Parallel Implementation of EDAs Based on Probabilistic Graphical Models

Alexander Mendiburu, Jose A. Lozano, and José Miguel-Alonso

***Abstract*—This paper proposes new parallel versions of some estimation of distribution algorithms (EDAs). Focus is on maintenance of the behavior of sequential EDAs that use probabilistic graphical models (Bayesian networks and Gaussian networks), implementing a master–slave workload distribution for the most computationally intensive phases: learning the probability distribution and, in one algorithm, "sampling and evaluation of individuals." In discrete domains, we explain the parallelization of $\mathrm{EBNA_{BIC}}$ and $\mathrm{EBNA_{PC}}$ algorithms, while in continuous domains, the selected algorithms are $\mathrm{EGNA_{BIC}}$ and $\mathrm{EGNA_{EE}}$.**

Implementation has been done using two APIs: message passing interface and POSIX threads. The parallel programs can run efficiently on a range of target parallel computers. Experiments to evaluate the programs in terms of speed up and efficiency have been carried out on a cluster of multiprocessors. Compared with the sequential versions, they show reasonable gains in terms of speed.

***Index Terms*—Cluster computing, estimation of distribution algorithms (EDAs), evolutionary computation, performance evaluation, probabilistic graphical models.**

I. Introduction

IN RECENT YEARS, great improvements have been made in the development and use of heuristic techniques for optimization. Different algorithms have been proposed and applied to optimization problems, usually with very good results. However, there is still a significant drawback, which, in many situations, makes the use of such algorithms prohibitive in real environments: computation time. That is, the algorithms are able to solve a complex problem but not as quickly as we need, with execution times lasting hours or even days. One approach to reduce computation time is the use of parallelization techniques.

Some of the algorithms that have received much attention from researchers are those included in the field of evolutionary computation. Such techniques include genetic algorithms (GAs), evolutionary strategies, evolutionary programming, and genetic programming. A number of interesting proposals for parallel evolutionary computation can be found in the literature. Efforts in the design of parallel evolutionary algorithms have focused on two areas: 1) looking for algorithms where many populations evolve in parallel and 2) parallelization of the evaluation of the objective function. A good summary of different ideas for designing parallel evolutionary algorithms can be found in [1] and [2].

Recently, a new group of algorithms has been proposed in the field of evolutionary computation: estimation of distribution algorithms (EDAs) [3], [4]. Like most evolutionary computation heuristics, EDAs maintain at each step a population of individuals but, instead of using crossover and mutation operators to evolve the population, EDAs learn a probability distribution from the set of selected individuals and sample this distribution to obtain the new population.

One drawback of these algorithms is that the estimation of the joint probability distribution can easily become a computational bottleneck. To lessen this problem, several authors have proposed simplifying assumptions on the probability model to be learnt at each step, related to the way the model takes into account relations between variables. Three groups could be differentiated: algorithms that assume all the variables are independent, those that assume second-order statistics, and those that consider unrestricted models.

In the field of EDAs, the algorithms that obtain the best results are those of the last group (probability models with unrestricted dependencies [4]). In these algorithms, a probabilistic graphical model codifies the probability distribution. Excluding the computation time of the objective function, the main computational cost of the algorithms is the time consumed in learning the probabilistic graphical model at each step. In fact, it ranges from 50% to 99% of the total execution time. Therefore, our effort focuses on the parallelization of this learning process, proposing different solutions for certain EDAs that use probabilistic graphical models to learn the joint probability distribution of the selected individuals. There is also another phase that could require an important amount of time during the execution: sampling. In this paper, the parallelization of the "sampling and evaluation" step is presented for the $\mathrm{EGNA_{EE}}$ algorithm although it can easily be adapted to any of the others.

In the field of EDAs, previous parallel implementations can be found for some algorithms. In [5], two different parallel proposals for an algorithm that uses probabilistic graphical models in the combinatorial field ($\mathrm{EBNA_{BIC}}$) are proposed. These implementations use threads on a multiprocessor with shared memory, so that all data structures are shared between the different processes, with each of them computing a piece of work.

Also interesting is the work done in [6] and [7], where two different parallel versions of the Bayesian optimization algorithm

Manuscript received October 16, 2003; revised July 7, 2004 and January 14, 2005. This work was supported in part by the University of the Basque Country,Gipuzkoa, Spain, under Grant 9/UPV/EHU 00140.226-15334/2003, in part by the Basque Government under Grant ETORTEK-BIOLAN and Grant SAIOTEK, and in part by the Ministerio de Educacion y Ciencia, Spain, under Grant TIN 2004-07740-C02-02.

A. Mendiburu and J. Miguel-Alonso are with the Department of Computer Architecture and Technology, University of the Basque Country, Gipuzkoa 20018, Spain (e-mail: amendiburu@si.ehu.es).

J. A. Lozano is with the Department of Computer Science and Artificial Intelligence, University of the Basque Country, Gipuzkoa 20018, Spain.

Digital Object Identifier 10.1109/TEVC.2005.850299

(BOA) [8] are proposed using a pipelined parallel architecture and clusters of computers respectively. Recently, in [9], the parallelization of the learning of decision trees using multithreaded techniques has been proposed.

