Blocked Stochastic Sampling versus Estimation of Distribution Algorithms

Roberto Santana [‡] and Heinz Mühlenbein [†]
[†] ICIMAF, Calle 15, e/ C y D, Vedado, CP-10400, La Habana, Cuba
[‡] AiS.FHG — Schloss Birlinghoven, D-53757 Sankt Augustin, Germany

Abstract - The Boltzmann distribution is a good candidate for a search distribution for optimization problems. In this paper we compare two methods to approximate the Boltzmann distribution - Estimation of Distribution Algorithm's (EDA) and Markov Chain Monte Carlo methods (MCMC). It turns out that in the space of binary functions even blocked MCMC methods outperform EDA on a small class of problems only. In these cases a temperature of $T = 0$ performed best.

I. Introduction

Let $\mathbf{x} = (x_1, \ldots, x_n)$ denote a vector, $x_i \in \{0, 1\}$. We use the following conventions. Capital letters X_i denote variables, small letters x_i assignments. For simplicity we only consider binary variables here. Let a function $f : \{0,1\}^n \to \mathbb{R}_{>0}$ be given. We consider the optimization problem $\mathbf{x}_{opt} = \text{argmax} f(\mathbf{x})$.

The Boltzmann distribution has been proposed as a search distribution for optimization problems a number of times. It is defined as follows:

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

where Z_T is the usual partition function. In this paper we will mainly use the inverse temperature $\beta = \frac{1}{T}$ as a parameter of the search algorithms. If search 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* [9]. The Boltzmann Estimation Distribution Algorithm is defined in algorithm 1.

The convergence of this algorithm to the set of global optima can easily be shown. The problem is to compute efficiently the Boltzmann distribution [7], [9]. The most simple approximation *UMDA* uses only univariate marginal frequencies [6]. From this algorithm *FDA*, the

Algorithm 1: **BEDA**

```
1  $t \Leftarrow 0$. Generate $N$ points according to the uniform distribution $p(\mathbf{x}, 0) = 2^{-n}$.
2  With a given $\Delta\beta(t) > 0$, let
       $p^s(\mathbf{x}, t) = \frac{p(\mathbf{x}, t) e^{\Delta\beta(t) f(\mathbf{x})}}{\sum_y p(\mathbf{y}, t) e^{\Delta\beta(t) f(\mathbf{y})}}$.
3  Generate $N$ new points according to the distribution $p(\mathbf{x}, t+1) = p^s(\mathbf{x}, t)$.
4  $t \Leftarrow t + 1$.
5  If stopping criterion not met go to STEP 2
```

factorized distribution algorithm and *LFDA* are derived. *LFDA* computes the factorization out of the sampled data. The interested reader is referred to [8]

In this paper we will investigate another techniques to approximate the Boltzmann distribution. These techniques are based on *stochastic sampling.*

II. Stochastic sampling techniques

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. These 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 at any point x, and after a sufficiently long time (say for k steps) take each state of the chain as a sample of the desired probability π. Therefore these algorithms are called Markov chain Monte Carlo methods MCMC.

Most algorithms use a sufficient large k in order to be sure that the chain has reached the equilibrium. We started our investigation with newly developed *exact* MCMC methods. A typical example is the coupling from the past algorithm from Propp and Wilson [11]. This algorithm uses a test to decide if the chain has reached equilibrium. To our great disappointment it turned out

E-mail: [†] rsantana@cidet.icmf.inf.cu, [‡] muehlenbein@gmd.de

that for optimization the approximate MCMC methods performed better. Thus we do not include these results here.

The performance of any MCMC method further depends on the temperature. If a small T is used, it takes a very long time to reach equilibrium. If a large T is used, the optima will be generated very seldomly. *Simulated annealing* [4] circumvents this problem, but at the expense of finding a suitable annealing schedule. Again, using some of the popular annealing schedules gave for our benchmark problems in almost all cases worse results than using a fixed temperature. This paper gives some explanation for this result.

The well known Gibbs sampler is described in [3]. It is also known as the heat bath algorithm for discrete problems. 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 value of the variable at this position is selected using the conditional probabilities of the chosen variables given the values assigned to the rest of variables. In this way the new vector x^{t+1} is constructed.

