A Markov Network Based Factorized Distribution Algorithm for Optimization

Roberto Santana

Institute of Cybernetics, Mathematics, and Physics (ICIMAF),
Calle 15, entre C y D, Vedado, C-Habana, Cuba
`Rsantana@icmf.inf.cu`

Abstract. In this paper we propose a population based optimization method that uses the estimation of probability distributions. To represent an approximate factorization of the probability, the algorithm employs a junction graph constructed from an independence graph. We show that the algorithm extends the representation capabilities of previous algorithms that use factorizations. A number of functions are used to evaluate the performance of our proposal. The results of the experiments show that the algorithm is able to optimize the functions, outperforming other evolutionary algorithms that use factorizations.

Keywords: Genetic algorithms, EDA, FDA, evolutionary optimization, estimation of distributions.

1 Introduction

In the application of Genetic Algorithms (GAs) [4] to a wide class of optimization problems is essential the identification and mixing of building blocks. It has been early noticed that the Simple GA (SGA) is in general unable to accomplish these two tasks for difficult problems (e.g. deceptive problems). Perturbation techniques, linkage learners and model building algorithms are among the alternatives proposed to improve GAs. They try to identify the relevant interactions among the variables of the problem, and to use them in an efficient way to search for solutions.

Estimation Distribution Algorithms (EDAs) [10] are evolutionary algorithms that do not use the crossover and mutation operators. They construct in each generation a probabilistic model of the selected solutions. The probabilistic model must be able to capture a number of relevant relationships in the form of statistical dependencies among the variables. Dependencies are then used to generate solutions during a sampling step. It is expected that the generated solutions share a number of characteristics with the selected ones. In this way the search is led to promising areas of the search space. The interested reader is referred to [5] for a good survey that covers the theory, and a wide spectrum of EDAs applications.

One efficient way of estimating a probability distribution is by means of factorizations. A probability distribution is factorized when it can be computed as

the product of functions defined on subsets of the variables. A subclass of EDAs includes the algorithms that use factorizations of the probability distribution. In this paper we call to this subclass Factorized Distribution Algorithms (FDAs)¹ [9].

FDAs have outperformed other evolutionary algorithms in the optimization of complex additive functions, and deceptive problems with overlapping variables [9]. However, a shortcoming of FDAs is that the probabilistic model they are based on is constrained to represent a limited number of interactions. In this paper we investigate the issue of extending the representation capabilities of FDAs. To this end we introduce the Markov Network FDA (MN-FDA), a new type of FDA based on an undirected graphical model, and able to represent the so called "invalid" factorizations [9].

The paper is organized as follows. In section 2 we discuss the problem of obtaining a factorization of the probability. Section 3 presents the main steps for learning an approximate factorization from data. Section 4 explains the way the sampling step has been implemented. We introduce the MN-FDA in section 5. Section 6 presents the functions used in our experiments. We discuss the numerical results of the simulation. Section 7 analyzes the MN-FDA in the context of recent related research on evolutionary computation, we also present in this section the conclusions of our paper.

2 Factorization of a Probability

The central problem of FDAs is how to efficiently estimate a factorization of the joint probability of the selected individuals. To compute a factorization the theory of graphical models is usually employed. The following definitions will help in the explanation of our proposal.

Let $X = (X_1, X_2, \dots, X_n)$ represent a vector of integer random variables, where n is the number of variables of the problem. $x = (x_1, x_2, \dots, x_n)$ is an assignment to the variables, and $p(x)$ is a joint probability distribution to be modeled. Each variable of the problem has associated one vertex in an undirected graph $G = (V, E)$. The graph G is a conditional independence graph respect to $p(x)$ if there is no edge between two vertices whenever the pair of variables is independent given all the remaining variables.

Definition 1. *Given a graph G , a clique in G is a fully connected subset of V . We reserve the letter C to refer to a clique. The collection of all cliques in G is denoted as $\mathcal{C}$. C is maximal when it is not contained in any other clique. C is the maximum clique of the graph if it is the clique in $\mathcal{C}$ with the highest number of vertices.*

Definition 2. *A junction graph (JG) of the independence graph G is a graph where each node corresponds to a maximal clique of G , and there exists an edge*

¹ In the literature the term FDA is frequently used to name a particular type of Factorized Distribution Algorithms. Our definition covers it, and other algorithms that use factorizations.

between two nodes if their corresponding cliques overlap. A labeled junction graph is a JG that has an associated ordering of the nodes with a distinguished node called the root, and satisfies that a node belongs to the graph if at least one the variables in the cliques is not contained in the previous nodes in the ordering.