Another parallelization approach can be found in [10], where a basic framework that facilitates the development of new multiple-deme parallel EDAs is presented.

In this paper, we propose new parallel versions of certain EDAs that use probabilistic graphical models to learn the probability distribution of the selected individuals in both discrete and continuous domains. Specifically, we have chosen one algorithm for each different method of learning a probabilistic graphical model used in EDAs. Our main goal has been to implement different parallel versions for each algorithm, trying to reduce the execution time, while maintaining exactly the same behavior of the sequential version. In this way, we provide parallel algorithms that are able to be applied on problems that require an inaccessible execution time for the sequential versions.

Implementation has been carried out mixing two parallel application programming interfaces: message passing interface (MPI) [11] and POSIX threads [12].

Concerning the algorithms, $\mathrm{EBNA_{BIC}}$ and $\mathrm{EBNA_{PC}}$ have been parallelized in the discrete domain. Previous parallel works can be consulted in this domain, such as the proposal of [5], where a parallel version of $\mathrm{EBNA_{BIC}}$ is presented (using threads). Our work presents a parallel solution that can be used in a multiprocessor computer (where processes communicate via shared variables), in a cluster (where processes communicate using messages), or any combination of both scenarios. Also, our work extends [6] and [7] allowing the learning of the Bayesian network without taking into account order restrictions. In the continuous domain, parallel versions of $\mathrm{EGNA_{BIC}}$ and $\mathrm{EGNA_{EE}}$ algorithms are proposed.

The rest of this paper is organized as follows. A description of EDAs can be seen in Section II, where the main characteristics of the algorithms will be explained. In Section III, general considerations for the parallel proposals are presented. In Sections IV and V, the parallel solution for the four different algorithms is explained following this schema: a brief explanation of the algorithm, parallel implementation(s) and an evaluation of the results obtained. Finally, conclusions, as well as some ideas for future work are presented in Section VI.

II. Estimation of Distribution Algorithms

EDAs were introduced in the field of evolutionary computation in [3], although similar approaches can be found in [13]. In EDAs, there are neither crossover nor mutation operators. Instead, the new population of individuals is sampled from a probability distribution, which is estimated from a database that contains the selected individuals from the previous generation. Thus, the interrelations between the different variables that represent the individuals are explicitly expressed through the joint probability distribution associated with the individuals selected at each generation. In order to better understand the behavior of this heuristic, a common outline for all EDAs follows.

Step 1) Generate the first population of M individuals and evaluate each of them. Usually, this generation is made assuming a uniform distribution on each variable.

Step 2) N individuals are selected from the set of M, following a given selection method.

Step 3) A n (size of the individual) dimensional probability model that shows the interdependencies among the variables is induced from the N selected individuals.

Step 4) Finally, a new population of M individuals is generated based on the sampling of the probability distribution learnt in the previous step.

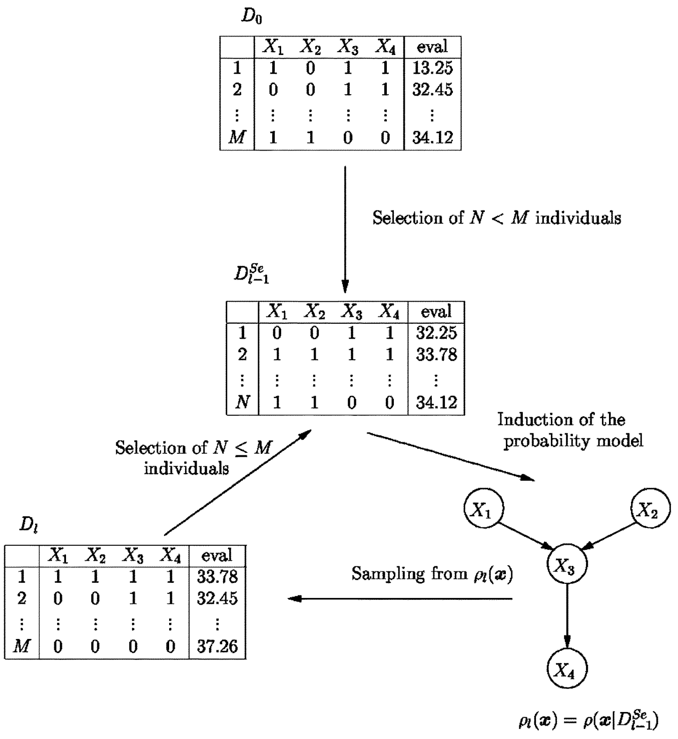

D_0
	X_1	X_2	X_3	X_4	eval
1	1	0	1	1	13.25
2	0	0	1	1	32.45
⋮	⋮	⋮	⋮	⋮	⋮
M	1	1	0	0	34.12
Selection of $N < M$ individuals
D^{Se}_{l-1}
	X_1	X_2	X_3	X_4	eval
1	0	0	1	1	32.25
2	1	1	1	1	33.78
⋮	⋮	⋮	⋮	⋮	⋮
N	1	1	0	0	34.12
Induction of the probability model
Selection of $N \leq M$ individuals
D_l
	X_1	X_2	X_3	X_4	eval
1	1	1	1	1	33.78
2	0	0	1	1	32.45
⋮	⋮	⋮	⋮	⋮	⋮
M	0	0	0	0	37.26
Sampling from $p_l(\boldsymbol{x})$
X_1
X_2
X_3
X_4
$p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D^{Se}_{l-1})$

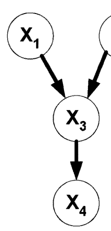
Fig. 1. EDA approach to optimization.

