x \in S_1} \pi(x)\mu\{\phi : f_\phi(x) = y\} = \pi(y) \tag{12}$$

A common choice in the construction of these functions is the Gibbs sampler, as presented in algorithm 1. If the construction of the chain is successful, the backward iterations

$$(f_{\phi_{-M}} \circ f_{\phi_{-M+1}} \circ ... \circ f_{\phi_0(s)}) \tag{13}$$

will converge to a limiting random variable with distribution π.

In essence, to pick a random element of S_1 with respect to π the Propp-Wilson algorithm runs a Markov chain from the indefinite past until the present, where the distance into the past the algorithm has to look is determined dynamically, and more particularly, is determined by how long it takes for the $|S_1|$ runs of the Markov chain (starting in each of the $|S_1|$ possible states, at increasingly remote times) to coalesce [21]. The coupling time is the expected number of steps until coalescence.

When the state space is very large it is not feasible to run one Markov chain for all the possible states. An alternative solution has been proposed for the situation when S_1 is a partially ordered set with smallest and largest elements, and it is possible to find a monotone transition function (i.e. $x \leq y \Rightarrow f_\phi(x) \leq f_\phi(y)$). Then coalescence can be checked by looking only at the chains that begin in the smallest and highest states. This is to verify in every step t if

$$(f_{\phi_{-M}} \circ f_{\phi_{-M+1}} \circ ... \circ f_{\phi_t}(0)) = (f_{\phi_{-M}} \circ f_{\phi_{-M+1}} \circ ... \circ f_{\phi_t}(1)) \tag{14}$$

Algorithm 3: **Propp-Wilson algorithm**

```
1  T ⇐ 1
2  do {
3      upper = highest state
4      lower = smallest state
5      for t ⇐ −T to −1
6          upper ⇐ f_φ(upper)
7          lower ⇐ f_φ(lower)
8      T ⇐ 2T
9  } until upper=lower
10 return upper
```

The CFTP algorithm also needs a schedule for making jumps further into the past every time that coalescence has not been achieved at time 0. The most used schedule begins at time $T = -1$, and updates the beginning time as $T \leftarrow 2T$. Another important requirement of the algorithm is that in every time t the choice from f_ϕ must be the same. This requirement can be better understood with the help of an example we present later. When using the Gibbs sampler as the transition function we have included variables U and W in the definition of ϕ. We guarantee that at instant t the same site i and random value α will be selected for making the transition $x^t \rightarrow x^{t+1}$. Algorithm 3 presents the steps of the CFTP.

3.2 Interruptible algorithm for Perfect Sampling

Although the CFTP guarantees that the exact distribution will be sampled it is important that when the algorithm is taking a long time to reach coalescence, the experimenter does not simply interrupt the current run of the procedure and discard its results, retaining only those samples obtained during earlier runs. The run must either be allowed to terminate or else regard the final sample as indeterminate. Otherwise the sample could be contaminated with some bias because in general the running time of the CFTP in terms of number of transitions is not independent of the state sampled.

In [7] Fill has introduced an exact sampling algorithm for finite state space Markov chains which, in contrast to CFTP, is unbiased for user impatience. The algorithm does not use the idea of coupling from the past but is based on rejection sampling. Like the CFTP algorithm Fill's algorithm requires an ordering of the state space with unique maximal and minimal elements. Also monotonicity conditions are required. The algorithm needs to simulate in every step two chains, and similarly to the CFTP every time a sample is rejected the number of steps the algorithm goes back to the past is doubled.

In our experiments we only used the Propp-Wilson algorithm. In general it could be affirmed that for the purposes of optimization both algorithms exhibit similar benefits (they guarantee an exact approximation) and drawbacks (the number of transitions needed to get a sample is higher than for approximate Markov chain simulations). For recent accounts of research on exact simulation of Markov chains [22, 25] can be consulted. Examples of the bias introduced in the approximation of π by an early interruption of the CFTP algorithm can be found in the paper by Fill [7]. Finally, we point out that exact approximation is at the moment a very active research area. Research devoted to relax the strict monotonicity conditions required by the distributions to be approximated [11, 6] could be of interest for future applications in optimization.

4 MCMC in optimization

Our goal is to evaluate the performance of the MCMC algorithms we have presented above in the framework of optimization. Like SA for Markov chain simulation, both Gibbs Sampling and the CFTP algorithms need a number of minor modifications to deal with the optimization problem.

4.1 Simulated Annealing

It is important to highlight the existing differences between the problem of estimating the Boltzmann distribution for a given function $f(x)$ and the problem of finding the element of the space with highest probability (e.g. the optimum of $f(x)$). If we have the exact Boltzman distribution π or an approximation σ of π that satisfies: $\sigma(x) > \sigma(y)$ if and only if $\pi(x) > \pi(y)$ then the optimum is known. But in aim to find the optimum a good approximation of the Boltzmann distribution is not necessary. We only need to generate those points which have high probability.

To illustrate this point with an example we compare the case of the well known Simulated Annealing (SA) method [13] with the Simulated Tempering mentioned before. SA is inspired in the annealing process that arises in physics. It uses a non increasing function $T : N \rightarrow [0, \infty]$, called the cooling schedule. N is the set of integers, and $T(t)$ is called the temperature at time t. Also for each $x \in S_1$, a set $S(x) \subset S_1 - \{x\}$ called the set of neighbors of x is defined. The algorithm, that is usually presented in the context of function minimization with $\pi_{MIN}(x) = \frac{1}{Z_T} e^{-\frac{f(x)}{T}}$ is shown in algorithm 4.

