A Factorized Distribution Algorithm of bounded complexity for integer problems

Roberto Santana, Alberto Ochoa, Marta R. Soto
Institute of Cybernetics, Mathematics and Physics.
Calle 15, e/ C y D, Vedado, C-Habana, Cuba
{rsantana,msoto,ochoa}@cidet.icmf.inf.cu

Abstract

A Factorized Distribution algorithm that use up to pairwise dependencies for the optimization 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. The algorithm overperforms the Univariate Marginal Distribution Algorithm for the integer problems tested.

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)[4], 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) [8]. FDAs combine results from Graphical Models and Evolutionary Computation research, and are considered as a tractable subclass of Estimation Distribution Algorithms [9]. 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, or the use of simple probabilistic models. In [17]

the authors use a binary codification to represent integers in a pruefernumber encoding of a telecommunication tree network design. The Bayesian Optimization Algorithm [11] is used to solve the problem, the results did not show significant difference to other traditional GAs. FDAs that learn the structure of the probabilistic model when binary codification of integers is used face a major drawback. They learn a number of dependencies that are irrelevant to the problem, and thus add noise to the optimization process. On the other hand, when parametric FDAs are used with a binary codification of integers, the semantic links between the codified integer variables can be lost.

We [15] have employed an FDA that makes a parametric learning of trivariate marginals working on a binary codification of integers. Each integer is codified using three binary values, some integer values are illegal. The algorithm was used as an alternative to the Simple Genetic Algorithm (SGA) [4] in the context of the Classifier System XCS [22]. Results evidenced an improvement of the system's learning abilities when the FDA was used. However, as other changes were also incorporated to the XCS's discovery component, it is difficult to determine the exact contribution of the FDA to the improvements achieved.

Rosete [16] uses an Univariate Marginal Distribution Algorithm (UMDA) [7] 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.

Finally in [19] results are shown in the multi-way partitioning of graphs using the Bivariate Marginal Distribution Algorithm (BMDA) [14]. Authors acknowledged that BMDA performs good enough for the quadrisection, but for 6-way partition the number of fitness evaluations rapidly increases. In their experiments the cardinality of variables was limited to 6, besides, authors recognized that the relatively good results achieved for 6-way partition are due to the small size of the graph used for partitioning ($|V| = 36$).

BMDA belongs to a particular subclass of FDAs that considers up to pairwise dependencies among variables. These algorithms have been successfully applied to many problems [3][1][13][11][12], 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 $\frac{c^2 \cdot n \cdot (n-1)}{2}$. For the Traveling Salesman Problem (TSP) the memory requirements are of order n^4 where n is the number of cities to be visited.

In this paper we present an FDA that considers up to second order statistics and permits to carry out the optimization of integer problems with high cardinality of the variables. The algorithm reduces the storage requirements of previous

FDAs that employ pairwise dependencies by avoiding to store the bivariate probabilities.

2 GPR and UMDA

Gene Pool Recombination (GPR) was introduced in [10] 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 [10]. For binary functions the bit–based simulated crossover (BSC) of Syswerda [20] is similar to GPR. However, his implementation merged selection and recombination. GPR is an extension of BSC and it can be used for any representation, discrete or continuous.

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 simply 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

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

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

Summarizing, in the GPR the steps of model discovery and generation of new points are fusioned, but effects in the optimization are the same. It is important to highlight that the separation of these two steps was a relevant benefit the introduction of EDAs brought out.

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 general scheme of algorithms that consider pairwise dependencies can be seen in figure 1. The main differences (although not the only ones) among these algorithms are in the type of dependency graphs they employ and the way they are constructed (step 2 in the figure).

Now we briefly analyze the kind dependency graphs present in FDAs that use pairwise dependencies. We will focus on the ways they are constructed and their representational capabilities. Let us to formally introduce some definitions.

$I(X_i, X_j)$: Mutual Information between variables x_i and x_j.

$$I(X_i, X_j) = \sum_{x_i, x_j} p_{i,j}(x_i, x_j) \cdot \log \frac{p_{i,j}(x_i, x_j)}{p_i(x_i) \cdot p_j(xj)} \tag{2}$$

The Pearson's chi-square statistics [5] for independence of two random variables is given by:

$$\chi^2_{i,j} = \sum_{x_i,x_j} \frac{(Np_{i,j}(x_i,x_j) \ - \ Np_i(x_i)\cdot p_j(x_j))^2}{Np_i(x_i)\cdot p_j(x_j)} \tag{3}$$

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 descendant of the root. This process is repeated recursively until every vertex has been incorporated as a descendant of its parent. The parent of 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.