Definition 3. A junction tree (JT) is a single connected junction graph. It satisfies that if the variable X_k is a member of the junction tree nodes i and j , then X_k is a member of every node on the path between i and j . This property is called the running intersection property.

If the independence graph G is chordal, an exact factorization of the probability, based on the cliques of the graph, exists. The factorization can be represented using a JT . If G is not chordal, a chordal super-graph of G can be found by adding edges to G in a process called triangulization. The problem is that we can not guarantee that the maximum clique of the super-graph will have a size that would make feasible the calculation of the marginal probabilities. The problem of finding a triangulization with maximum clique of minimum size is NP-complete.

Our goal is to find an approximate factorization that contains as many dependencies as possible, but without adding new edges to the graph. An exact factorization would comprise all the dependencies represented in the independence graph. We will assume that approximate factorizations of the probability are more precise as they include more of the dependencies represented in the independence graph. The approximate factorization will be represented using a labeled JG . The algorithm for learning the probabilistic model has five main steps.

Algorithm 1: Model learning

1. Learn an independence graph G from the data (the selected set of solutions).
 2. If necessary, refine the graph.
 3. Find the set L of all the maximal cliques of G .
 4. Construct a labeled JG from L .
 5. Find the marginal probabilities for the cliques in the JG .
-

3 Learning an Approximate Factorization

In this section we consider in detail the different steps for learning an approximate factorization from data.

3.1 Learning of an Independence Graph

The construction of an independence graph from the data can be accomplished by means of independence tests. To determine if an edge belongs to the graph, it is enough to make an independence test on each pair of variables given the

rest. Nevertheless, from an algorithmic point of view it is important to reduce the order of the independence tests. Thus, we have adopted the methodology followed previously by Spirtes [12]. The idea is to start from a complete undirected graph, and then try to remove edges by testing for conditional independence between the linked nodes, but using conditioning sets as small as possible.

To evaluate the independence tests we use the Chi-square independence test. If two variables X_i and X_j are dependent with a specified level of significance α , they are joined by an edge. α is a parameter of the algorithm. In the general case we can assume that each edge $i \sim j$ in the initial independence graph is weighted with a value $w(i, j)$ stressing the pairwise interaction between the variables. This information might be available from prior information, or from the statistical tests conducted on the data (the value of the chi-square test). When such information is not available we assume that all the values of the dependencies are equal to a parameter w' (i.e. $w(i, j) = w', \forall i \sim j \in E$).

3.2 Refinement of the Graph

When the independence graph is very dense, we can expect that the dimension of the cliques will increase. An alternative to solve this problem is, in a step previous to the calculation of the cliques, to make the graph sparser. One way of doing this is allowing a maximum number of incident edges to each vertex. If the vertex has more than r incident edges, those with the lower weights are removed. In this way the size of the maximum clique will be always smaller or equal than r . Our refinement algorithm avoids introducing a bias in the way the edges are removed. However, it has a main drawback: it could be the case that there exist more than $MaxEdges$ variables depending from a single one, but the maximum clique where this variable is included be smaller than r . In this case, the procedure that eliminates the edges would remove dependencies from the graph without a real need to do so. We have not found a better practical solution to this problem.

3.3 Maximal Cliques of the Graph

To find all the cliques of the graphs the Ken and Kerbash algorithm [1] is used. This algorithm uses a branch and bound technique to cut off branches that can lead to cliques. Once all the cliques have been found they are stored in a list L , and their weights are calculated from the information about dependencies. The weight of any subgraph G' of G is calculated as $W(G') = \sum_{i \sim j \in G'} w(i, j)$. In this way the weights of the maximal cliques $w(C_i)$ are calculated.

3.4 Construction of the Labeled JG

Algorithm 2 receives the list of cliques L with their weight, and outputs a list L' of the cliques in the labeled JG . The first clique in L' is the root, and the labels of cliques in the labeled JG correspond to their position in the list. Each clique in the labeled JG is a subset of a clique in L .