Algorithm 4: **Simulated Annealing**

1 $t \Leftarrow 0$, Set $x(0) = x^0$
2 **do** {
3 Generate a neighbor y of $x(t)$
4 **if** $f(y) \leq f(x(t))$ **then** $x(t+1) = y$
5 **if** $f(y) > f(x(t))$ **then** $x(t+1) = y$ with probability $exp^{-\frac{(f(y)-f(x(t)))}{T(t)}}$
6 $t \Leftarrow t+1$
7 } **until** Termination criteria are fulfilled
8 **return** $x(t)$

SA uses the temperature as a dynamical variable that changes with time and may allow the system to make transitions which would be improbable at a fixed temperature. However there are two main differences between SA and ST, determined to a great extent by the different goals they have as an optimization technique and a sampler, respectively.

The first difference is that in SA the schedule for varying the temperature is commonly fixed in advance. Usually the suggested schedule for SA increases the inverse temperature ($\beta = \frac{1}{T}$) monotonically, and it has been pointed out [20] that such strategies are bad for functions where a valley has to be crossed after a large number of trials. In ST the way in which the temperature is updated depends on the simulation, although the choice of the initial parameters c_i is also a critical step for the behavior of the algorithm.

The second difference between both algorithms is that while in ST the changes in temperature keep the system in equilibrium, in SA every change of the temperature drives the system out of equilibrium. This is a logical consequence of the SA strategy that is not committed to an exact sampling but only to reach the optimum, something

that can be done without satisfying the equilibrium condition.

4.2 Gibbs Sampling and CFTP in the optimization context

In this paper we evaluate the convenience of using the CFTP and the Gibbs Sampling as optimization algorithms. When used for optimization the CFTP algorithm is started in the smallest and highest states of the ordered space, and it progresses until it is verified that there exists coalescence at time zero. Notice that there is no need to test the coalescence condition at every point $t < 0$ because anyway the sample will be taken at time 0, and if coalescence has been achieved before this time path will remain merged from that time on.

Once coalescence has been verified, if the sample point is the optimum the algorithm stops, otherwise it is started again until coalescence is achieved at an optimum of the function or a maximum number of runs of the CFTP has been achieved. For all the functions considered in our experiments the optimum is known in advance. Every transition in the Propp-Wilson algorithm counts as two function evaluations corresponding to the paths started at the smallest and highest states. Certainly, from the time of coalescence to time zero only one evaluation is really needed in every time step, nevertheless we did not consider this improvement in our experiments because it would have meant to test the coalescence condition in every step of the simulation.

It is important to point out that when conditions of monotonicity have not been verified the satisfaction of the coupling condition does not guarantee that the state is a sample from the target distribution. This is because the trajectories started at different states are not sandwiched between the trajectories begun at the smallest and highest states, and thus it can not be affirmed that all the trajectories have really coalesced.

For comparison with the CFTP algorithm we have used a Gibbs sampler with some modifications that permit it to work as an optimization algorithm. The algorithm will begin in a random state and stops whenever the optimum has been found or a maximum number of transitions have been done. Recall that when the Gibbs Sampling is employed for approximating π only the latter stop condition is used. Every transition of the Markov Chain is counted as one function evaluation. This condition can be relaxed for those functions where only a partial evaluation is needed for implementing the transition.

In our implementation of the Gibbs sampler the algorithm receives as parameters the size of the burn-in period k, and a maximum number of points to be sampled N_s. It uses as an auxiliary data structure an array that contains the frequency of all the different points sampled, and outputs the estimation of π that is calculated from this auxiliary structure.

4.3 Blocked Gibbs Sampling for optimization

Recently Mühlenbein and Zimmermann [20] have shown that for stochastic local search algorithms like SA the size of the neighborhood is more important than the temperature schedule. In fact, in the field of local optimization, it has been reported that updating more than one variable at the time can be a good heuristic for escaping local optima

when sequential optimization algorithms are used. This is the case for example of GSAT [12], a very effective local optimization strategy used to solve the Satisfiability problem, where the number of variables to be updated in each step is not fixed.

We hypothesize that improvements achieved by enlarging the local neighborhood of stochastic local techniques that use the Boltzman distribution are (at least in part) due to a reduction of the mixing time of the Markov chain. If so is the truth, two conditions have to be fulfilled:

- The whole Boltzman distribution can be approximated using Markov chains with transition rules that update more than one variable at the time.
- For a certain class of functions, a Markov chain with transition rules that update more than one variable at the time can be faster in approximating π than a transition rule that updates only one variable.

The rationale behind these statements is the same that has been shown above to explain the performance of the Boltzman Estimation Distribution Algorithm. There could be a mapping between the definition sets of an additive function and the dependency sets that arise among its variables when a Boltzman distribution based on this function is calculated. If additionally the subfunctions defined on the definition sets of this function are deceptive, then algorithms that only change one variable in every step take a long time to make an improvement during the search.

In MCMC a similar related effect has been described [10]: The presence of modes. When this holds, if the ordinary Gibbs Sampling is started from some distribution that is biased towards one of the modes, it will take a long time to visit all of the areas of the state space with the desired frequency. Many small movements will be made around some locality of the state space before a transition to another area is made. Something similar happens to the CFTP algorithm, the probability of a transition of two chains to the same state can be quite low if they start in different modes.

The blocked Gibbs Sampling (algorithm 2) can be used as an optimization algorithm in the same way the ordinary Gibbs Sampling is used. When used for optimization the blocked Gibbs Sampling is in fact a stochastic neighborhood search technique. As stated before the way in which variables are selected and transitions to new values are done determine the structure of the neighborhood. This structure has an important role in the efficiency of the search.

4.4 The importance of the structure of the neighborhood

We will constrain our analysis to the case of the algorithms that use neighborhood search, where the neighborhood size s is fixed. A transition from the current state to the next state is done by changing at most s variables of the current state. The search can be done in two steps.

- The s variables that will be changed are selected.
- The values of all or some of the variables are changed.

