Solving problems with integer representation using a tree based Factorized Distribution Algorithm

Roberto Santana, Alberto Ochoa, Marta R. Soto
Center of Mathematics and Theoretical Physics, Havana, Cuba
ICIMAF, Calle 15, e/ C y D, Vedado,
C-Habana, Cuba, CP-10400
{rsantana,ochoa,mrosa}@cidet.icmf.inf.cu

Abstract

In this paper a tree based Factorized Distribution Algorithm for the solution of integer problems is introduced. Our proposal combines classical methods for structural learning of dependencies with a procedure that approximates the bivariate marginals by sampling the data using auxiliary tables. Experiments done for a number of problems with an integer representation show evidence of the superiority of the algorithm with respect to the Univariate Marginal Distributed Algorithm.

1 Introduction

Population Based Search Methods that use Selection (PBSMS) differ from other classical search methods in their use of a population of points to accomplish the search. The initial population of points is evaluated using an objective function, and a subset of the points is selected based on their values. The next population is generated using information extracted from this subset of points. Classical PBSMS are Genetic Algorithms (GAs) [3], where the information contained in the selected set of solutions is manipulated to generate the next population by means of the so called crossover and mutation operators.

Another class of PBSMS that do not use genetic operators to conduct the search are the Factorized Distribution Algorithms (FDAs) [7]. FDAs combine results from Graphical Models and Evolutionary Computation, and are considered as a tractable subclass of Estimation Distribution Algorithms [8]. They apply the selection operator in each generation but do not use the crossover and mutation operators, a factorized probabilistic model of the selected points is constructed instead. Probabilistic models allow to explicitly represent the statistical information that is contained in the selected points, and use it to improve the search by sampling points in the promising regions of the space of solutions.

Previous applications of FDAs to integers problems have mainly relied upon binary codifications of integers [11], or the use of simple probabilistic models. Rosete [10] uses an Univariate Marginal Distribution Algorithm (UMDA) [6] that works straight on integer representation. The algorithm was ranked in the worst class of techniques used to solve the automatic graph drawing problem he considered. When more complex models have been used, the cardinality of the variables has been low [13].

There exists a particular subclass of FDAs that considers only up to pairwise dependencies among variables. These algorithms have been successfully applied to a number of problems [1, 9], and in principle they could deal with integer problems in the same way they do with binary ones. However, FDAs that use pairwise or higher dependencies have not been used to solve problems defined on integers, and the reason is the very high memory requirements needed. FDAs that use up to bivariate pairwise dependencies require, for a problem of n variables, $\frac{n \cdot (n-1)}{2}$ bivariate marginals to be stored. Let c be the cardinality of variables, the total numbers of entries needed to store all the bivariate marginals is $\frac{c^2 \cdot n \cdot (n-1)}{2}$.

In this paper we present a FDA that considers up to second order statistics, and permits to carry out the optimization of integer problems with a high cardinality of the variables.

2 Gene Pool Recombination and the Univariate Marginal Distribution Algorithm

Gene Pool Recombination (GPR) was introduced in [5] as an alternative to the Two Parent Recombination (TPR) methods commonly used in GAs. In GPR the two "parent" alleles of an offspring are randomly chosen for each locus with replacement from the gene pool given by the parent population selected before. Then the offspring's allele is computed using any of the standard recombination schemes for TPR [5]. On the other hand in the Univariate Marginal Distribution Algorithm (UMDA) the estimation of the distribution of the selected set is done with a very simple linear model, the so-called univariate marginal distribution. Before to present the UMDA model we formally introduce the notation that will be used throughout this paper.

Let $X = (X_1, ..., X_n)$ where X_i is a discrete variable with r_i (not necessarily consecutive) possible assignments : $(v_{i,1}; ...; v_{i,r_i})$ and we will denote one of these possible instantiations as x_i, i.e. $x_i = v_{i,j}$ with $j \in \{1, 2, ..., r_i\}$. Let $P = \{x^1, ..., x^N\}$ be a set of N instantiations of X that following the traditional GA's notation we will call population. Each x^j will be called a vector, point or individual of P, and we will refer to its ith component as x_i^j. Given a population of vectors P we define the univariate marginal probability $p(X_i = x_i)$ as the probability of vectors in P that satisfy $x_i^j = x_i$.