Steps 2)–4) are repeated until some stop criterion is met (e.g., a maximum number of generations, a homogeneous population, or no improvement after a certain number of generations). A diagram of this process (for $n = 4$) can be seen in Fig. 1.

The probabilistic graphical model learnt at each step has a significant influence on the behavior of the EDA (computing times and obtained results). Below, we provide a classification of EDAs that uses as criterion the complexity of this probability model and the dependencies it considers.

- Without dependencies: It is assumed that the n-dimensional joint probability distribution factorizes as a product of n univariate and independent probability distributions. Algorithms that use this model are, among others, UMDA [14] or PBIL_c [15].
- Bivariate dependencies: Only the dependencies between pairs of variables are taken into account. In this way, estimation of the joint probability can be already done quickly. This group includes MIMIC [16], MIMIC_c [17], [18].
- Multiple dependencies: All possible dependencies among the variables are considered without taking into account

Structure



Local probabilities

$$\begin{array}{ll}
\boldsymbol{\theta_1} = (\theta_{1-1}, \theta_{1-2}) & p(x_1^1), p(x_1^2) \\
\boldsymbol{\theta_2} = (\theta_{2-1}, \theta_{2-2}, \theta_{2-3}) & p(x_2^1), p(x_2^2), p(x_2^3) \\
\boldsymbol{\theta_3} = (\theta_{311}, \theta_{321}, \theta_{331}, & p(x_3^1|x_1^1, x_2^1), p(x_3^1|x_1^1, x_2^2), p(x_3^1|x_1^1, x_2^3), \\
\qquad \theta_{341}, \theta_{351}, \theta_{361}, & p(x_3^1|x_1^2, x_2^1), p(x_3^1|x_1^2, x_2^2), p(x_3^1|x_1^2, x_2^3), \\
\qquad \theta_{312}, \theta_{322}, \theta_{332}, & p(x_3^2|x_1^1, x_2^1), p(x_3^2|x_1^1, x_2^2), p(x_3^2|x_1^1, x_2^3), \\
\qquad \theta_{342}, \theta_{352}, \theta_{362}, & p(x_3^2|x_1^2, x_2^1), p(x_3^2|x_1^2, x_2^2), p(x_3^2|x_1^2, x_2^3), \\
\boldsymbol{\theta_4} = (\theta_{411}, \theta_{421}, \theta_{412}, \theta_{422}) & p(x_4^1|x_3^1), p(x_4^1|x_3^2), p(x_4^2|x_3^1), p(x_4^2|x_3^2)
\end{array}$$

Factorization of the joint mass-probability

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3|x_1, x_2)p(x_4|x_3)$$

Fig. 2. Structure, local probabilities, and resulting factorization for a Bayesian network with four variables (X_1, X_3 and X_4 with two possible values, and X_2 with three possible values).

required complexity. $\mathrm{EBNA_{BIC}}$ [19], [20], BOA [8], [21]–[23], learning factorized distribution algorithm (LFDA) [24], or $\mathrm{EGNA_{EE}}$ [17], [18] are some algorithms that belong to this group.

For detailed information about the characteristics and different algorithms that constitute the family of EDAs, see [4] and [25]. Theoretical results related to convergency and stability properties of EDAs can be consulted in [26]–[28].

The algorithms of the last group (multiple dependencies) use different paradigms to codify the probabilistic model. Particularly, we are interested in those that use probabilistic graphical models based on Bayesian or Gaussian networks and, therefore, we will explain briefly the main characteristics of both types of networks.

1) Bayesian Networks: Formally, a Bayesian network [29] over a domain $\mathbf{X} = (X_1, \ldots, X_n)$ is a pair $(S, \boldsymbol{\theta})$ that represents a graphical factorization of a probability distribution. The structure S is a directed acyclic graph which reflects the set of conditional (in)dependencies between the variables. The factorization of the probability distribution is codified by S

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\boldsymbol{pa_i}) \qquad (1)$$

where $\boldsymbol{Pa_i}$ is the set of parents of X_i (variables from which there exists an arc to X_i in the graph S). In Fig. 2, for example, $\boldsymbol{Pa_3} = \{\boldsymbol{X_1}, \boldsymbol{X_2}\}$ (X_1 and X_2 are the parents of X_3). The second part of the pair, $\boldsymbol{\theta}$, is a set of parameters for the local probability distributions associated with each variable. If variable X_i has r_i possible values, $x_i^1, \ldots, x_i^{r_i}$, the local distribution, $p(x_i|\boldsymbol{pa_i^j}, \boldsymbol{\theta}_i)$ is an unrestricted discrete distribution

$$p\left(x_i^k|\boldsymbol{pa_i^j}, \boldsymbol{\theta}_i\right) \equiv \theta_{ijk} \qquad (2)$$

where $\boldsymbol{pa_i^1}, \ldots, \boldsymbol{pa_i^{q_i}}$ denote the values of $\boldsymbol{Pa_i}$ and the term q_i denotes the number of possible different instances of the parent variables of X_i. In other words, parameter θ_{ijk} represents the conditional probability of variable X_i being in its kth value, knowing that the set of its parent variables is in its jth value. Therefore, the local parameters are given by $\boldsymbol{\theta}_i = (((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i})$ $i = 1, \ldots, n$. An example of a Bayesian network can be seen in Fig. 2.