Let us suppose that the optimum of $f(x)$ is located at point $x = (1, 1, ..., 1)$. We want to estimate the probability of making a transition from a point with unitation u, where the unitation is the number of variables set to 1 in the vector, to the optimum using a neighborhood search algorithm with neighborhood size s. Let us call this probability P_{trans}. To hit the optimum the following facts have to occur simultaneously.

- The $(n-u)$ variables with value 0 are selected among the s variables.
- The new proposal changes the values of these $(n-u)$ variables and keep the values of the rest of the variables intact.
- The new proposal is accepted.

Then P_{trans} can be factorized as:

$$P_{trans} = P_{sel} \cdot P_{opt} \cdot P_{acceptance} \tag{15}$$

where $P_{sel} = \frac{\binom{u}{s-(n-u)}}{\binom{n}{s}}$ is the probability of having the $(n-u)$ variables among the s variables selected. P_{opt} is the probability of changing the $(n-u)$ variables to 1, while keeping the remaining $s-(n-u)$ in their current values. P_{opt} depends on the way the neighborhood structure is defined. Finally, given that x' is the new proposal $P_{acceptance} = \frac{e^{\frac{f(x')}{T}}}{e^{\frac{f(x')}{T}}+e^{\frac{f(x)}{T}}}$, this probability depends on T and the values of the function.

If we assume that the temperature goes to infinity, a transition will be always accepted if the new proposal has a better fitness function. Then the probability of hitting the optimum is equal to the probability of selecting the optimum as the next proposal. If the current solution has unitation u, and $n-u > s$, the probability of generating the optimum as the new proposal is 0 because in this case it is impossible to make the transition in just one step. Thus, for $n-u \leq s$, and infinite temperature, P_{trans} can expressed as:

$$P_{trans} = \frac{\binom{u}{s-(n-u)}}{\binom{n}{s}} \cdot P_{opt} \tag{16}$$

Corollary 2. *If $P_{opt} = 1$ then equation (16) gives the maximum probability that a random neighborhood based search algorithm with neighborhood size equal s reaches the optimum in one step.*

$P_{opt} = 1$ implies that once the $(n-u)$ variables that need to be changed are included in the set of s variables selected then the transition to the optimum will be done. This bound of P_{trans} can be improved only if the s variables are not selected at random, for instance, in the case that problem information could be used to select the variables to be changed.

Now let us consider P_{opt} for $Neig1$ and $Neig2$ introduced in section (2.2.2). In $Neig1$ an assignment for the s variables is randomly generated. This is:

$$P_{trans}^{Neig1} = \frac{\binom{u}{s-(n-u)}}{\binom{n}{s}} \cdot \left[\frac{1}{2^s}\right] \tag{17}$$

This would correspond to a search algorithm with uniform transition rules. But uniform transition rules do not imply a uniform search of the space. For the particular case of uniform search, the $Neig2$ case, the probability of selecting a new assignment for the s variables must satisfy that all the points of the neighborhood are visited with the same probability.

Let us consider the probability of having a neighbor y of x whose Hamiltonian distance from x is h (i.e. $H(x,y) = h$). Obviously, the probability for $h > s$ is zero. For $h < 0$ this probability is:

$$P_{H(x,y)=h} = \frac{\binom{n}{h}}{\sum_{h'=0}^{s}\binom{n}{h'}} \tag{18}$$

Then P_{opt}^{Neig2} has to be calculated according to the distance to the optimum h_{opt} . If the current solution has unitation u then $h_{opt} = n - u$ and

$$P_{opt}^{Neig2} = \frac{\binom{n-u}{h}}{\sum_{h'=0}^{s}\binom{n}{h'}} \cdot \frac{1}{\binom{s}{n-u}} \tag{19}$$

where the first term in the expression corresponds to the probability of changing exactly $n - u$ components while the second expression is the probability of finding the right $n - u$ variables among the s variables selected. Substituting (19) in (16) we get:

$$P_{trans}^{Neig2} = \frac{\binom{u}{s-(n-u)}}{\binom{n}{s}} \cdot \frac{\binom{n}{n-u}}{\sum_{h'=0}^{s}\binom{n}{h'}} \cdot \frac{1}{\binom{s}{n-u}} \tag{20}$$

Considering the case when $s = n - u + a$, $a \geq 0$ and substituting in (20) we arrive to:

$$P_{trans}^{Neig2} = \frac{\binom{u}{a}}{\binom{n}{n-u+a}} \cdot \frac{\binom{n}{n-u}}{\sum_{h'=0}^{s}\binom{n}{h'}} \cdot \frac{1}{\binom{n-u+a}{n-u}} \tag{21}$$

$$= \frac{u!}{(u-a)!a!} \cdot \frac{(n-u+a)!(u-a)!}{n!} \cdot \frac{n!}{(n-u)!u!)} \cdot \frac{(n-u)!a!}{(n-u+a)!} \cdot \frac{1}{\sum_{h'=0}^{n-u+a}\binom{n}{h'}} \tag{22}$$

$$= \frac{1}{\sum_{h'=0}^{n-u+a}\binom{n}{h'}} \tag{23}$$

Corollary 3. *P_{trans}^{Neig2} is maximal when $a = 0$ (i.e. $s = n - u$).*

If $a > 0$ then additional terms are added to the denominator decrementing the probability, on the other hand a can not be less than zero because then it will be impossible to reach the optimum in just one move.

Summarizing: We have shown which is the maximum probability for a random neighborhood based search algorithm with neighborhood size s to reach the optimum in one step. This formula allows to estimate the average number of steps needed by the algorithm to jump from a local suboptimum of unitation u to the optimum. We have also presented the formulae for calculating this probability for two commonly used neighborhood structures, $Neig1$ and $Neig2$. In the case of $Neig2$ we have also shown that the optimal choice of the neighborhood size is $s = n - u$.

5 Numerical Results

In the first experiments we study the behavior of the CFTP to sample the Boltzman distributions of functions with different levels of difficulty for population based optimization methods.