In FDAs the processes of selection, estimation and generation are applied in every generation, thus variable t is incorporated to the previous notation to represent the generation t. We will define $S(t)$ as the selected population at generation t, and $N_s(t)$ as its size. In the UMDA the univariate marginal frequencies for the selected set $p_i(x_i)$ are calculated first. The distribution of points in the selected set is estimated as

$$p^s(x_1, ..., x_n) = \prod_{i=1}^{n} p_i^s(x_i) \qquad (1)$$

Vectors of the new population are generated according to this distribution. When the problem under consideration is based on an integer representation, and the recombination scheme used by GPR is the uniform crossover, GPR and UMDA have similar behavior. Nevertheless, while in the UMDA a probabilistic model is explicitly stored in the univariate marginals of variables, in the GPR the existence of an univariate model is implicitly assumed during the generation step.

FDAs that considers up to pairwise dependencies

- **STEP 0:** Set $t \Leftarrow 0$. Generate $N \gg 0$ points randomly.
- **STEP 1:** Select $k \leq N$ points according to a selection method. Compute the univariate marginal distributions $p_i^s(x_i, t)$ and the bivariate marginal distributions $p_{i,j}^s(x_i, x_j, t)$ of the selected set $S(t)$.
- **STEP 2:** Create the dependency graph $G = (V, E)$ using the distributions p_i and $p_{i,j}$.
- **STEP 3:** Generate N new points using the dependency graph G and the distributions p_i and $p_{i,j}$. Set $t \Leftarrow t + 1$
- **STEP 4:** If the termination criteria are not met, go to STEP 1

Figure 1: General scheme of the FDAs that consider up to pairwise dependencies

3 FDAs that considers up to pairwise dependencies

According to the graphical model paradigm, each variable is seen as a vertex of an (undirected) graph $G = (V, E)$, there is an edge between two vertices if the corresponding variables are not independent in a data set. A pseudo-code explaining the general scheme of algorithms that consider up to pairwise dependencies is presented in figure 1. The main differences among these algorithms is the type of dependency graphs they employ and the way they are constructed. FDAs that use forest models and mixture of trees, for example, are able to capture a more general class of independency relationships than tree based FDAs.

A connected graph that has no cycles is called a tree. Given that one vertex of the tree has been set as the root, all the vertices connected to it are called descendants of the root. This process is repeated recursively until every vertex has been incorporated as a descendant of its parent. The parent of the vertex corresponding to variable X_i is represented as $pa(X_i)$, the root vertex has no parent. Now we define a probability distribution T that is conformal with a tree.

$$T(X_1, ..., X_n) = \prod_{i=1}^{n} p(_{X_i|pa(X_i)}) \qquad (2)$$

The distribution T itself will be called a tree when no confusion is possible.

Trees are the class of dependency graphs used by the FDA introduced in this paper. Trees have been used before in the context of FDAs, but mostly for problems with binary representation. Baluja and Davies [1] present an optimization algorithm that uses a probabilistic model defined on trees. The algorithm has shown to be superior to the simple GA and to FDAs that employ chain shaped dependency graphs [1]. The FDA presented in this paper also uses the Chow and Liu's algorithm [2] for searching the best probability distribution T that is conformal with p. This algorithm finds the tree that minimizes the mutual information between p and T.

It is important to note that the bivariate probabilities are only needed for calculating the mutual information. There is not need to save them for later use (at least during this model learning step). Hence, the algorithm that we present later uses the procedure explained in figure 2 to calculate the mutual information matrix I.

It can be seen that the time complexity this algorithm exhibits is equal to $\frac{n\cdot(n-1)N\cdot(N-1)}{4}$, that is $\theta(n^2N^2)$. The time complexity of the traditional algorithm for finding the matrices of bivariate probabilities and matrix I is $\theta(n^2N)$. The difference in the magnitudes can be explained because when bivariate probabilities are stored, only one pass to the population (columns i and j) is needed to calculate $I_{i,j}(X_i, X_j)$. Nevertheless, through generations, the diversity in the populations diminishes and less time is consumed by our procedure. It can be noticed that when there is only one pair of values in variables X_i and X_j, the time complexity of our algorithm is also $\theta(n^2N)$.

Once the matrix I has been computed the next step is to determine the tree structure. This is usually done by applying the Maximum Weight Spanning Tree (MWST) method using as the tree's weights the values of I. The edges in the maximum spanning tree of G determine an optimal set of $(n-1)$ first–order conditional probabilities with which to model the original probability distribution. Several variants of this algorithm exist [4], the simplest one is called Kruskal algorithm. It runs in $\theta(n^2\log(n))$ time. Other variants can reduce this time complexity to $\theta(n^2)$.