The learning phase of EDAs involves the learning of a Bayesian network from the selected individuals at each generation, that is, learning the structure (the graph S) and the parameters (the local probability distribution $\boldsymbol{\theta}$). There are different methods to complete this learning process, including for example the "score + search" and the "detecting conditional (in)dependencies" approaches. In the following section (discrete domain), we propose the parallelization of two EDAs that use these methods: $\mathrm{EBNA_{BIC}}$ and $\mathrm{EBNA_{PC}}$, respectively.

2) Gaussian Networks: The other particular case of probabilistic graphical model considered in this work is of use when each variable $X_i \in \boldsymbol{X}$ is continuous and each local density function is the linear-regression model

$$f(x_i|\boldsymbol{pa_i}, \boldsymbol{\theta_i}) \equiv \mathcal{N}\left(x_i; m_i + \sum_{x_j \in \boldsymbol{pa_i}} b_{ji}(x_j - m_j), v_i\right) \qquad (3)$$

where $\mathcal{N}(x; \mu, \sigma^2)$ is a univariate normal distribution with mean μ and variance σ^2. Given this form, a missing arc from X_j to X_i implies that $b_{ji} = 0$ in the former linear-regression model. The local parameters are given by $\boldsymbol{\theta}_i = (m_i, \boldsymbol{b}_i, v_i)$, where $\boldsymbol{b}_i = (b_{1i}, \ldots, b_{i-1i})^t$ is a column vector. A probabilistic graphical model constructed with these local density functions is a Gaussian network [30].

The interpretation of the components of the local parameters is as follows: m_i is the unconditional mean of X_i, v_i is the conditional variance of X_i given $\boldsymbol{Pa}_i$, and b_{ji} is a linear coefficient reflecting the strength of the relationship between X_j and X_i.

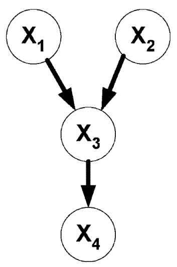
It can be seen that there is a bijection between Gaussian networks and a n-dimensional multivariate normal distribution, whose density function can be written as

$$f(\boldsymbol{x}) \equiv \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) \equiv (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})} \qquad (4)$$

where $\boldsymbol{\mu}$ is the vector of means, Σ is an $n \times n$ covariance matrix, and $|\Sigma|$ denotes the determinant of Σ. An example of a Gaussian network can be seen in Fig. 3.

Like for Bayesian networks, there are also different methods to complete Gaussian network learning. The ones we point out

Structure



Local densities

$$\boldsymbol{\theta_1} = (m_1, -, v_1) \quad f_{X_1} \hookrightarrow \mathcal{N}(x_1; m_1, v_1)$$

$$\boldsymbol{\theta_2} = (m_2, -, v_2) \quad f_{X_2} \hookrightarrow \mathcal{N}(x_2; m_2, v_2)$$

$$\boldsymbol{\theta_3} = (m_3, \boldsymbol{b_3}, v_3) \quad f_{X_3|X_1=x_1, X_2=x_2} \hookrightarrow$$
$$\boldsymbol{b_3} = (b_{13}, b_{23})' \quad \mathcal{N}(x_3; m_3 + b_{13}(x_1 - m_1) + b_{23}(x_2 - m_2), v_3)$$

$$\boldsymbol{\theta_4} = (m_4, \boldsymbol{b_4}, v_4) \quad f_{X_4|X_3=x_3} \hookrightarrow$$
$$\boldsymbol{b_4} = (b_{34})' \quad \mathcal{N}(x_4; m_4 + b_{34}(x_3 - m_3), v_4)$$

Factorization of the joint density function

$$f(x_1, x_2, x_3, x_4) = f(x_1)f(x_2)f(x_3|x_1, x_2)f(x_4|x_3) =$$
$$\frac{1}{\sqrt{2\pi v_1}} e^{-\frac{1}{2v_1}(x_1 - m_1)^2} \frac{1}{\sqrt{2\pi v_2}} e^{-\frac{1}{2v_2}(x_2 - m_2)^2}$$
$$\frac{1}{\sqrt{2\pi v_3}} e^{-\frac{1}{2v_3}(x_3 - (m_3 + b_{13}(x_1 - m_1) + b_{23}(x_2 - m_2)))^2}$$
$$\frac{1}{\sqrt{2\pi v_4}} e^{-\frac{1}{2v_4}(x_4 - (m_4 + b_{34}(x_3 - m_3)))^2}$$

Fig. 3. Structure, local probabilities, and resulting factorization for a Gaussian network with four variables.