A second class of experiments focus in the efficiency of the CFTP method as an optimization technique. We compare its results with a Gibbs sampling based optimization algorithm. After the completion of these experiments we show that the number of evaluations to reach the optimum can be estimated in terms of the probability of the optimum and the average number of transition to coalescence. Finally we propose different ways of reducing the number of function evaluations. Our proposals are based on MCMC methods that simultaneously update more than one variable at every step. These methods have a faster mixing time, allowing to diminish the number of function evaluations needed to reach the optimum. For these algorithms we also analyze the influence of the neighborhood structure. In the experiments conducted we deal with the maximization of the functions.

5.1 Deceptive functions of order 3

A function of unitation is a function whose value depends only on the number of ones in an input string. The function values of the strings with the same number of ones are equal. In Evolutionary Computation unitation functions are commonly employed to evaluate the efficiency of the optimization methods for diverse dimensions of difficulty of the problems. We first investigate the deceptive function f^3_{dec}.

$$f^3_{dec} = \begin{cases} 2 & for \quad u = 0 \\ 1 & for \quad u = 1 \\ 0 & for \quad u = 2 \\ 3 & for \quad u = 3 \end{cases} \tag{24}$$

Before applying the CFTP algorithm the necessary conditions have to be verified. In fact, for many functions it could be very difficult to show that such conditions are satisfied. First an ordering of space has to be found where the transition function is monotone. In the case of f^3dec and many other binary functions the inclusion criteria can be used to order the states where $x \leq y$ if $x_i \leq y_i, \forall i \in S_2$. The null and ones vectors are respectively the smallest and highest elements of this ordering.

To demonstrate that $f_\phi(x^t, U, W)$ is monotone for f^3_{dec} it is sufficient to show that if $(0, a, b) \leq (1, c, d)$ then $f_\phi((0, a, b), U, W) \leq f_\phi((1, a, b), U, W)$. Note that f^3_{dec} is a unitation function and thus it is invariant to a permutation of the vector x. In table 1 the joint and conditional probabilities are shown for the Boltzman distribution of the f^3_{dec} and $\beta = 1$. Using the Gibbs Sampling transition rules in (8) and conditional probabilities in table 1 it is possible to confirm that the function is monotone and to make a run of the CFTP algorithm (Table 2). In this table H represents the Hamiltonian distance between x and y, α is the value of the random constant and i the position selected for updating.

The algorithm starts at time -1, random values are chosen for U and W. The transitions at point 2 keeps the vectors in the same configuration at time 0, thus it is necessary to start simulation again now at point -2. This time there is a change in the state of the chain started at the smallest state. At time -1 the algorithm requires to apply the random values U and W already used at this time in the previous simulation. After applying the transition rules no change is evidenced, and because coalescence has not been reached, the Propp-Willson algorithm begins now again at time -4. In this attempt coalescence will be achieved and at time $t = 0$ the algorithm outputs 100 as the sampled point.

x	$f(x_1x_2x_3)$	$P(x_1x_2x_3)$	$P(x_1\|x_2x_3)$
000	2	0.1913	0.7313
001	1	0.0703	0.7307
010	1	0.0703	0.7307
100	1	0.0703	0.2687
110	0	0.0259	0.2692
101	0	0.0259	0.2692
011	0	0.0259	0.0475
111	3	0.5199	0.9525

Table 1: Joint and conditional probabilities for the function f^3_{dec} , $\beta = 1$.

t	-1	0	-2	-1	0	-4	-3	-2	-1	0
x, y	000	000	000	100	100	000	000	000	100	100
	111	111	111	111	111	111	110	100	100	100
H	3	3	3	2	2	3	2	1	0	0
α	0.45		0.78	0.45		0.001	0.03	0.78	0.45	
i	2		1	2		3	2	1	2	

Table 2: Example of a run of the CFTP algorithm for the function f^3_{dec}, $\beta = 1$

In this section we will also use the $f_{3deceptive}$ function (25), this is defined as a sum of more elementary deceptive functions f^3_{dec} of 3 variables.

$$f_{3deceptive}(X) = \sum_{i=1}^{i=\frac{n}{3}} f^3_{dec}(X_{3i-2}, X_{3i-1}, X_{3i}) \tag{25}$$

Unit	P	P^*_{20}	P^*_{1000}	P^*_{10000}
0	0.00144	0.0	0.001	0.0013
1	0.00176	0.0	0.001	0.0025
2	0.00951	0.0	0.0	0.0005
3	0.03227	0.05	0.034	0.0331
4	0.02602	0.35	0.025	0.0283
5	0.00879	0.0	0.008	0.0074
6	0.23773	0.35	0.273	0.2349
7	0.09604	0.20	0.086	0.0936
8	0.01298	0.0	0.018	0.0129
9	0.58202	0.40	0.554	0.5855
diverg.		0.04770	0.00247	$4.30149E-5$

Table 3: Exact probabilities and CFTP approximations for the function $f_{3deceptive}$.

We first evaluate how exact is the approximation of the CFTP algorithm for the $f_{3deceptive}$, $n = 9$, as the number of points sampled is increased. Table 3 shows results, P is the exact condensed probability for the different unitation values of the function. This condensed probability is calculated by adding the probability of all the points with the same unitation. P^*_N is the probability calculated for N points sampled by the CFTP. The quality of the approximation (*diverg.*) is evaluated using the square of the difference between the two distributions. It gets better as the number of points in incremented. At the bottom of the table these values are shown.

For the $f_{3deceptive}$ function we evaluate how the temperature and the increment in the number of variables affect the probability of the optimum in the approximation of the Boltzman distribution given by the CFTP algorithm. Table 4 shows results achieved for this function when the number of variables (n) is increased. For each value of the inverse temperature β and each value n the CFTP algorithm is run $3n$ times. The probability of the optimum (P^*) is equal to the frequency of achieving coalescence at the optimum. For all values of n coalescence was achieved in the $3n$ experiments. In the table *trans.* is the average number of transitions to coalescence calculated from the $3n$ experiments.