The performance of updating a single variable at a time MCMC samplers deteriorate for problems with highly correlated components [5]. 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 [2]. This technique is called "blocking" and when it is used with Gibbs Sampling the algorithm is usually called blocked Gibbs Sampling.

In this paper we present results in the use of the blocked Gibbs Sampling for the optimization 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.
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 $\pi$ using the sample set $A$.
```

In algorithm 2 we present our blocked Gibbs Sampler. The algorithm receives as a parameter the block size s, i.e. the maximum number of variables that can 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 input of the algorithm is the time (k) the Markov chain is run before sample points are taken and the maximum number of steps N_s. $\phi(x^t, U, W, Ng)$ is the transition rule for the state x^t, where Ng is the set of variables to be updated and U and W are random variables, that are used to determine the new configuration of Ng, and if the new point is accepted or not. The transition rule is shown in equation (2) where T is the fixed temperature. The determination of the proposal comprises two steps. First, one determines the set of variables to be updated, and second, one selects the new values for this set of variables.

$$x^{t+1} = \begin{cases} x_j^{t+1} = x_j^t & : \quad for\ j \notin Ng \\ x_{Ng}^{t+1} = x'_{Ng} & if \quad W \leq \frac{e^{\frac{f(x')}{T}}}{e^{\frac{f(x')}{T}} + e^{\frac{f(x)}{T}}} \\ x_{Ng}^{t+1} = x_{Ng}^t & \quad otherwise \end{cases} \tag{2}$$

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 our experiments we considered two versions of the blocked Gibbs Sampling. In the first, the set of variables N_g is selected from the vector with uniform distribution. In another version 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. The way in which transitions are done has been implemented as in [10], the probability of taking a new value x'_{Ng} depends on the Hamiltonian weight of the final solution.

III. Easy functions for population search

The following functions were used in our first group of experiments ($u = \sum x_i$):

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

$$f_{3deceptive}(X) = \sum_{i=1}^{i=\frac{n}{3}} f_{dec}^3(X_{3i-2}, X_{3i-1}, X_{3i}) \tag{4}$$

$$Jump(n, g, x) = \begin{cases} u & u < n - g \\ 2 * (n - g) - u & n - g \leq u \\ n & u = n \end{cases} \tag{5}$$

The $f_{3deceptive}$ function (4) is defined as a sum of more elementary deceptive functions f^3_{dec} of 3 variables. The *Jump* function (5) was used in [10], its parameter g defines the number of steps one has to go downhill in order to reach the unique maximum. For $g = 0$ we have the very simple *Onemax* function. As the parameter g increases so does the difficulty of the function.

The Gibbs Sampling algorithms with local updates and random blocks are implemented as it has been previously described. For the sampling algorithm that uses problem information we determine 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 for an additively decomposed function as all the variables which appear together in a sub-function.

For $f_{3deceptive}$ it is clear that using this method the probability of having s related variables in a block is higher than when blocks are selected at random. This method can be applied uniquely when the related variables are sequentially ordered. 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.

A. 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. The chain simulation is stopped when the optimum is found. Experiments are done for different values of n and β. In table I 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.

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 update is used, however, the difference for some values of n is small.

TABLE I
DIFFERENT VERSIONS OF THE GIBBS SAMPLING ALGORITHM FOR THE $f_{3deceptive}$ FUNCTION.

n	β	Local	Random		Neighborhood	
		trans.	s	*trans.*	s	*trans.*
12	1	528	2	524	3	372
12	2	1142	4	538	3	232
12	3	5174	5	705	3	229
18	1	1863	3	2478	2	1326
18	2	2289	3	1775	3	456
18	3	10022	5	2428	3	475

Table II shows the results of population based heuristics. By far the best results are obtained by *FDA*. The results of *UMDA* and *LFDA* are very similar. This indicates that the network computed by *LFDA* is too small and does not capture the real interactions [9].

TABLE II
RESULTS FOR THE $f_{3deceptive}$ FUNCTION.

n	*Alg.*	*Pop*	T	*succ.*	*Ave.gen.*	*eval*
12	*UMDA*	100	0.1	100	7.4	828
18	*UMDA*	100	0.1	100	88.7	8877
12	*LFDA*	100	0.1	100	11	1187
18	*LFDA*	100	0.1	100	87.2	8737
12	*FDA*	100	0.1	100	1.4	236