Algorithm 2: Algorithm for learning a JG

1. Order the cliques in L in a decreasing order according to their weights
 2. Add element $L(1)$ to list L'
 3. Remove element $L(1)$ from L
 4. While L is not empty
 5. Find the first element C in L such that $C \cap (L'(1) \cup L'(2) \cdots \cup L'(N_{Cliques})) \neq C$, and the number of variables in $C \cap L'(1) \cap L'(2) \cdots \cap L'(N_{Cliques})$ is maximized
 6. If $C = \emptyset$ Remove all the elements in L
 7. else Insert C in L'
-

We focus now on step 5 of algorithm 2. The condition of maximizing the number of variables in $C \cap (L'(1) \cup L'(2) \cdots \cup L'(N_{Cliques}))$ states that the clique C in L that has the highest number of overlapping variables with all the variables already in L' , will be added to L' . The number of overlapping variables has to be less than the size of the clique, constraint meaning that at least one of the variables in C has not appeared before. If there exist many cliques with maximum number of overlapped variables, the one that appears first in L is added to L' . On the other hand, if the maximum number of overlapped variables is zero, then there exists in the JG more than one connected component. In this case we have a set of junction graphs, however we have preferred to abuse the notation and call it JG , whether it has one or more connected components. Finally, the addition of cliques stops when all the variables are already in the JG .

Marginal probabilities are found by calculating the number of counts associated to each configuration, and normalizing. In the implementation, the learned model's parameters can be changed by adding a perturbation in the form of probabilistic priors.

3.5 Description of an Example of the Learning Algorithm

We introduce an example of the application of algorithm 1. The information about the dependencies among the 12 variables of a given problem is represented by the independence graph shown in figure 1 (left). Let us suppose that the maximum number of incident edges allowed is 6. In the refinement step only edges incident to the vertex x_5 has to be removed. If information were available about the dependencies of each link, the two edges with the weakest dependencies would be removed. In the present example we assume that all the dependencies are equally strong, and two arbitrary edges ($x_1 \sim x_5$ and $x_5 \sim s_{10}$) are removed. The refined graph is shown in figure 1 (right).

In the next step all the maximal cliques of the graph are found. There are 9 maximal cliques, all of order 3. Also in this case the cliques have the same weight, therefore we arbitrarily select the clique (x_1, x_2, x_3) as the root.

Fig. 1. Original and refined independence graphs.

Fig. 2. Ordered junction graph and junction tree.

Construction of the ordered JG: In the first step the clique with maximum overlapping with all the variables already in the ordered JG is (x_2, x_3, x_5) . In the next step either of the cliques (x_2, x_4, x_5) or (x_3, x_5, x_6) can be incorporated. Figure 2 (left) shows the final JG . In the cliques shown in the figure, each new variable incorporated to the graph is represented to the left of the bar. Only eight of the cliques are included in the ordered JG , the clique (x_5, x_7, x_8) is missing. Its absence is explained because the algorithm for finding the ordered JG can not guarantee that all the dependencies will be captured. On the other hand, the factorization represented by the labeled JG is invalid because there exists a cycle comprising different cliques.

Construction of the JT: Figure 2 (left) shows the JT obtained using the algorithm. Notice that the JT can represent less dependencies than the JG . As the JT prohibits the existence of cycles, the clique (x_{10}, x_7, x_8) can not be fully represented.

4 Sampling of the Approximate Factorization

Points are sampled from the labeled JG following the order determined by the labels. The variables corresponding to the first clique in the JG are instantiated sampling from the marginal probabilities. For the rest of cliques, each subset of variables that has not been instantiated is sampled conditionally on the variables already instantiated that belong to the clique. The process is very similar to Probabilistic Logic Sampling (PLS) [3], when it is used in junction trees. There exists however an important difference. The definition of JT discards the existence of cycles. A labeled JG can contain cycles, and this fact allows the representation of more interactions, but it does not essentially change the performance of the sampling algorithm. The reason is that in every step of the JG sampling algorithm, the conditioning and conditioned subsets of variables belong to the clique whose variables are being sampled.

5 MN-FDA

Our algorithm is called Markov Network FDA (MN-FDA), its pseudo-code is presented in algorithm 3. The main difference between it and previous FDAs based on undirected models is that it uses as its probabilistic model a labeled JG while previous FDAs based on undirected graphical models [9,11] represent the factorizations using a JT .