When the structure of the tree has been found, the following step is to generate new vectors. But to do so FDAs that use up to pairwise dependencies need the bivariate marginals. In the next section we show how can vectors be generated, keeping the pairwise dependencies and avoiding the use of the bivariate probabilities.

Algorithm for finding the mutual information

```
INPUT : Population $P$ of size $N$
OUTPUT: Matrix $I$ of mutual information
BEGIN Mutual Information
FOR $i := 1$ To $n-1$ DO
FOR $j := i+1$ To $n$ DO
$k := 0$;
$I(X_i, X_j) := 0$;
$Ind := [1...N]$;
WHILE ( $k < N$) DO
$biv\_freq := 1$;
$univ\_freq\_x_i := 1$;
$univ\_freq\_x_j := 1$;
    FOR $l := k+1$ TO $N$
    IF $(x_i^{Ind(l)} = x_i^{Ind(k)})$
    $univ\_freq\_x_i := univ\_freq\_x_i + 1$;
    END
    IF $(x_j^{Ind(l)} = x_j^{Ind(k)})$
    $univ\_freq\_x_j := univ\_freq\_x_j + 1$;
    END
    IF $(x_i^{Ind(l)} = x_i^{Ind(k)}$ and $x_j^{Ind(l)} = x_j^{Ind(k)})$
    $biv\_freq := biv\_freq + 1$;
    $swap\ Ind(k + current\_biv\_freq), Ind(l)$;
    END
    END
$k := k + current\_biv\_freq$;
END
$tmp = \frac{biv\_freq}{N} \cdot \log \frac{N \cdot biv\_freq}{univ\_freq\_x_i \cdot univ\_freq\_x_j}$;
$I(X_i, X_j) := I(X_i, X_j) + tmp$;
END
END
END Mutual Information
```

Figure 2: Algorithm for calculating the mutual information

4 Vector generation

Now we outline the idea of our algorithm. We will create two tables that allow to identify which are the feasible values $v_j \subset (v_{j,1}; ...; vj_{j,r_j})$ a variable X_j can take given that its parent X_i has been assigned a value $v_i \subset (v_{i,1}; ...; v_{i,r_i})$. Tables permit also to generate X_j with the same probability $p(X_j|X_i)$ that exists in the selected set. During the generation step the instantiation of the root variable will be done by randomly choosing a vector from the selected population picking the value of the root variable from this vector as it is usually done for all

$$p(X) = p(x1)p(x_2|x_1)p(x_5|x_1)p(x_3|x_2)...$$
$$p(x_4|x_2)p(x_6|x_5)p(x_7|x_6)p(x_8|x_6)$$
$a)$

$$P = \left[\begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ 4 & 2 & 4 & 1 & 2 & 0 & 1 & 1 \\ 2 & 2 & 1 & 2 & 2 & 4 & 0 & 3 \\ 2 & 2 & 3 & 1 & 2 & 3 & 0 & 3 \\ 1 & 0 & 0 & 2 & 2 & 2 & 3 & 3 \\ 1 & 1 & 0 & 0 & 2 & 1 & 0 & 0 \end{array}\right]$$
$b)$

$$PV = \left[\begin{array}{ccc} x_1 & x_5 & x_6 \\ 4 & 1 & 1 \\ 5 & 2 & 5 \\ 2 & 3 & 4 \\ 3 & 4 & 3 \\ 1 & 5 & 2 \end{array}\right] \quad PI = \left[\begin{array}{ccc} x_1 & x_5 & x_6 \\ -1 & -1 & 1 \\ 2 & -1 & 2 \\ 4 & 5 & 3 \\ 4 & 5 & 4 \\ 5 & 5 & 5 \end{array}\right]$$
$c)$

Figure 3: Example of the use of the auxiliary tables for the sampling

variables in the GPR.

In our case the problem to be solved is how to instantiate variables that descend from variables already instantiated in the tree. The structure of the dependency tree is used during this generation step to track the parental relations among the variables. Given that the parent X_i has been assigned a value x_i, possible values for the descendant X_j are those in the position j of all the vectors in the original selected population that satisfy $X_i = x_i$. Hence, it is enough to know the indices, in the selected population, of all the vectors corresponding to each value of each parent.