It can be seen in the table that an increment of the inverse temperature also increases the probability of finding the optimum. However, the average number of transitions needed to reach coalescence gets higher with β too. The relation is nonlinear. Doubling the probability to generate the optimum by a higher β needs much more that twice the number of transitions.

n	$\beta = 1$		$\beta = 2$		$\beta = 3$	
	P^*	*trans.*	P^*	*trans.*	P^*	*trans.*
6	0.33	306	0.66	1394	0.94	10866
12	0.05	825	0.52	4267	0.83	25600
18	0.02	1394	0.42	7908	0.68	48242
24	0.03	1934	0.29	12174	0.58	81692
30	0.01	2079	0.12	18204	0.56	102309

Table 4: Influence of n and β in the CFTP approximation for function $f_{3deceptive}$.

5.2 General deceptive function

$$Gendec_n = \begin{cases} n & \text{for} \quad u = n \\ n - u - 1 & otherwise \end{cases} \tag{26}$$

n	$\beta = 1$			$\beta = 2$			$\beta = 3$		
	C	P^*	*trans.*	C	P^*	*trans.*	C	P^*	*trans.*
6	18	0.11	494	18	0.44	1756	18	0.83	153600
12	36	0.03	59307	36	0.06	473543	0	0.0	0
18	54	0.0	$5.70406E6$	1	1.0	$3.35544E7$	0	0.0	0

Table 5: Influence of n and β in the CFTP approximation for function $Gendec_n$.

We analyze the case of the general deceptive function (26). For this function the influence of β and n in the performance of the CFTP algorithm were studied. Results are shown in table 5. The distance between the optimum and the sub-optimum of $Gendec_n$ increases with n and so the difficulty of the function. As the number of variables increases the behavior of the CFTP algorithm deteriorates. In the table C refers to the numbers of times that coalescence has been achieved, a maximum number of 2^{25} transitions were allowed. If the inverse temperature is increased the CFTP algorithm will take a huge number of steps to reach coalescence. The experiment shows that for the difficult $Gendec_n$ function, the CFTP algorithm might not be able to converge with less than an exponential (in n) number of transitions.

Although previous experiments give an idea that both (the probability of the optimum and the number of transitions) depend on β it is not clear how to make a choice of β so that the trade off between the probability of the optimum and the number of transitions can be optimized (i.e. the number of function evaluations to reach the optimum could be minimized). We investigate this issue at the end of this section.

5.3 Jump function

$$Jump(n, gap, i) = \begin{cases} i & i < n - gap \\ 2 * (n - gap - 1) - i & n - gap \leq i \\ n & i = n \end{cases} \quad (27)$$

In the following experiments our purpose is twofold. To compare the behavior of two optimization algorithms, based on the CFTP and the Gibbs Sampling, in terms of the number of function evaluations they make; and to do this comparison for a function of tunably difficulty. We use the *Jump* function (27) that was introduced in [20]. The parameter gap of this function defines the number of steps one has to go downhill in order to reach the unique maximum. For $gap = 0$ we have the very simple *Onemax* function. As the parameter gap increases so does the difficulty of the function.

n	β	g	CFTP		GS		g	CFTP		GS	
			C	τ	C	τ		C	τ	C	τ
8	1.0	1	10	2124	50	216	3	10	9472	50	3113
8	2.0	1	10	1664	50	321	3	10	125795	50	45557
8	3.0	1	10	2662	50	753	3	10	$209715E6$	50	781169
16	1.0	1	10	69107	50	2951	3	10	584883	50	110218
16	2.0	1	10	9446	50	2084	3	10	$5.27565E6$	50	$1.10424E6$
16	3.0	1	10	10880	50	3876	3	10	$3.73817E7$	45	$1.52393E7$
24	1.0	1	3	511386	50	39276	3	3	$3,60018E6$	50	$1,87867E6$
24	2.0	1	10	36864	50	7413	3	10	$3.45522E7$	50	$8.07421E6$
24	3.0	1	10	30873	50	8003	3	10	$4.32852E8$	13	$2.86155E7$
32	1.0	1	0	0	50	366433	3	1	$4.64451E6$	37	$1.8411E6$
32	2.0	1	10	113306	50	25289	3	10	$1.6549E8$	30	$2.27484E7$
32	3.0	1	10	100045	50	14595	3	10	$1.5737E9$	6	$3.19828E7$

Table 6: CFTP and Gibbs Sampling for function *Jump* and different values of n and β.

We study the performance of the algorithms for different values of the parameter *gap* and of the inverse temperature. Results are shown in table 6. Ten runs of The CFTP based optimization algorithm described before were done. For function *Jump* we used a maximum number of attempts to find the optimum in each run equal 1000. In the table C refers to the numbers of times the optimum was found in the 10 experiments, the next columns indicate the total number of transitions, and the average passage time τ, i.e. the average number of evaluations needed before to reach the optimum. For each n and *gap* it can be recognized a bounded region where the optimal β lies. In the case of $gap = 1$, $\beta = \infty$ seems to be the best option. However, this is not the case when the gap parameter is changed. The number of evaluations needed by the Gibbs Sampling algorithm is less than that needed by the CFTP algorithm. This could be explained by the fact that the CFTP algorithm needs two evaluations for each transition. An extra burden of evaluations comes as a requirement of the time schedule used, it goes back to time $-2T$ every time that coalescence has not been achieved.