TABLE I
TIME MEASUREMENT (%) OF THE LEARNING PHASE FOR DIFFERENT ALGORITHMS AND PROBLEMS

	individual sizes		
Algorithm, Problem	**150**	**250**	**500**
$EBNA_{BIC}$, OneMax	98.6	99.4	99.6
$EBNA_{PC}$, OneMax	98.0	99.1	99.6
Algorithm, Problem	**50**	**70**	**100**
$EGNA_{BIC}$, Sphere model	98.3	98.6	99.9
Algorithm, Problem	**500**	**1,000**	**1,500**
$EGNA_{EE}$, Sphere model	51.3	58.8	69.6

are "score + search" and "detecting conditional (in)dependencies." In Sections V-A and V-B, the parallelization of two algorithms that use these approaches are shown: $\mathrm{EGNA_{BIC}}$ ("score + search") and $\mathrm{EGNA_{EE}}$ (edge exclusion tests).

III. GENERAL CONSIDERATIONS FOR PARALLEL PROPOSALS

Before undertaking the implementation of a parallel solution for an existing sequential algorithm, it is essential to study the structure and functionality of the sequential solution and find the parts amenable to parallelization. To this end, we have executed the sequential version of the algorithms with different individual sizes in order to measure the computation time that each part of the algorithm requires. As we have said in the introduction and can be seen in Table I (showing the time required to complete the learning step in percentages), the most time-consuming process is the learning phase, which generally accounts for 50%–99% of total execution time. Our effort will therefore focus on understanding how the learning process proceeds and finding a good parallel approach for this phase.

It must also be pointed out that, for particular problems where the evaluation of an individual is also processor-intensive, the sampling (creation of new individuals) and evaluation phase must also be parallelized in order to obtain a good parallel solution. As a preliminary work, in this paper, we introduce the parallelization of this step for a discrete algorithm ($\mathrm{EGNA_{EE}}$).

Before explaining the implementation, two important concepts should be introduced: MPI and POSIX threads.

MPI [11] is an API that can be used by an application process to communicate over the network with processes that are running in other computers. It was designed and standardized by the MPI forum, a group of academic and industrial experts, to be used in very different types of parallel computers. A wide set of functions are available for managing, sending and receiving messages.

In relation to POSIX threads [12], most operating systems have thread support, that is, they include capabilities to spawn new "lightweight processes" (actually, threads) that share the same variable space. With this paradigm, a set of processes collaborating to perform a common task are implemented as threads that share the same memory space, so they can communicate directly using a set of global variables. The advantage of using this approach is that communication is fast and efficient, although the synchronization between threads has to be explicitly implemented.

For the implementation of the programs, we have chosen a two-level master–slave scheme (Fig. 4). At the first level (communication between computers), the communication is carried out using MPI, where a computer acts as a manager that distributes workload among the remaining computers (workers). These workers execute orders sent by the manager and return results to it. In some situations, the manager must stay idle for a long time, waiting until all the workers finish their work. To make good use of this time, it is possible to make the manager act as another worker. At the second level, we present another master–slave schema (inside each computer), where the MPI

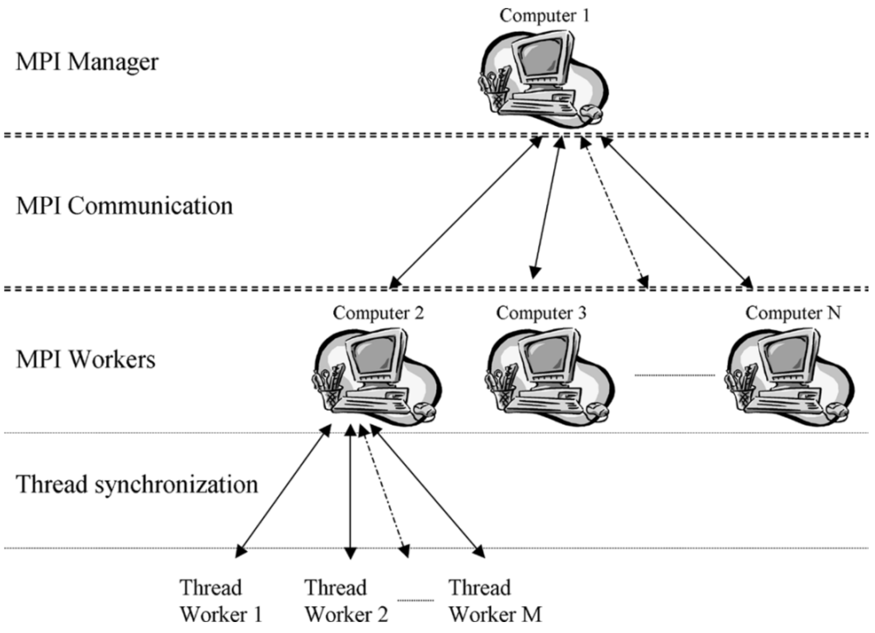


Fig. 4. Two-level manager–worker schema.

process (one per computer) is the master, and the workers are the multiple threads that can be created. This is a good choice when computers are multiprocessors. For reasons that will be explained later, the implementation of the parallel $\mathrm{EBNA_{PC}}$ algorithm is made using a single-level master–slave scheme.