$$T(X_1,...,X_n) = \prod_{i=1}^{n} p(_{X_i|pa(X_i)}) \tag{4}$$

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

The Mutual Information Maximization for Input Clustering algorithm (MIMIC) [3] searches for connected graphs where each node has only one parent and only one son. It corresponds to a permutation of the problem's variables. MIMIC searches for the best permutation between the variables in order to find the probability distribution belonging to the class $P\pi(X)$ where:

$$P\pi(X) = \{p\pi(X)|p\pi(X) = p(X_{_1}|X_{_2})\cdot p(X_{_2}|X_{_3})...p(X_{n-1}|X_n)\cdot p(X_n)\}, \tag{5}$$

that is closest with respect to the Kullback-Leibler distance to the distribution of points in the selected set. This permutation can be found [3] by minimizing the function:

$$H\pi(X) = h(X_n) + \sum_{j=1}^{n-1} I(X_i,X_j) \tag{6}$$

where $h(X_n)$ is the entropy of variable X_n . The authors use a greedy algorithm to avoid the search over the $n!$ permutations.

Baluja and Davies [1] present an optimization algorithm that uses a probabilistic model defined on trees. They use 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.

Finally, the BMDA expands the representation capabilities of previous algorithms by allowing their acyclic graphical models to be unconnected. They are set of trees mutually non connected (forest). Pelikan and Muehlenbein use a rather different approach to calculate this graph. The χ^2 is calculated for each pair of variable (we will call M to the matrix that keeps χ^2 values for all pairs of variables). Thereafter, the acyclic graph is constructed using edges whose χ^2 value indicates X_i and X_j are dependent in a 95%.

It is important to note that the bivariate probability between X_i and X_j is only needed to calculate $I(X_i, X_j)$ or $M(X_i, X_j)$. It does not need to be saved for later calculus (at least during this model learning step). Hence, the algorithm that we present later uses the following procedure to calculate the mutual information matrix I (alternatively M).

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 $\theta(n^2N)$. We go back to this issue in the section of experiments in order to show how as generations pass the algorithm speeds. From now on we will refer only to the matrix of Mutual Information I, but we emphasize the algorithm is basically the same when we use M or any other measure of pairwise dependencies.

Once 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 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 [6], 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 bivariate probabilities.

4 Vector generation

Before to present the main results of this section we illustrate how the cardinality of variables influence the storage requirements of bivariate probabilities. Let us consider the following scenario. We have two problems defined on integers.

Problem I: There are n variables of cardinality a^q. $n, a, q \in N^+$, $n > 1$, $a > 1$, $q > 1$.

Problem II: There are $n \cdot q$ variables of cardinality a.

The cardinality of the search space of both problems is the same $a^{n\cdot q}$. Nevertheless the storage requirements (SR) of their bivariate marginal probabilities

Algorithm for finding the mutual information

INPUT : Population P of size N
OUTPUT: Matrix I of mutual information
NOTES: Ind is an auxiliary array of vectors indexes in P.
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

are different.

Problem I: $SR_I = \frac{a^{2 \cdot q} \cdot n \cdot (n-1)}{2}$

Problem II: $SR_{II} = \frac{a^2 \cdot n \cdot q \cdot (n \cdot q - 1)}{2}$

SR_I grows faster than SR_{II} when a or q grows, keeping fixed the other variables. It means that bivariate models with higher cardinality of the variables (we consider the dominion of models that have the same cardinality of the search space and all the variables have the same cardinality) needs more memory to be saved.

This fact explains, for instance, that it is not rare to find solutions to binary problems of 60 variables ($SR = 7080$) but it is however very difficult to design an efficient FDA that covers pairwise probabilities, for the solution of integer problems of only 10 variables, 64 values each ($SR = 184320$). Note that in both cases the cardinality of the search space is 2^{60}.

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 at it is done for all 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.

We will store this information in the tables $PopulValues$ and $ParentIndices$. 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 indexes of vectors in P ordered 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 cero in P), then values $(2, 3)$ correspond to indices where $x_1 = 2$, and so on.

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

$a)$

P = Population, PV = PopulationValues and PI = ParentIndices.

$$P = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ 4 & 2 & 4 & 1 & 2 & 0 & 1 & 1 \\ 2 & 2 & 1 & 2 & 2 & 4 & 0 & 3 \\ 2 & 2 & 3 & 1 & 2 & 3 & 0 & 3 \\ 1 & 0 & 0 & 2 & 2 & 2 & 3 & 3 \\ 1 & 1 & 0 & 0 & 2 & 1 & 0 & 0 \end{bmatrix} \quad PV = \begin{bmatrix} x_1 & x_5 & x_6 \\ 4 & 1 & 1 \\ 5 & 2 & 5 \\ 2 & 3 & 4 \\ 3 & 4 & 3 \\ 1 & 5 & 2 \end{bmatrix} \quad PI = \begin{bmatrix} x_1 & x_5 & x_6 \\ -1 & -1 & 1 \\ 2 & -1 & 2 \\ 4 & 5 & 3 \\ 4 & 5 & 4 \\ 5 & 5 & 5 \end{bmatrix}$$

$b)$ (7)

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

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 $n-1$ 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 $(n-1)\cdot(N(S)+c) \ll \frac{c^2 \cdot n \cdot (n-1)}{2}$.