In Table 7 we present results of the Propp-Wilson algorithm for the *Jump* function ($n = 8$) with different values of the gap parameter. The probability of the best and the number of transitions to coalescence are calculated as the average of 1000 runs of the algorithm. Another independent 1000 runs are used to calculate the passage time. The experiments support evidence that the passage time can be estimated as follows.

$$\tau = \frac{1}{P(opt)} \dot{trans} \tag{28}$$

Thus, in principle there are two ways of reducing the number of evaluations:

- To increase the probability of the optimum.
- To reduce the number of transitions.

As we have seen in the experiments for the CFTP shown before, by varying the temperature we can achieve an increment in the probability, but also an increment in the number of transitions is experienced. In the next subsection we present an alternative that reduces the number of transitions without affecting the probability of the optimum. What we do is to find a way to reduce the mixing time of the Markov chains.

	gap 0			gap 1		
β	P(opt)	trans.	τ	P(opt)	trans.	τ
1.0	0.081	118	1523	0.102	236	2013
1.5	0.199	116	598	0.301	460	1290
2.0	0.362	116	317	0.589	807	1270
2.5	0.531	118	217	0.814	1394	16801
3.0	0.677	115	172	0.927	2138	2337
	gap 2			gap 3		
β	P(opt)	trans.	τ	P(opt)	trans.	τ
1.0	0.164	716	4170	0.296	3084	8876
1.5	0.525	3172	5864	0.804	25154	32101
2.0	0.851	11396	13482	0.971	112949	114996
2.5	0.965	31416	33541	0.996	498336	503817
3.0	0.992	85004	82080	0.999	$206701E6$	$200532E6$

Table 7: Exact probabilities and average passage time to reach the optimum of the *Jump* function ($n = 8$) for the CFTP algorithm.

5.4 Experiments for the Blocked Gibbs Sampling

Now we test the usual Gibbs Sampling algorithm (algorithm 1), the blocked Gibbs Sampling algorithm with random blocks and the version that defines the neighborhood

based on the structure of the function in the problem of approximating the stationary Boltzman distribution of the $f_{3deceptive}$ deceptive function, $n = 9$. For such a small number of variables it is possible to calculate the exact stationary distribution π and the error of our approximation. In our simulation $\beta = 2$, the burn-time was set to 0, it means that all the points generated by the Markov chain are used in the approximation.

The Gibbs Sampling algorithms with local updates and random blocks are implemented as it has been previously described. For the third algorithm we determined the probability of a variable to be included in the block in the following way. First, a variable x_i is randomly selected with uniform distribution. The neighborhood of x_i is a set s_i defined in an additively decomposed function. To the effect of the neighborhood the extremes of the vector join to form a ring. The two different ways of defining the transitions, $Neig1$ and $Neig2$, are evaluated in the experiments.

It is clear that for $f_{3deceptive}$ the probability of having s related variables in a block using this method is higher than when blocks are selected at random. This method can be applied uniquely when related variables are sequentially ordered like in the case of the deceptive function we have shown. The important point, however is that when some information about the dependencies between variables is available it can be used to form the blocks, even if it is not exact. Note that in our method to form the blocks there is always the possibility to include in the blocks variables that are not related.

In figure 1 are presented the results for this function, they are an average of 100 runs of a Gibbs Sampling started at a random point. The graph represents the variation distance after every 10 steps of the algorithm.

$Unit$	P	P_{GS}	$P_{RBGS-N1}$	$P_{NBGS-N1}$	$P_{RBGS-N2}$	$P_{NBGS-N2}$
0	0.00144	0.0	0.00030	0.00043	0.0	0.00133
1	0.00176	0.0	0.00373	0.00100	0.00293	0.00060
2	0.00095	0.0	0.00053	0.00120	0.00153	0.00083
3	0.03227	0.03227	0.03203	0.03530	0.02933	0.02633
4	0.02602	0.02957	0.02237	0.02960	0.02790	0.02807
5	0.00879	0.00980	0.00847	0.00657	0.01157	0.00713
6	0.23773	0.21240	0.22390	0.23467	0.26363	0.23340
7	0.09604	0.08650	0.09240	0.08847	0.11010	0.10113
8	0.01298	0.01363	0.01553	0.01467	0.01473	0.01030
9	0.58202	0.59553	0.60073	0.58810	0.53827	0.59087
$var.dist.$		0.10856	0.08367	0.06348	0.08198	0.05118

Table 8: Exact probability and different Gibbs Sampling approximations for the function $f_{3deceptive}$.

The first thing that can be appreciated in the figure is that all the 5 algorithms have a tendency to minimize the error. This coincides with values for the approximate probability σ we have obtained and are shown in Table 8. The second aspect is that the ordinary Gibbs Sampling has the slowest mixing time. To reach an error of 0.2 it

Figure 1: Total variation distance of the different Gibbs Sampling Algorithms in the approximation of the Boltzman distribution of function $f_{3deceptive}$, $n = 9$.

requires almost the double of time than both of the random blocked Gibbs Sampling algorithms. The non random blocked Gibbs sampling algorithms are the best in this case. In the comparison between $Neig1$ and $Neig2$ better results are achieved by the second algorithm, although the difference is small.

5.5 Approximation of the Boltzmann distribution for the Jump function

We consider now the function $Jump$ $(n = 9, gap = 2)$. The Boltzman distribution corresponding to this function $(\beta = 2)$ is sampled using the classical Gibbs Sampling algorithm and the version that uses random blocks. The algorithm that finds blocks based on the neighborhood is of no application here. The random blocked algorithms were tested for $Neig1$, $Neig2$ and 2 values of s, $(s = 2, 4)$. In figure 2 are presented the results that are also an average of 100 runs. The graph represents the variation distance after every 10 steps of the algorithm.

Figure 2: Total variation distance of the different Gibbs Sampling Algorithms in the approximation of the Boltzman distribution of function $Jump$, $n = 9$, $gap = 2$.