The rationale for the two-level schema is that, in many environments, machines available for scientific computing are an assortment of multiprocessors, clusters of single-processor nodes or even clusters of multiprocessors. Our programs can be easily adapted to any of these environments by selecting the right choice of mechanism for internode and intranode communication and synchronization.

It should be noted that, to simplify the explanations, all the pseudocodes shown in this paper describe only the first level (MPI communication between computers). The schemes for the second level are similar to those of the first, although there are no send/receive operations because communication is performed via a shared memory space.

Finally, it is very important to remark that, throughout this paper, performance results for the parallel implementations (in terms of efficiency and/or speedup) are compared to an optimized sequential version of the programs, running on a single processor.

IV. Discrete Domain

A. $\mathrm{EBNA_{BIC}}$ *Algorithm [19], [20]*

1) Algorithm Description: This algorithm follows the common schema described for EDAs.

In the learning phase, it uses a "score + search" approach to learn the structure of the probabilistic graphical model (Bayesian network). This method assigns a score to each possible Bayesian network structure, that is a measure of its performance given a data file of cases. To look for a structure that maximizes the given score it uses an algorithm that basically consists of adding or deleting edges to the existing Bayesian network.

$\mathrm{EBNA_{BIC}}$ uses the penalized maximum-likelihood score denoted by BIC (Bayesian information criterion) [31]. Given a structure S and a dataset D (set of selected individuals), the BIC score can be written as

$$\mathrm{BIC}(S,D)=\sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i}N_{ijk}\log\frac{N_{ijk}}{N_{ij}}-\frac{1}{2}\log N\sum_{i=1}^{n}q_i(r_i-1) \tag{5}$$

where

- n is the number of variables of the Bayesian network (size of the individual);
- r_i is the number of different values that variable X_i can take;
- q_i is the number of different values that the parent variables of X_i, $\boldsymbol{Pa_i}$, can take;
- N_{ij} is the number of individuals in D in which variables $\boldsymbol{Pa_i}$ take their jth value;
- N_{ijk} is the number of individuals in D in which variable X_i takes its kth value and variables $\boldsymbol{Pa_i}$ take their jth value.

An important property of this score is that it is decomposable. This means that it can be calculated as the sum of the separate local BIC scores for the variables, that is, each variable X_i has associated with it a local BIC score $(\mathrm{BIC}(i,S,D))$

$$\mathrm{BIC}(S,D)=\sum_{i=1}^{n}\mathrm{BIC}(i,S,D) \tag{6}$$

where

$$\mathrm{BIC}(i,S,D)=\sum_{j=1}^{q_i}\sum_{k=1}^{r_i}N_{ijk}\log\frac{N_{ijk}}{N_{ij}}-\frac{1}{2}\log(N)q_i(r_i-1). \tag{7}$$

The structure search algorithm used in $\mathrm{EBNA}_{\mathrm{BIC}}$ is usually a hill-climbing greedy search algorithm. At each step, an exhaustive search is made through the set of possible arc modifications. An arc modification consists of adding or deleting an arc from the current structure S. The arc modification that maximizes the gain of the BIC score is used to update S, provided that it results in a directed acyclic graph (DAG) structure (note that the structure of a Bayesian network must always be a DAG). This continues until there is no arc modification that improves the score. It is important to bear in mind that if we update S with the arc modification (j, i), then only $\mathrm{BIC}(i, S, D)$ needs to be recalculated.

The structural learning algorithm involves a sequence of actions that differs between the first step and all subsequent steps. In the first step, given a structure S and a database D, the change in the BIC is calculated for each possible arc modification. Thus, we have to calculate $n(n-1)$ terms as there are $n(n-1)$ possible arc modifications. The arc modification that maximizes the gain of the BIC score, whilst maintaining the DAG structure, is applied to S. In the remaining steps, only changes to the BIC due to arc modifications related to the variable X_i need to be considered (it is assumed that in the previous step, S was updated with the arc modification (j, i)). Other arc modifications have not changed its value because of the decomposable property of the score. In this case, the number of terms to be calculated is $n - 2$.

We use four data structures for this algorithm. A vector $\mathrm{BIC}[i]$, $i = 1, 2, \ldots, n$, where $\mathrm{BIC}[i]$ stores the local BIC score of the current structure associated with variable X_i. A structure $S[i]$, $i = 1, 2, \ldots, n$, with the DAG represented as adjacency lists, that is, $S[i]$ represents a list of the immediate successors of vertex X_i. A $n \times n$ matrix $G[j, i]$, $i, j = 1, \ldots, n$, where each (j, i) entry represents the gain or loss in score associated with the arc modification (j, i). Finally, a matrix $paths[i, j]$, $i, j = 1, 2 \ldots, n$, of dimension $n \times n$ that represents the number of paths between each pair of vertices (variables). This structure is used to check if an arc modification produces a DAG structure. For instance, it is possible to add the arc (j, i) to the structure if the number of paths between i and j is equal to 0, that is, $paths[i, j] = 0$.