In tables *PopulValues* and *ParentIndices* this information will be stored. We will introduce and illustrate the meaning of these tables with an example. Figure 3a) shows a tree factorization corresponding to a problem of 8 variables, each variable can take values in $[0, ..., 4]$. In figure 3b), P represents a population of 5 vectors (the distribution of points in P does not correspond to the factorization).

Column i in the *PopulValues* table contains the vector's indices in the population ordered according to the values of variable i. In figure 3c) PV represents columns corresponding to variables x_1, x_5 and x_6 of the *PopulValues* table. Column 1 in PV represents the indices of vectors in P ordered

Algorithm for the generation of vectors

```
INPUT: $P$, $PopulValues$, $ParentIndices$, $Tree$
BEGIN Vector Generation
Instantiate the root variable.
FOR each variable $X_j$ whose parent $X_i$ is already
instantiated to value $v_{i_k}$ in the vector
Select a random value $r$ between
$ParentIndices(k-1,i)+1$ and $ParentIndices(k,i)$;
$X_j = PopulValues(ParentIndices(k-1,i)+r\ ,j)$;
END
  IF there exists any variable that has not
  been already instantiated.
   Go to step 2;
  END
END Vector Generation
```

Figure 4: Algorithm for the generation of vectors

according to values of variable x_1, the first couple of values $(4, 5)$ correspond to the indices where x_1 is equal 1 in P (note that x_1 is never equal zero in P), then values $(2, 3)$ correspond to indices where $x_1 = 2$, and so on.

Entry (k, i) in *ParentIndices* keeps the last index of values v_{i_k} for variable i in *PopulValues*. If $ParentIndices(k, i) = ParentIndices(k-1, i)$ or $ParentIndices(1, i) = -1$ then there are not values of type v_{i_k} in the component i of all the vectors in the population. In figure 3d) PI represents the columns of *ParentIndices* corresponding to variables x_1, x_5 and x_6. First entry in column 1 of PI is -1 because x_1 is never zero in P. Entry 2 of the same column is 2, which is the last row in *PopulValues* corresponding to indices where $x_1 = 1$.

Finally we point out that in the worst case (when there are $n-1$ parents) the total memory required to store tables *PopulValues* and *ParentIndices* is $N(S) \cdot (n-1) + c \cdot (n-1)$. Considering that truncation selection is used, and thus $N(S)$ is a small fraction of the original population size we confirm that $(n-1) \cdot (N(S) + c) \ll \frac{c^2 \cdot n \cdot (n-1)}{2}$.

In figure 5 we present the final algorithm that uses the procedures introduced in previous sections.

5 Experiments

We conducted several experiments to study the behavior of our algorithm. We will first present re-

Tree based FDA for Integers (Int-Tree)

- **STEP 0:** Set $t \Leftarrow 0$. Generate $N \gg 0$ points randomly.
- **STEP 1:** Select $k \leq N$ points according to a selection method.
- **STEP 2:** Create the tree T using the algorithm for finding the mutual information and the MWST method.
- **STEP 1:** Create, using T, the auxiliary tables that represent the information stored in the selected set.
- **STEP 3:** Generate N new points using T and the auxiliary tables. Set $t \Leftarrow t+1$
- **STEP 4:** If the termination criteria are not met, go to STEP 1

Figure 5: General scheme of the FDAs that consider up to pairwise dependencies

sults of the comparison between the introduced algorithm with a tree based FDA that stores the bivariate probabilities of a binary problem.

Table 1 presents the results achieved for the checkerboard problem [1]. The goal of the problem is to create a checkerboard pattern of 0's and 1's in an NxN grid. In our experiments N was set to 12. We used truncation selection, and the elitism strategy of best selection that adds to the next population all the points selected in the current generation.

In Table 1 are shown the times the optimum was found for different population sizes. The discrepancy between the number of times that both algorithms found the optimum was very tiny. In experiments with other functions we validated that for binary problems, the algorithm has similar efficiency that the tree based FDA that stores the bivariate probabilities.

Alg/Psize$\cdot 10^3$	1	1.5	2	2.5	3	3.5
Tree	0	21	55	81	93	96
Int-Tree	0	18	51	83	93	97

Table 1: Times the optimum of the checkerboard function was reached for Int-Tree and a tree based FDA.