Algorithm 3: MN-FDA

-
1. Set $t \leftarrow 0$. Generate $N \gg 0$ points randomly.
 2. **do** {
 3. Select a set S of $k \leq N$ points according to a selection method.
 4. Learn a labeled JG from the data.
 5. Calculate the marginal probabilities for all the cliques in the JG .
 6. Generate a the new population sampling from the JG .
 7. $t \leftarrow t + 1$
 8. } **until** Termination criteria are met
-

In all the experiments presented in this paper the algorithm used to learn the independence graph only considered independence tests up to third order. The level of significance α was set to 0.75. This choice was motivated by the need of capturing as many dependencies as possible. Even if some of the found dependencies might be false, this is preferable that missing some of the real dependencies. The number of allowed neighbors for the refinement algorithms was set to 8.

5.1 Computational Cost of the MN-FDA

The number of operations needed to make the independence tests is upper bounded by $O(Nn^3)$. The worst complexity of the refinement algorithm is bounded by $O(n^2 \log(n))$. It has been calculated considering the case when after the independence tests, the graph remains complete. The time complexity of the Bron and Kerbosch algorithm is not calculated in their original work [1]. However from comparisons with other algorithms for which bounds have been calculated the worst complexity of the algorithm can be estimated as $O(\mu^2)$, where μ is the number of maximal cliques. When there are at most k edges for each variable and $k \ll n$, a bound for the number of cliques can be given by kn , and the complexity of the Bron and Kerbosch algorithm roughly estimated as $O(\mu^2) \approx O(n^2)$. The complexity of learning the parameters depends on the size of the population N , the number of cliques μ , and their size. The order of this step is $O(N\mu) \approx O(Nn^2)$. The total complexity of the MN-FDA is $O(Nn^3)$.

6 Experiments

In our experiments we compare the behavior of the MN-FDA with the following FDAs: The FDA^* [9], it uses a fixed model of interactions, only the parameters of the cliques are learned in each generation. The Univariate Marginal Distribution Algorithm (UMDA) [6], which uses a model that assumes all the variables are independent. In every step, the algorithm makes a parametric learning of the univariate probabilities. The Tree-FDA [11] uses a probability model where each variable is conditioned on at most one parent. The Learning FDA (LFDA) [8] is a FDA that uses a Bayesian probabilistic model.

First, a number of functions commonly used to evaluate evolutionary algorithms are presented. A practical problem used in the experiments is also described. All the problems used in the experiments are defined on binary variables. The numerical results and the analysis of the experiments are presented afterward.

6.1 Functions Used in the Experiments

Deceptive functions were introduced by Goldberg to show the deceptive nature of the GAs behavior, and to address the problems given by the convergence to local optima of the function. The following 4 elementary deceptive functions of k variables are used to define some of the additive functions used in our experiments. They are defined in terms of the unitation value $u(x) = \sum_{i=1}^n x_i$. Each entry of the table shows the evaluation of the corresponding deceptive function when the unitation value of the subset of variables it is evaluated on is u .

u	0	1	2	3	4	5
f_{dec}^3	0.9	0.8	0	1		
f_{dec}^4	3	2	1	0	4	
$IsoT_1$	m	0	0	0	0	0
$IsoT_2$	m	0	0	0	0	$m-1$

$$Onemax(x) = \sum_{i=1}^n x_i \quad (1)$$

$$f_{3deceptive}(x) = \sum_{i=1}^{i=\frac{n}{3}} f_{dec}^3(x_{3i-2}, x_{3i-1}, x_{3i}) \quad (2)$$

$$Deceptive_4(x) = \sum_{i=1}^{i=\frac{n}{4}} f_{dec}^4(x_{4i-3}, x_{4i-2}, x_{4i-1}, x_{4i}) \quad (3)$$

$$F_{IsoP}(n, m, k, x) = \left(\sum_{i=1}^n x_i \right) + k \cdot (m+1)((1-x_1) \cdots (1-x_m)x_{m+1} \cdots x_n) \quad (4)$$

When analyzing interactions between variables it is important to consider interactions that do not depend on the linear codification of solutions. To this end we considered function $F_{IsoTorus}$ (5) where x_{up} , x_{left} , etc., are defined as the appropriate neighbors, wrapping around.