A pseudocode for the sequential structure learning algorithm can be seen in Fig. 5.

```
EBNA_BIC algorithm. Sequential version.

Input: $D$, $S$, $paths$
Step 1. for $i = 1, \ldots, n$ calculate $BIC[i]$
Step 2. for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ $G[j, i] = 0$
Step 3. for $i = 1, \ldots, n$ and $j = 1, \ldots, n$
            if $(i \neq j)$ calculate $G[j, i]$ /* the change of the BIC
            produced by the arc modification $(j, i)$ */
Step 4. find $(j, i)$ such that $paths[i, j] = 0$ and $G[j, i] \geq G[r, s]$
        for each $r, s = 1, \ldots, n$ such that $paths[s, r] = 0$
Step 5. if $G[j, i] > 0$
            update $S$ with arc modification $(j, i)$
            update $paths$
        else stop
Step 6. for $k = 1, \ldots, n$
            if $(k \neq i$ or $k \neq j)$ calculate $G[k, i]$
Step 7. go to Step 4
```

Fig. 5. Pseudocode for the sequential structural learning algorithm $\mathrm{EBNA}_{\mathrm{BIC}}$.

2) Parallel Approach—First Version: This first version is a straightforward parallelization of the sequential algorithm. That is, the only changes made in the code are those necessary to implement the parallel version, maintaining the sequential functionality.

Parallelization is proposed only for the learning phase, thus, all the other phases of the algorithm are carried out sequentially by the manager (master). During this learning phase, the manager sends different orders to the workers identifying the work to be done. To begin, the manager sends some information to the workers (e.g., size of the individual D database, number of workers) and in the subsequent steps different orders are sent requesting the computation of the decomposed BIC score and maintenance of the local S structure. Workers need their own data structures (because are running in different machines) to store temporary results. Once initialization has been completed, workers wait for an order. Two different types of orders can be received: one, to compute the BIC score for their respective set of variables (as each worker has a unique identifier it is easy to divide the work that each worker must complete in terms of this identifier) and return the results to the manager; the second type of order is to maintain the integrity of the S local structure: each addition or deletion of edge (i, j) performed at one worker has to be notified to the remaining workers.

This is mainly the MPI schema but, as mentioned at the beginning of this part, in this implementation we have also added thread-programming techniques. When combined with MPI, a variable number of threads can be created at the beginning of the program execution and, when an order of BIC computation for a subset of variables arrives to a worker, it can distribute that workload among the preforked threads. Each of them will independently compute a BIC value for a variable, and then another one, until all variables have been processed. It can be seen that in every "MPI worker," new workers (threads) are locally created, all collaborating to complete the work assigned to the "MPI worker." As they use shared memory, all the changes are made in the same set of data structures, thus not involving MPI communication.

The pseudocode for this algorithm is shown in Fig. 6 (manager) and Fig. 7 (workers).

3) Parallel Approach—Second Version: The previous version of the parallel program strictly follows the structure of the sequential counterpart. All the computations are made for each $G[i, j]$, even though in the search for the best $G[r, s]$ value, it must be also verified that the Bayesian network is still a DAG. This means that workers perform some tasks that can be considered useless: why must they compute a BIC score if it does not fulfill the DAG property? This led us to a second version of the program that reduces this overhead.

The first idea is to send to the workers the data structure needed for them to check if the DAG property is still maintained ($paths$). However, this alternative is not efficient, because work has been distributed among the workers as $numvariables/numworkers$ but it is impossible to know how many BICs must be calculated at each worker (only those

```
EBNA_BIC Algorithm. First parallel version. Manager

Input: $D$, $S$, $paths$
Step 1.  send $D$ to the workers
         set the number of variables ($NSet$) to work with
Step 2.  send "calculate $BIC$" order to the workers
Step 3.  receive $BIC$ results from workers
Step 4.  for $i = 1, \ldots, n$ and $j = 1, \ldots, n$ $G[j,i] = 0$
Step 5.  for $i = 1, \ldots, n$
             send $i$, $BIC[i]$ to the workers
             send "calculate $G[k,i]$" order to the workers
Step 6.  receive from workers all the changes and update $G$
Step 7.  find $(j,i)$ such that $paths[i,j] = 0$ and $G[j,i] \geq G[r,s]$
         for each $r, s = 1, \ldots, n$ such that $paths[s,r] = 0$
Step 8.  if $G[j,i] > 0$
             update $S$ with arc modification $(j,i)$
             update $paths$
             send "change arc $(j,i)$ order" to the workers
         else send workers "stop order" and stop
Step 9.  send "calculate $G[k,i]$ for $(i,j)$" to the workers
Step 10. receive from workers all the changes and update $G$
Step 11. go to Step 7
```

Fig. 6. Pseudocode for the parallel structural learning phase. First version of the EBNA$_{\text{BIC}}$ algorithm for the manager.

```
EBNA_BIC algorithm. First parallel version. Workers

Step 1. create and initialize $S$ local structure
        receive $D$
        define the set of variables ($NSet$) to work with
Step 2. wait for an order
Step 3. case order of
            "calculate $BIC$"
                for each variable $i$ in $NSet$ calculate $BIC[i]$
                send $BIC$ results to the manager
            "calculate $G[k,i]$"
                for each variable $k$ in $NSet$
                    if $(k \neq i)$ calculate $G[k,i]$
                send $G$ modifications to the manager
            "calculate $G[k,i]$ for $(i,j)$"
                for each variable $k$ in $NSet$
                    calculate $G[k,i]$
                send $G$ modifications to the manager
            "change arc $(i,j)$"
                Update $S$ with $(i,j)$ arc modification
            "stop" stop
Step 4. go to Step 2
```

Fig. 7. Pseudocode for the parallel structural learning phase. First version of the EBNA$_{\text{BIC}}$ algorithm for the workers.

that kept the DAG property). Therefore, notable differences in computational effort could exist between different workers (unbalanced distribution) and the manager must wait until the last of them finishes.