The algorithm for the generation of vectors is shown in figure 4. We present in figure the Int-Tree algorithm that uses the procedures introduced in previous sections.

In order to use another dependency measures only the step 2 of the algorithm has to be changed.

5 Experiments

The first group of experiments were aimed to compare the behavior of the algorithm with previous methods that use bivariate probabilities for binary problems. The second group of experiments was designed in order to show the behavior of

Algorithm for the generation of vectors

INPUT: P, $PopulValues$, $ParentIndices$, $Tree$
OUTPUT: Vector.
BEGIN Vector Generation

Instanstiate 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 $ParentIndices(k,i)$;

$X_j = PopulValues(ParentIndices(k-1,i)+r\ ,j)$;
IF there exists any variable that has not
been already instantiated.
Go to step 2;
END

END
END Vector Generation

Figure 4: Algorithm for the generation of vectors

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: FDA that uses up to second order dependencies for problems with integer representation

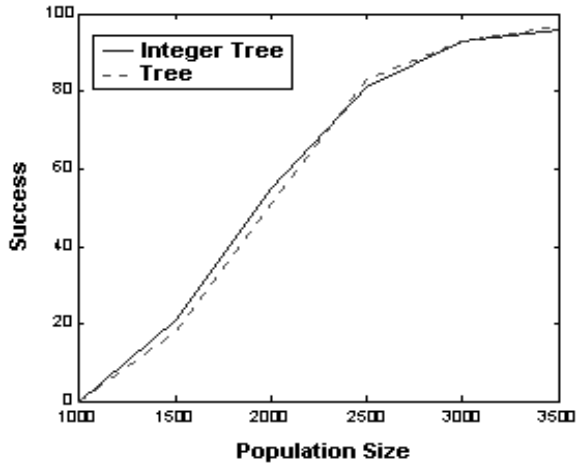

Figure 6: Times the optimum of the checkerboard function was reached for Integer trees and tree algorithm.

the proposed algorithm on non-overlapping decomposable problems with deceptive building blocks.

As optimization of problems defined on integers have been exceptionally treated in the FDA literature we introduce a number of functions defined on integers. These functions extend the idea of unitation functions [13] to integer variables.

We will first present results of the comparison between the introduced algorithm with a tree based FDA that stores the bivariate probabilities of a binary problem.

There were no significant differences in the results for the functions tested. Figure 6 presents the results achieved for the checkerboard problem. The problem was originally introduced in [1]. The goal of the problem is to create a checkerboard pattern of 0's and 1's in an NxN grid. Only the primary four directions are considered in the evaluation. For each bit in an (N-2)x(N-2) grid centered in an (NxN) grid +1 is added for each of the 4 neighbors that are set to the opposite value. In our experiments N was set to 12.

We use Truncation selection, and the elitism strategy of best selection. Best selection adds to the next population all the points selected in the current generation. Figure 6 shows the times the optimum was found for different population sizes. Curves are very near each other. In Figure 7 the time spent by both algorithms is shown, while the time spent by the algorithm that keeps bivariate marginals is almost constant, the time spent by our algorithm decrease through generations. Nevertheless, as expected, the time consumed by the proposed algorithm is higher.

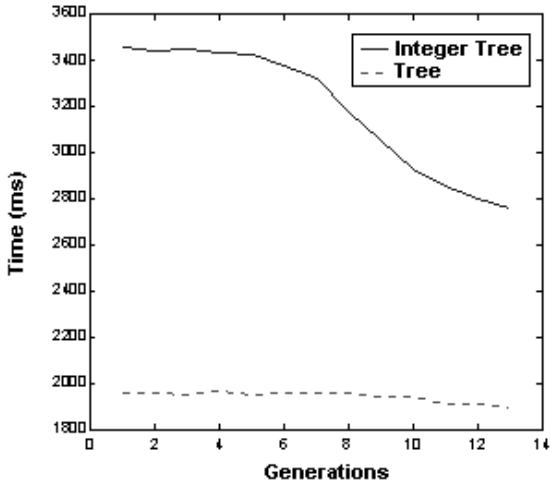

Figure 7: Elapsed time for the integer trees and trees algorithm.

5.1 Integer Problems

5.1.1 Test Functions

We evaluate the algorithm in a number of functions defined on integers.