$$F_{IsoTorus}(x) = IsoT_1(x_{1-m+n}, x_{1-m+n}, x_1, x_2, x_{1+m}) + \sum_{i=2}^n IsoT_2(x_{up}, x_{left}, x_i, x_{right}, x_{down}) \quad (5)$$

Function $BigJump$ (6) was introduced in [7]. A valley has to be crossed in order to reach the global optimum of this function. The bigger the parameter m is for this function, the wider the valley. k can be increased to give bigger weight to the maximum.

$$BigJump(u, n, m, k) = \begin{cases} u & \text{for } 0 \leq u \leq n-m \\ 0 & \text{for } n-m \leq u \leq m \\ k \cdot n & \text{for } u = m \end{cases} \quad (6)$$

The generalized Ising model (7) is described by the energy functional (Hamiltonian) where L is the set of sites called a lattice. Each spin variable σ_i at site $i \in L$ either takes the value 1 or value -1 . A specific choice of values for the spin variables is called a configuration. The constants J_{ij} are the interaction coefficients. In our experiments we take $h_i = 0$, $\forall i \in L$. The ground state is the configuration with minimum energy.

$$H = - \sum_{i < j \in L} J_{ij} \sigma_i \sigma_j - \sum_{i \in L} h_i \sigma_i \quad (7)$$

6.2 Numerical Results

In all the experiments we use the truncation selection. In the following tables n is the number of variables, N is the population size, *succ* is the number of times the optimum was reached in 100 experiments, *gen* the average number of generations to convergence, $\hat{f}$ the average fitness of the best found solutions, and *eval* is the average number of evaluations needed to find the optimum.

Table 1. Comparison between the MN-FDA with other FDAs for unitation functions.

<i>OneMax</i>				<i>BigJump</i> (30, 3, 1)				<i>Deceptive4</i>			
n	<i>Alg.</i>	N	<i>succ.</i>	n	<i>Alg.</i>	N	<i>succ.</i>	n	<i>Alg.</i>	N	<i>succ.</i>
30	UMDA	30	75	30	UMDA	200	100	32	UMDA	800	0
30	LFDA _{0.25}	100	2	30	LFDA _{0.25}	200	58	32	FDA*	100	81
30	LFDA _{0.5}	100	38	30	LFDA _{0.5}	200	96	32	LFDA _{0.25}	800	92
30	LFDA _{0.75}	100	80	30	LFDA _{0.75}	200	100	32	LFDA _{0.5}	800	72
30	LFDA _{0.25}	200	71	30	LFDA _{0.25}	400	100	32	LFDA _{0.75}	800	12
30	MN-FDA	30	72	30	MN-FDA	100	92	32	MN-FDA	600	90
30	MN-FDA	100	98	30	MN-FDA	200	100	32	MN-FDA	800	100

In table 1 results of the MN-FDA for different functions are compared with results published in [7] for the UMDA and the LFDA. For the functions considered in our experiments, results of the LFDA were available in [7] only for the values of the LFDA parameter α presented in the table². In these functions the MN-FDA achieved equal or better results than the LFDA. We have observed that the learning algorithm used by the MN-FDA easily detects variables that are independent. The BN learning algorithms used by Bayesian FDAs may have problems recognizing independence, particularly if α is small.

In table 2 we have included the results for the UMDA, the Tree-FDA, and the LFDA for other functions. In all the cases $N = 1000$, the truncation parameter is 0.15 and the maximum number of generations is 25. For the LFDA, $\alpha = 0.5$.

For function $f_{3deceptive}$ the results of the MN-FDA are the best. For function $F_{IsoTorus}$ LFDA finds the optimum more times than the MN-FDA, however its average fitness is lower. For function F_{IsoP} the LFDA achieved the best results, although the difference is not as significant as in the case of the $f_{3deceptive}$. The UMDA was not able to solve the problems with interactions.

We have generated 4 random instances of the Ising model for different number of variables ($n \in \{25, 36, 49, 64\}$). For each of the instances we investigate two different issues. First, the influence of using the prior information about the interactions of the variables. MN-FDA^s is a Markov Network FDA that does not learn the independence graph from the data. In this case, the lattice where the Ising model is defined serves as the independence graph. The maximum size of the

² Parameter α has different meanings in the LFDA and the MN-FDA, although in the LFDA it also serves to specify the density of the Bayesian network.