In this case the difference between the approximation of the Gibbs Sampling and the blocked Gibbs Sampling algorithms is more evident. These results could look quite impressive but there will not if we consider that it is very difficult for an algorithm like

the ordinary Gibbs Sampling that only changes one variable at the time to scape from the points with unitation 6 and 9. In the figure can be also observed that best results are obtained with $s = 4$. When $s = 2$ the performance of $Neig2$ is clearly superior to $Neig1$.

5.6 Blocked MCMC algorithms for optimization

Now we present results in the optimization of the $f_{3deceptive}$ function using the blocked Gibbs Sampling with random and neighborhood based selected blocks. As before, the chain simulation is stopped when the optimum is found. Experiments are done for different values of n and β. In table 9 a comparison among the three algorithms is presented. Local (ordinary Gibbs Sampling), Random and Neighborhood blocked Gibbs Sampling. For the blocked Gibbs Sampling algorithms values of s between 2 and 6 were evaluated. For each value of n and β the best result in terms of the average passage time is presented with the size of the neighborhood it was achieved with.

n	β	Local	Random		Neighborhood	
		trans.	s	*trans.*	s	*trans.*
9	1	253	3	219	3	163
9	2	672	5	245	3	139
9	3	3444	5	266	3	148
12	1	528	2	524	3	372
12	2	1142	4	538	3	232
12	3	5174	5	705	3	229
15	1	1056	2	1062	2	703
15	2	1768	4	1092	3	327
15	3	7902	4	1501	3	345
18	1	1863	3	2478	2	1326
18	2	2289	3	1775	3	456
18	3	10022	5	2428	3	475

Table 9: Comparison for the 3 different versions of the Gibbs Sampling algorithm for the $f_{3deceptive}$ function when n and β are increased.

As expected best results are obtained with the neighborhood based algorithm, and in almost all the cases they were achieved when $s = 3$, i.e. when the size of the block was equal to the size of the set of related variables. For the random blocked algorithm results change. First, it seems to be that when β is increased better results are achieved by increasing the neighborhood size. Second, the best value of β for the random blocked Gibbs Sampling algorithm is different to the best inverse temperature for the Neighborhood algorithm. Third, results are better than when local updated is employed, however, the difference for some values of n is small.

The issue of the relationship between the inverse temperature and the neighborhood

beta	*gap*	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
0.1	1	861	801	765	837	824
0.9	1	351	157	216	317	430
1.4	1	330	102	162	276	392
1.8	1	387	80	154	273	388
0.1	2	993	845	785	809	789
0.5	2	913	546	385	429	487
0.9	2	1270	553	248	320	399
0.13	2	2277	669	198	289	406
0.1	3	1099	927	822	740	794
0.5	3	1767	943	650	455	482
0.9	3	4233	1692	831	381	435
0.13	3	11163	3176	1000	312	388

Table 10: Number of transitions for function $Jump$, $n = 9$ when β and s are changed

size is investigated in detail for function $Jump$, $n = 9$. Table 10 presents the results of several runs of the random blocked Gibbs Sampling when β and s are changed. $Neig2$ was used in this case. It can be appreciated in the table that if s is under a critical threshold (in the case of function $Jump$ this critical value is related to the parameter gap) then small β is better than high β. On the other hand, search done using high neighborhood sizes is generally improved when a high inverse temperature is used. The inverse temperature and the neighborhood size are not the only important components of local search algorithms. In order to carry out an efficient search another relevant aspect has to be considered. This is the way in which transitions to the neighborhood states have to be done. We consider this issue in the next section.

5.6.1 Structure of the neighborhood, $Jump$ function

We investigate now the role of the neighborhood structure in the case of the $Jump$ function. For this function most of the steps spent by a neighborhood search algorithm with neighborhood size s are used to pass from a local optimum with unitation $n-gap-1$ to the optimum of unitation n.

Using (16), and (23) we estimate the total number of steps to reach the optimum of the $Jump$ function with gap equal gap when $Neig1$ and $Neig2$ are used. We substitute u by $n - gap - 1$ and s by $gap + 1 + a$.

$$N_{Neig1} = \frac{\binom{n}{gap+1+a}}{\binom{n-gap-1}{a}} \cdot 2^{gap+1+a} \tag{29}$$

$$N_{Neig2} = \sum_{h'=0}^{gap+1+a} \binom{n}{h'} \tag{30}$$

For the *Jump* function it is clear that the minimal number of steps to reach the optimum needed by the $Neig2$ (equation 30) is achieved when $s = gap + 1$, otherwise the number of steps is incremented.

n	g	τ	N_{Neig2}	g	τ	N_{Neig2}	g	τ	N_{Neig2}
8	1	47.6	37.0	2	96.7	93.0	3	164.6	163.0
16	1	191.0	137.0	2	718.6	697.0	3	2538.5	2517.0
24	1	430.0	301.0	2	2379.1	2325.0	3	13026.6	12951.0
32	1	764.6	529.0	2	5590.5	5489.0	3	41633.0	41449.0
64	1	3059.4	2081.0	2	44202.2	43745.0	3	680863.5	679121.0

Table 11: Comparison of the simulation results τ for the function *Jump* with the expected number of steps N_{Neig2}

The expected number of steps N_{Neig2} for the *Jump* function and different number of variables are presented in table 11. The predictions are compared with results of the simulations appeared in [20] . As N_{Neig2} is just the number of the steps required to jump from the local optimum to the optimum, it is only a lower bound of the expected passage time τ. Nevertheless it can be appreciated in the table how close is the prediction.