18	*FDA*	100	0.1	100	2.4	338

The results for the function *Jump* are still more in favor of population search. Whereas $UMDA$ solves these kind of problem in $O(n^{1.5})$ [6], scales even blocked Gibbs sampling like $O(n^{gap+1})$ [10].

Thus for these kind of problems, we find that blocked Gibbs Sampling is better than the local Gibbs Sampling. Nevertheless, the results are worse than those achieved with population based search methods. Therefore we tried to find functions where stochastic local search performs substantially better. It turned out that this was more complicated than expected.

IV. Easy functions for blocked Gibbs sampling

We start with the function F_{Block}. It is maximal when x contains only one block of B contiguous variables set to 1. The block can start at the end of the vector and continue at the beginning.

$$F_{Block}(x, B) = 2n - |B - G(x, B)| - |u(x) - G(x, B)| \quad (6)$$

Where $u(x) = \sum_{i=1}^{n} x_i$ and $G(x,B)$ is the size of the block with size closest to B in x.

The maximum of this function is $2n$, it is reached when the second and third terms in the expression are 0, i.e. there is only one block with size B. The first term is 0 when there is at least one block of size B. The second is 0 when there is only one block of ones in the vector. If $B \in [1, n-1]$ then $F_{Block}(x,B)$ is a multi-modal function with exactly n optima. Each optima corresponds to a block beginning at position $i \in [1,n]$.

TABLE III
EXAMPLE FUNCTION $F_{Block}(x,B)$

	$x = (10111011001)$				$x = (00110000000)$			
B	1	2	3	4	1	2	3	4
$G(x,B)$	2	2	3	3	2	2	2	2
$F_{Block}(x,B)$	15	17	18	18	21	22	21	20

Table III shows an example of the evaluation of this function for two vectors of size $n = 11$ and different values of the block size B.

We conducted a number of experiments to compare the behavior of the GS, the UMDA and the LFDA in the optimization of this function. In these experiments we have used GS with $\beta \mapsto \infty$, this algorithm is better known as local hill climber or $(1+1)-algorithm$ [1]. At every step a new move is proposed. It is accepted only if it is better or equal to the current one. Tables IV and V present the results of the experiments. In each case we have run 100 experiments, and in every case the optimum was found. The population size for the LFDA and the UMDA were of 100 and 50 points. In the tables, Ave. gen. is the average number of generation for the population based search methods, eval. is the average number of evaluation calculated for every set of 100 experiments.

TABLE IV
RESULTS FOR THE FUNCTION F_{Block}.

n	B	*Alg.*	*Trunc.*	*Ave.gen.*	*eval*
40	20	*GS*			281
40	35	*GS*			573
40	20	*LFDA*	0.1	12.3	1321
40	35	*LFDA*	0.1	31.1	3179
40	20	*UMDA*	0.02	17.2	894
40	35	*UMDA*	0.02	39.2	1971
40	20	*UMDA*	0.1	12.1	1300
40	35	*UMDA*	0.1	30.1	3078

In table IV it can be seen that $F_{Block}(X,B)$ is a very easy function for a single point searcher. Results for the EDAs are worse. One possible explanation of the behavior of the GS in this case is that for function $F_{Block}(X,B)$ a monotonic path can be constructed from every point of the search space to one of the optima by changing only one variable in every step. Let us explain this point.

Let us suppose that we start from a random solution x with one or none block. If x is not the optimum then at every step there exists at least one move that increments the value of the function. It is to set or reset (depending if the size of the block is smaller or higher than B) one of the variables at the extreme of the block. The probability of this move is $\frac{2}{n}$. On the other hand, if the initial point x has more than one block then every move that reduces the size of the blocks other than the one closest to B will increase the fitness of the function until the other blocks will eventually disappear and we will then be in the previously considered case.

Now we analyze why this function is difficult for the UMDA. In the first selected population of the UMDA there is the same probability of having solutions close to any of the optima. The expected univariate probabilities of all the variables is the same and the next population will have then solutions with more than one block or very large blocks. As B approaches n the probability of generating vectors with higher unitation is increased.This function is also deceptive for the LFDA.

The failure of the EDAs in this particular case is that they are unable to learn the constraint related to the maximum number of variables in the block, and they will focus on one of the optima only after a high number of evaluations have been done. Nevertheless, as a mutation effect is present due to the use of priors, the algorithm will find the optimum by means of the diversity injected by the local moves. In this case a very small selected set helps the algorithm to concentrate the search around only one of the optima. However, still the number of evaluations will be higher that in the case of the local hill climber.