The second group of experiments was designed to test the algorithm on integer functions. In the experiments we compare the Int-Tree algorithm with the UMDA. It has been shown that the UMDA behaves as well or better than the simple GA [8]. We have shown that the UMDA is also very similar to a GA with multi-parent recombination. We will present results for the general deceptive function $intdecp_k^n$ of order 2 ($k = 2$). This function is formed as an additive function composed by the function $F_{dec(X,k,c)}$ evaluated in substrings of size k, where c is the cardinality of the variables, $Sum(x)$ is the sum of the values of the k variables evaluated by the sub-function, and

$$F_{dec(X,k,p)} = \begin{cases} k \cdot c, \ for\, Sum(X) = k \cdot c \\ k \cdot c - Sum(X) - 1, \ otherwise \end{cases} \quad (3)$$

Table 2 (Left) shows a comparison of the proposed algorithm with the UMDA for the $intdecp_2^n$ function $n = (10, ..., 90)$, $c = 10$. Table 2 (Right) shows the results obtained for the $intdecp_2^{50}$ function when the cardinality of variables is increased from 2 to 10. In both experiments the parameters of the algorithms were $N = 25000$, truncation selection with $T = 0.02$, and best selection. In the tables, entries represent the mean fitness of 50 runs for the UMDA and the proposed algorithm.

The mean of the sub-optima found by the Tree-Int algorithm is higher than when the UMDA is used. Thus, even when the population size is too small, it is expected that an algorithm based on pairwise dependencies, dealing with a problem defined on integers, could achieve partial solutions better than those obtained by the UMDA.

n	UMDA	IntTree
10	86.74	89.84
20	170	177.9
30	255	267.18
40	340	352.46
50	425	436.5
60	510	519.68
70	595	603.6
80	680	687.1
90	765	770.44

c	UMDA	IntTree
2	50	50
3	100	100
4	130.68	150
5	175.46	194.76
6	225.02	244.96
7	275	292.7
8	325	334.08
9	375	386.82
10	425	436.5

Table 2: Comparison between the Tree-Int algorithm and the UMDA for $intdecp_2^n$. (Left): when the number of variables is increased. (Right): when the cardinality of variables is increased

5.1 The Multiple Sequence Alignment problem

After testing the algorithm in a number of theoretical functions we evaluate its behavior for a more practical problem. The main problem of Multiple Sequence Alignment (MSA) is to find an optimal superposition between n strings defined on an alphabet B. The most simple objective is to optimize the percentage of letters that are identical in each position between the n sequences. Multiple alignments are very used in Molecular Biology, for instance to find diagnostic patterns, to characterize protein families, and to detect or demonstrate homology between new sequences and existing families of proteins.

When the number of sequences is two the problem can be efficiently solved with a simple dynamic program. This program can incorporate insertions or deletions (gaps) in the sequences in order to obtain a better alignment. Our approach to the MSA problem splits its solution in two parts. First we look for the substrings that are common in the sequences. Then, in a second phase, that we do not analyze in this paper, these common substrings are combined to create the final alignment. To find the common substrings no gaps are required.

The algorithm Int-Tree will be used to find an alignment between the n sequences. Only some substrings of the sequences are considered for the alignment. Thus, a window of observation W is defined. The width of the window $|W|$ represents how long are the substrings of the sequences we try to match. The window can be set at position S_{i_j} of sequence S_i guaranteeing that $S_{i_j} + |W| < |S_i|$.

Representation: Each possible solution is represented using a vector with length equal to the number of sequences. Each variable x_i in the vector takes values in $[1, L_i - |W|]$ where L_i is the length of sequence i. In our experiments all the sequences have the same length L. A vector encodes the positions of the window for every sequence. The optimal solution is when the windows have been located in such a way that their content is the same for all the sequences

Fitness function: We use a measure of similarity between all substrings enclosed by the windows. This measure has been previously used in [12] where a hill climbing algorithm is employed to find a multiple alignment of ungapped DNA sequences.

Let Ω denote the alphabet where sequences are defined. Each member of the alphabet will be called a letter or a base. $W(i,j)$ denotes the base at position j of the window located on sequence S_i. Let $n(b,j)$ to denote the number times letter b is found at position j of the windows $(W(i,j) = b)$. Then, the uncertainty $f(b,j)$ of each possible base in position j of the window is calculated as:

$$f(b,j) = \frac{n(b,j)}{n} \tag{4}$$

The function information from the entire window is finally considered in the following summation.

$$F(S_i) = |W| \cdot \log_2 |\Omega| + \sum_{j=1}^{|W|} \sum_{b \in \Omega} f(b,j) \cdot \log_2 f(b,j) \tag{5}$$

This function is maximized when the second term of the expression is zero, i.e. all the substrings are identical.