- The IntMax function (generalization of onemax function)

$$IntMax(X) = \sum_{i=1}^{n} x_i \tag{8}$$

Results are also presented 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, and

$$F_{dec(X,k,p)} = \begin{cases} k \cdot c & for\ IntMax(X) = k \cdot c \\ k \cdot c - IntMax(X) - 1 & otherwise \end{cases} \tag{9}$$

Table II shows a comparison of the proposed algorithm with the UMDA for the $intdecp_2^n$ function $n = (10, ..., 90)$, $c = 10$. Table III 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 elitism. In the tables, entries represent the mean fitness of 50 runs for the UMDA and the proposed algorithm.

The mean of the suboptima found is 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 that 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

Tables II,III: Comparison between the Tree-Int algorithm and the UMDA for $intdecp_2^n$. Table II (left): when the number or variables is increased. Table III (right): the cardinality of variables is increased.

5.2 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 principle 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[21].

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. Figure 8 shows an example of the alignment of 6 initial sequences. Note that spaces have been inserted in the sequences in order to improve the 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.

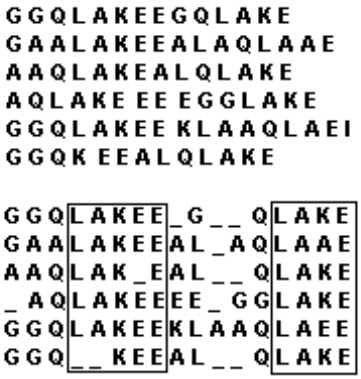

Figure 8: An initial set of 6 sequences and a possible optimal multiple alignment.

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 we try to match in the sequences. 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 $[|W|, L_i - |W|]$ 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 [18] 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 in position j of the window located in 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:

$$f(b, j) = \frac{n(b, j)}{n} \tag{10}$$

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

$$F(S_i) = |W| \cdot \log_2 |\Omega| + \sum_{j=1}^{|W|} \sum_{b \in \Omega} f(b, j) \cdot \log_2 f(b, j) \tag{11}$$

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

	Convergence				Mean Fitness			
	$UMDA$		$Int-Tree$		$UMDA$		$Int-Tree$	
	5	10	5	10	5	10	5	10
$\|W\|=3$	30	30	30	30	9	9	9	9
$\|W\|=4$	15	0	30	0	11.33	11.67	12	11.85
$\|W\|=5$	0	0	23	15	14.37	14.37	14.83	14.42
$\|W\|=6$	0	0	0	0	17.1	16.43	17.61	16.9
$\|W\|=9$	0	0	0	0	24.92	23.43	24.94	23.91
$\|W\|=12$	5	0	29	7	34.73	35.61	35.94	34.19

Tables IV: Comparison between the Int-Tree algorithm and the UMDA in the MSA problem.

In our experiments a number of characteristics of the MSA problem were identified that influence the complexity of the search. The total number of sequences to be aligned, the cardinality of the alphabet, the sequences' configuration, and the width of the window are all factors that make the selection of an appropriate test problem a difficult question.

In Table IV we present results achieved for an artificial generated set of 15 sequences defined on an alphabet of cardinality 4. Sequences were generated to contain many partial alignments of short size but only one of size 12. In table IV 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 size used was of width 12. This could be explained because the existence of many partial solutions when the windows are small. In this case there are many suboptima that lead the search to different, some time opposite, directions. When the window is of width 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 is an interesting benchmark problem for the FDAs that work on integer representation. This 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. This algorithm has direct application to the field of Genetic Programing and constrained optimization on integers. Besides, it can be improved by means of using mixtures distributions, a path we pretend to follow in the future.

References

[1] S. Baluja and S. 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. volume IT14, pages 462–467, 1968.

[3] J. S. De Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in neural information processing systems*, volume 9, page 424. The MIT Press, Cambridge, 1997.

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

[5] L. A. Marascuilo and M. McSweeney. *Nonparametric and distribution-free methods for the social sciences*. Brooks/Cole publishing company, CA, 1977.

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

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

[8] H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.

[9] 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.

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

[11] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA.

[12] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Bayesian optimization algorithm, population sizing, and time to convergence. IlliGAL Report No. 2000001, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2000.

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

[14] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[15] J. P. Rivera and R. Santana. Design of an algorithm based on the estimation of distributions to generate new rules in the xcs classifier system. Technical Report ICIMAF 2000-100, CEMAFIT 2000-78, Institute of Cybernetics, Mathematics and Physics, Havana, Cuba, June 2000.

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

[17] F. Rothlauf, D. E. Goldberg, and A. Heinzl. Genetic algorithms, problem difficulty, and the modality of fitness landscapes. IlliGAL Report No. 200007, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2000.

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

[19] J. Schwarz and J. 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.

[20] G. Syswerda. Simulated crossover in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 239–295. Morgan Kaufmann, 1993.

[21] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleid Acid Research*, 22(22):4673–4680, 1994.

[22] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.