TABLE V
RESULTS FOR THE FUNCTION F_{Block}, $B = n - 5$.

n	*Alg.*	*Trunc.*	*Ave.gen.*	*eval*
50	*GS*			934
100	*GS*			4397
200	*GS*			18755
300	*GS*			43054
50	*UMDA*	0.02	66.2	3292
100	*UMDA*	0.02	339.7	16698
200	*UMDA*	0.02	1553.1	76153
300	*UMDA*	0.02	3617.4	177302

In table V Gibbs sampling is compared with *UMDA* for different values of n. It turns out that both Gibbs sampling and *UMDA* scale as $o(n^2)$, but Gibbs sampling needs only 1/4 of the function evaluations. It has to be noted that *UMDA* with a population size of 50 and a truncation value of $\tau = 0.02$ selects just one individual. Thus it cannot estimate the univariate marginal distributions, but it heavily depends on mutation (Bayesian prior). This algorithm can be seen as a $(1,50)-$ *algorithm* adapted to discrete variables [1].

A. Hierarchical function $F^h_{Block}(x,B)$

Next we consider an extension of the function F_{Block} by means of a hierarchical function of only two levels. In this hierarchical function the initial vector x is divided in groups of k variables. The current assignment for each group variables determines the values for another single variable in the next level. In the first level are the input variables $(x_1,...,x_n)$. The binary value of variable y_i in the second level is determined from the set of variables $(x_{(i-1)k},...,x_{(i)k})$ of the first level. The mapping between $(x_{(i-1)k},...,x_{(i)k})$ and y_i is done by means of a contribution function g. We will call k as the order of the hierarchical function and we assume that n is a multiple of k. The F_{Block} function is evaluated on $(y_1,...,y_{\frac{n}{k}})$. The following function g is used to interpret a subset of variables $(x_i,...,x_{i+k})$ of the input vector x.

$$g(x_1,...,x_k) = \begin{cases} 1 & \text{for} \quad u(x_1,...,x_k) = k \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

We denote the hierarchical version of function F_{Block} as F^h_{Block}, and it is defined as follows.

$$F^h_{Block}(x,B) = k \cdot F_{Block}((y_1,...,y_{\frac{n}{k}}),B) \qquad (8)$$

The maximum value of $F_{Block}((y_1,...,y_{\frac{n}{k}}),B)$ is $\frac{2n}{k}$, we multiply by k to guarantee that F^h_{Block} reaches the same maximum $(2n)$ for any k. The range of feasible values for B is: $0 < B \leq \frac{n}{k}$.

Table VI presents the results achieved in the optimization of this function. In this case no all the runs were successful and the table includes the success rate (succ.) achieved in the 100 runs. The population size used for the UMDA and LFDA were the same as before. The truncation parameter for the UMDA was $T = 0.02$, and for the LFDA $T = 0.1$. As it can be seen, the performance of the GS for this function is as good as in the case of the F_{Block} function. The existence of a hierarchy does not make any change in its behavior. For this function the LFDA reaches the optimum more times and with less function evaluations than the UMDA. The LFDA is able to detect the hierarchical structure and to exploit it. As expected both EDAs are sensitive to an increment in the order of the function.

TABLE VI
RESULTS FOR THE FUNCTION F^h_{Block}

n	k	B	*Alg.*	*Ave.gen.*	*succ.*	*eval*
40	4	5	*GS*		100	107
40	4	9	*GS*		100	190
40	4	5	*UMDA*	194.3	100	9570
40	4	9	*UMDA*	667.8	96	32773
40	4	5	*LFDA*	6.4	100	733
40	4	9	*LFDA*	21.3	100	2214
40	8	4	*GS*		100	112
40	8	4	*UMDA*	1998.2	35	97960
40	8	4	*LFDA*	67.6	100	6797

B. Deceptive hierarchical function DF^h_{Block}

Finally we will introduce the function DF^h_{Block} that is the combination of F^h_{Block} and the following bimodal function:

$$bim(x,p) = n - min(x, n-u(x))^p \qquad (9)$$

The optimum of $bim(x,p)$ is reached for a vector with all the variables set to one, or all the variables set to zero. The parameter p determines the penalty given to points that are far from the optima. As p is increased higher is the gap from one optimum to the other.

$$DF^h_{Block} = F^h_{Block} + n - \sum_{i=1}^{i=\frac{n}{k}-1} bim((x_{(i-1)k},...,x_{ik}),p) \qquad (10)$$