Table 2. Comparison between the MN-FDA with other FDAs for different functions.

	$f_{3deceptive}$			$F_{IsoTorus}$			F_{IsoP}		
	<i>succ.</i>	$\hat{f}$	<i>gen</i>	<i>succ.</i>	$\hat{f}$	<i>gen</i>	<i>succ.</i>	$\hat{f}$	<i>gen</i>
MN-FDA	77	11.97	8.0	78	210.78	5.8	69	1190.69	5.0
UMDA	0	0	0	17	175.01	8.7	0	0	0
Tree-FDA	36	11.92	10.47	75	209.30	6.2	65	1190.65	4.9
LFDA	45	11.91	6.4	85	210.55	6.0	76	1190.75	4.6

Table 3. Results of the MN-FDA for different Ising instances.

<i>Inst.</i>	<i>N</i>	MN-FDA ^s		MN-FDA	
		<i>succ.</i>	<i>eval.</i>	<i>succ.</i>	<i>eval.</i>
I^{25}	200	100	849	43	1163
I^{36}	400	86	2316	41	2453
I^{49}	700	82	3841	36	4201
I^{64}	700	67	6031	28	6641

cliques is equal 2. The second issue we study is the scaling of the algorithm when the population size is fixed to 1000, and the coefficient of truncation selection is $T = 0.15$.

The results of these experiments are shown in table 3. An analysis of the results reveals the convenience of using prior information about the optimization problem for increasing the efficiency of the MN-FDA. The small population size that is enough for the convergence of the MN-FDA^s is not sufficient for the MN-FDA. As expected, when the number of variables increases, a higher population size is needed to solve the problem.

7 Conclusions

In this paper we have presented a FDA that approximates the probability distribution determined by selection using a labeled JG . The JG is found by calculating the maximal cliques of a Markov Network that can be given as an input or learned from the data. Our work is related with previous work by Muehlenbein et al. [9], where approximate factorizations were recognized as an alternative for modeling probabilistic distributions. Our research, that has led to a different way of finding these approximations, is also related with the work presented by Brown et al. [2] in the application of MRFs to GAs. They have used probabilistic models of GA fitness functions to generate new solutions. Our work shows a number of relevant differences with this approach: The use of statistical tests to learn the structure of interactions. In [2] the structure of the interactions is known a priori. The construction of the JG from the MN, and the use of PLS on the JG . In [2] the Metropolis algorithm is used to generate new solutions.

The results of our experiments show that the MN-FDA is able to optimize theoretical functions as well as functions derived from practical problems, out-

performing other evolutionary algorithms. The MN-FDA generalizes other FDAs by learning factorizations that have not to be valid. More theoretical investigation is needed to determine bounds for the convergence of the MN-FDA. Other practical optimization problems must be tried to assess the performance of the algorithm.

References

1. C. Bron and J. Kerbosch. Algorithm 457—finding all cliques of an undirected graph. *Communications of the ACM*, 16(6):575–577, 1973.
2. D. F. Brown, A. Garmendia-Doal, and J. A. W. McCall. Markov random field modelling of royal road genetic algorithms. In P. Collet, editor, *Proceedings of EA 2001*, volume 2310 of *Lecture Notes in Computer Science*, pages 65–76. Springer Verlag, 2002.
3. M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Uncertainty in Artificial Intelligence*, 2:317–324, 1988.
4. J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.
5. P. Larrañaga and J. A. Lozano. *Estimation Distribution Algorithms. A new tool for Evolutionary Optimization*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2001.
6. H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.
7. 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.
8. H. Mühlenbein and T. Mahnig. Evolutionary synthesis of Bayesian networks for optimization. *Advances in Evolutionary Synthesis of Neural Systems*, MIT Press, pages 429–455, 2001.
9. H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.
10. H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In A. Eiben, T. Bäck, M. Shoenauer, and H. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, Berlin, 1996. Springer Verlag.
11. R. Santana, A. Ochoa, and M. R. Soto. The Mixture of Trees Factorized Distribution Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2001*, pages 543–550, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
12. P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and search*. Lecture Notes in Statistics. Springer-Verlag, New York, 1993.