Convergence				
	$UMDA$		$Int - Tree$	
	5	10	5	10
$\|W\| = 3$	30	30	30	30
$\|W\| = 4$	15	0	30	0
$\|W\| = 5$	0	0	23	15
$\|W\| = 6$	0	0	0	0
$\|W\| = 9$	0	0	0	0
$\|W\| = 12$	5	0	29	7

Table 3: Comparison between the Int-Tree algorithm and the UMDA for the MSA problem (Convergence).

Average Fitness				
	$UMDA$		$Int - Tree$	
	5	10	5	10
$\|W\| = 3$	9	9	9	9
$\|W\| = 4$	11.33	11.67	12	11.85
$\|W\| = 5$	14.37	14.37	14.83	14.42
$\|W\| = 6$	17.1	16.43	17.61	16.9
$\|W\| = 9$	24.92	23.43	24.94	23.91
$\|W\| = 12$	34.73	35.61	35.94	34.19

Table 4: Comparison between the Int-Tree algorithm and the UMDA for the MSA problem (Average Fitness).

In Tables 3 and 4 we present results achieved for an artificial generated set of 10 sequences of length 30, defined on an alphabet of cardinality 4. Sequences were generated to contain many partial alignments of short length but only one of size 12. In table 3 there are shown results achieved by the UMDA and Tree-Int algorithm in the alignment of 5 and 10 sequences, using windows with different width. 30 experiments where conducted for each set of parameters. In all the experiments the parameters of the algorithms were $N = 15000$, truncation selection with $T = 0.05$, and best elitism.

The alignment was achieved only when the window's width was 12. This could be explained due to the existence, when the windows are small, of many partial solutions. In this case there are many sub-optima that lead the search to different, some time opposite, directions. When the window's width is 12, more information is considered for the search. Nevertheless, it can be seen in the table that even for this case results of the UMDA are very poor. The Int-Tree algorithm is superior, but it also fails to find the optimum when the windows' width is shortened.

The MSA problem deserves more study. A rigorous comparison of the Int-Tree algorithm with other search strategies previously used for the MSA problem solution, and that are not based on populations is advisable. This comparison could throw light to the convenience of applying FDAs to complex real problems of this kind.

6 Conclusions and further work

In this paper we have tried to partially remedy an unsatisfactory state of affairs in the use of FDAs for the optimization of integer problems. The algorithm we have introduced allows the use of pairwise dependencies in the optimization of functions defined on integer variables. Although previous FDAs that use up to pairwise dependencies can theoretically deal with integer variables, in practice memory and time requirements make them useless. The algorithm shows better results than the UMDA for the optimization of the functions considered.

References

[1] Shumeet Baluja and Scott Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proceedings of the 14th International Conference on Machine Learning*, pages 30–38. Morgan Kaufmann, 1997.

[2] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT14(3):462–467, 1968.

[3] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA, 1989.

[4] Marina Meila. *Learning Mixtures of Trees.* PhD thesis, Massachusetts Institute of Technology, 1999.

[5] H. Mühlenbein and H. M. Voigt. Gene pool recombination in genetic algorithms. In I. H. Osman and J. P. Kelly, editors, *Proceedings of Metaheuristics International Conference*, Norwell, 1995. Kluwer Academic Publishers.

[6] Heinz Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

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

[8] Heinz Mühlenbein and Gerhard Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In A.E. Eiben, T. Bäck, M. Shoenauer, and H.P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, Berlin, 1996. Springer Verlag.

[9] Martin Pelikan and Heinz Mühlenbein. Marginal distributions in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel '98*, pages 90–95, Brno, Czech Republic, 1998.

[10] A. Rosete. *Automatic Graph Drawing and Stochastic Hill Climbing.* PhD thesis, CEIS, ISPJAE, Cuba, 2000. In Spanish.

[11] Franz Rothlauf, David E. Goldberg, and Armin Heinzl. Bad codings and the utility of well-designed genetic algorithms. IlliGAL Report No. 200007, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2000.

[12] Thomas. D. Schneider and David N. Mastronarde. Fast multiple alignment of ungapped dna sequences using information theory and a relaxaton method. *Discrete Applied Mathematics*, (71):259–268, 1996.

[13] Josef Schwarz and Jiri Ocenasek. Experimental study: Hypergraph partitioning based on the simple and advanced algorithms BMDA and BOA. In *Proceedings of the Fifth International Conference on Soft Computing*, pages 124–130, Brno, Czech Republic, 1999. PC-DIR.