The maximum of DF^h_{Block} is $3n$ and it is reached at the same points that are optima of F^h_{Block}. We have chosen function DF^h_{Block} because it combines two different sources of difficulty for optimization algorithms. The existence of a barrier that has to be crossed to reach the optima, and the presence of multiple optima. Notice that the second term of the function can be maximized for any combinations of null-blocks and one-blocks of order k. The first term is only maximized for certain combinations of null and one-blocks. One conflict arises when the second term has reached its maximum at a configuration that is not yet the optimum of the first term. In this case the optimization algorithm has to be able to decrease the value of the second term in order to reach the global optimum, and this is very difficult as is shown in table VII.

TABLE VII

RESULTS FOR THE FUNCTION DF^h_{Block}, $n = 40$, $k = 4$, $B = 9$

Alg.	p	β	*Ave.gen.*	*succ.*	*eval*
GS	1	∞		0	0
GS	1	1.16		100	1171
GS	5	0.51		1	499200
BGS	1	∞		97	198598
BGS	5	∞		93	221591
LFDA	1		890.9	9	88298
LFDA	5		577.3	6	57256

In the table are presented the results of the comparison between the GS, blocked GS and the LFDA for the function DF^h_{Block}, $n = 40$, $k = 4$ and $B = 9$. A maximum of 500000 evaluations are allowed to each run of the algorithms. For GS we have tried different values of the temperature. When $p = 1$ we have selected the temperature where less function evaluations are required to reach the optimum. For $p = 5$ the algorithm does not improve the results when the temperature is changed. Ave. is the average number of evaluations for the successful runs.

The difference in the behavior of the algorithms is very clear. EDAs cannot solve this problem, as already explained for F^h_{Block}. GS with infinite β can not solve it either because it can not escape from the local optimum. GS with a low β is the most interesting. When $p = 1$, a low β will allow the algorithm to reach the global optimum with much less functions evaluations than those needed for blocked GS. Nevertheless as p is increased changes in the inverse temperature will be useless.

The changes in the behavior of the algorithms due to changes in p can be explained as follows. When p is low, a number of consecutive moves away from the local optima can be accepted, increasing the probability of falling in a basin of attraction of the global optima. When p is increased, moves away from the local optimum are very rare, and the probability of falling in a basin of attraction of the global optimal is much smaller than that of returning to the previous point.

Blocked GS can solve this problem by jumping from the local optimum to the global optimum in only one transition. In this case the probability of hitting the optimum does not depend on p but on the size of the neighborhood. Nevertheless, even for the blocked GS the number of steps to reach the optimum increase exponentially with the distance between the optima.

V. Conclusions

We have shown that it is difficult to find in the space of binary functions optimization problems where stochastic sampling techniques outperform population search using a suitable search distribution substantially. We have defined such a class of problems. In these optimization problems there exist many paths leading to the optima, requiring either single bit or blocked bit flips.

In all cases considered, using a suitable neighborhood and accepting more or less only points of better or equal probability, performed better than using an annealing schedule for the temperature. This results confirms the theoretical analysis described in [10].

The class of problems where stochastic local search outperforms population search based on the Boltzmann distribution seems to be small, compared to the class where population search outperforms stochastic local search. This result confirms what has been already found in practice - to use stochastic local search not as an alternative to population search, but to use it as a hill-climber within population search.

References

[1] Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameteroptimization. *Evolutionary Computation*, 1:1–24, 1993.

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

[3] 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.

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

[5] 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.

[6] H. Mühlenbein and T. Mahnig. *Theoretical Aspects of Evolutionary Computing*, chapter Evolutionary Algorithms: From Recombination to Search Distributions, pages 137–176. Springer Verlag, Berlin, 2000.

[7] 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.

[8] Heinz Mühlenbein and Thilo Mahnig. Evolutionary computation and beyond. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, *Foundations of Real-World Intelligence*, pages 123–188. CSLI Publications, Stanford, California, 2001.

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

[10] 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.

[11] 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.