To analyze the case of $Neig1$ we consider a simplification of the formula (29) for N_{Neig1}.

$$N_{Neig1} = \frac{n!a!(n-(gap+1+a))!}{(n-(gap+1+a))!(gap+1+a)!(n-gap-1)!}2^{gap+1+a} \tag{31}$$

$$= \frac{n!a!}{(gap+1+a)!(n-gap-1)!}2^{gap+1+a} \tag{32}$$

$$= \frac{a!(n-gap-1)!\prod_{i=1}^{gap+1}(n-gap-1+i)}{(n-gap-1)!a!\prod_{i=1}^{gap+1}(a+i)}2^{gap+1+a} \tag{33}$$

$$= \left(\prod_{i=1}^{gap+1}\frac{n-gap-1+i}{a+i}\right)2^{gap+1+a} \tag{34}$$

In (34) it can be seen that the number of steps depends on two terms. When a is increased the first term decreases but the second one gets exponentially higher. When $a = n - gap - 1$ the first term is 1 and N_{Neig1} is equal to the size of the space. If $a = 0$ then the number of steps becomes:

$$N_{Neig1} = \binom{n}{gap+1}2^{gap+1} \tag{35}$$

and this value can be even higher than 2^n. So, for the *Jump* function the $Neig2$ makes a more efficient search. Figure 3 shows how N_{Neig1} and N_{Neig2} scale when the number of variables is increased for the *Jump* function, $gap = 1, 5$. The size of neighborhood for

$Neig1$ and $Neig2$ are respectively $(2gap + 1)$ and $(gap + 1)$. In the figure the y axes is logscaled, vertical lines show the first n for which the computation of the number of steps is possible $(n > s)$. It can be seen in the figure that the number of steps is always higher for $Neig2$. The difference between the number of steps needed by both algorithms can be intuitively appreciated if we realize that the average number of variables that change their value in every step of the $Neig1$ is less than the average for $Neig2$. As a consequence the first algorithm needs more steps for finding a way to cross the gap. An important conclusion of our analysis is that the way transition probabilities of the neighborhood are defined is as critical for the efficiency of the search as the own choice of the neighborhood size is.

Figure 3: Expected number of steps of the random blocked Gibbs Sampling for the $Jump$ function when n is increased.

6 Conclusions

References

[1] G. T. Barkema and M. E. J. Newman. New Monte Carlo algorithms for classical spin systems. Tech. Rep. No. SFI-97-03-026, Santa Fe Institute, 1997.

[2] Julian Besag. Markov Chain Monte Carlo for statistical inference. Technical Report Working paper No. 9, Center for Statistical and Social Science, University of Washington, September 2000.

[3] S. Chib and B. Carlin. On MCMC sampling in hierarchical longitudinal models. *Statistics and Computing*, 9:17–26, 1999.

[4] Persi Diaconis and David Freedman. Iterated random functions. *SIAM review*, 41(I):45–76, 1999.

[5] Ola Elerian, Siddharta Chib, and Neil Shephard. Likelihood inference for discretely observed non linear diffusions. *Econometrica*, 2001.

[6] James Allen Fill, Motoya Machida, Duncan J. Murdoch, and Jeffrey S. Rosenthal. Extension of Fill's perfect rejection sampling algorithm to general chains. *Random Structures Algorithms*, 17:290–316, 2000.

[7] J.A. Fills. An interruptible algorithm for perfect sampling via markov chains. *Annals of Applied Probabilities*, pages 131–162, 1998.

[8] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *IEEE transactions on pattern analysis and Machine Intelligence*, (6):721–741, 1984.

[9] C. Geyer. Markov chain monte carlo maximum likelihood. In *Proceedings of the 23rd Symposium on the Interface*, pages 156–163, 1991.

[10] M. Harvey. Monte carlo inference for belief networks using coupling from the past. Master's thesis, University of Toronto, Department of Computer Science, 1999.

[11] O. Häggström and K. Nelander. On exact simulation of markov random fields using coupling from the past. *Scandinavian Journal of Statistics*, 26(3):395–411, 1999.

[12] K. Kask and R. Dechter. GSAT and local consistency. In *Proceedings of the 14th IJCAI*, pages 616–622, Montreal, Canada, 1995.

[13] S. Kirpatrick, C. D. Jr. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[14] Jun S. Liu. Markov chain monte carlo and related topics. Technical report, Department of Statistics, Stanford University, 1999.

[15] Jun S. Liu, A. Kong, and W.H. Wong. Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika*, 81:27–40, 1994.

[16] D. Mackay. *Learning in Graphical Models*, chapter Introduction to Monte Carlo methods. MIT Press, 1998.

[17] E. Marinari and G. Parisi. Simulating tempering - a new monte-carlo scheme. *Europhysics Letters*, 19(6):451–458, 1992.

[18] Heinz Mühlenbein and Thilo Mahnig. Convergence theory and applications of the Factorized Distribution Algorithm. *Journal of Computing and Information Technology*, 7(1):19–32, 1998.

[19] Heinz Mühlenbein, Thilo Mahnig, and Alberto Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.

[20] Heinz Mühlenbein and J. Zimmermann. Size of neighborhood more important than temperature for stochastic local search. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 1017–1024, La Jolla Marriott Hotel La Jolla, California, USA, 2000. IEEE Press.

[21] J.G. Propp and D.B. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random structures and Algorithms*, (9):223–252, 1996.

[22] J.G. Propp and D.B. Wilson. Coupling from the past: a user's guide. *Microsurveys in Discrete Probability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 41:181–192, 1998.

[23] N. Shephard. Partial non-Gaussian state space. *Biometrika*, 81:115–131, 1994.

[24] N. Shephard and M.K. Pitt. Likelihood analysis of non-Gaussian measurement time series. *Biometrika*, 84(3):653–667, 1997.

[25] Elke Thönnes. Perfect simulation of some point processes for the impatient user. *Annals of Applied Probabilities*, pages 69–87, 1999.

[26] U. Wolff. Collective monte carlo updating for spin systems. *Physical Review Letters*, 62:361–364, 1989.

[27] Wing Hung Wong and Faming Liang. Dynamic weighting in Monte Carlo and optimization. *Applied Mathematics. Proceedings of the National Academic of Science*, 94:14220–14224, December 1997.