For this reason, it is necessary to find out in advance the number of edges that fulfill the restriction and, therefore, before sending the work, the manager tests for all the possible changes that maintain the DAG condition, creating a set with those. Then, this set is sent to the workers, and each of them calculates the BIC criterion for its proportional subset, guaranteeing an equally distributed workload. Fig. 8 (manager) shows the changes made for this second version. Code for the workers is not shown due to its similarity with the first version: the single difference is that in this version the workers explicitly receive the set of variables to work with, instead of calculating the set themselves.

```
EBNA_BIC algorithm. Second parallel version. Manager

Step 5. for $i = 1, \ldots, n$
            send $i$, $BIC[i]$ to the workers
            calculate the structures that fit the DAG property
            set the number of variables ($NSetDAG$) to work with
            send "calculate $G[k,i]$" order to the workers
            send $NSetDAG$ to the workers
Step 9. calculate the structures that fit the DAG property
        set the number of variables ($NSetDAG$) to work with
        send "calculate $G[k,i]$ for $(i,j)$" order
        send $NSetDAG$ to the workers
```

Fig. 8. Pseudocode for the parallel structural learning phase. Second version of the EBNA$_{\text{BIC}}$ algorithm for the manager. Only the steps that differ from the previous version are shown.

B. EBNA$_{\text{PC}}$ *Algorithm [19], [20]*

1) Algorithm Description: Like the previous algorithm, EBNA$_{\text{PC}}$ also maintains the common schema described for EDAs, but, unlike EBNA$_{\text{BIC}}$, to complete the learning phase a "detection of conditional (in)dependencies" method is used.

This method tries to detect whether or not there are conditional dependencies between all the variables that constitute the domain. All the detected dependencies are stored and, in the final step, a Bayesian network is created based on the detected dependencies.

EBNA$_{\text{PC}}$ receives its name from the fact that the algorithm used to detect the conditional (in)dependencies is the PC algorithm, described in [32]. Like most recovery algorithms based on independence detections, the PC algorithm starts by creating the complete undirected graph G, then "thins" that graph by removing edges with zero-order conditional independence relations, "thins" again with first-order conditional relations, then with second-order conditional relations, and so on. The set of variables conditioned on need only be a subset of the set of variables adjacent to one of the variables of the pair. The independence test is performed based on the χ^2 distribution. When there are no more tests to do, the orientation process begins, giving a direction to each edge in G.

The data structures used in this program are: a matrix G to store the undirected graph structure, where $G[i,j]$ is *true* when there is an edge connecting i and j nodes; and a structure—*SepSet*—to save the independence relation between two variables and the set of variables conditioned on.

Fig. 9 shows the pseudocode for the PC algorithm. $Adj(G,A)$ represents the set of vertices adjacent to the vertex A in the undirected graph G, and $I(X_i, X_j | S(X_i, X_j))$ indicates that X_i and X_j are independent given a subset of adjacent variables $S(X_i, X_j)$. Note that the graph G is continually updated, so

```
EBNA_PC algorithm. Sequential version.

Step 1. form the complete undirected graph $G$ on
        vertex set $V = \{X_1, \ldots, X_n\}$
Step 2. $r = 0$
Step 3. repeat
          repeat
            select an ordered pair of variables $X_i$ and $X_j$ that are
            adjacent in $G$ such that $| Adj(G, X_i)\backslash\{X_j\} | \geq r$
            and a subset $S(X_i, X_j) \subseteq Adj(G, X_i)\backslash\{X_j\}$
            of cardinality $r$
            if $I(X_i, X_j \mid S(X_i, X_j))$
              delete the edge $X_i - X_j$ from $G$ and
              record $S(X_i, X_j)$ in $Sepset(X_i, X_j)$
              and $Sepset(X_j, X_i)$
          until all ordered pairs of adjacent variables $X_i$ and $X_j$ such
          that $| Adj(G, X_i)\backslash\{X_j\} | \geq r$ and all $S(X_i, X_j)$ of
          cardinality $r$ have been tested for $u$-separation
          $r := r + 1$
        until for each ordered pair of adjacent vertices $X_i, X_j$ we have
        $| Adj(G, X_i)\backslash\{X_j\} | < r$
Step 4. for each triplet of vertices $X_i, X_j, X_l$ such that
        the pair $X_i, X_j$ and the pair $X_j, X_l$ are both adjacent in $G$
        but the pair $X_i, X_l$ is not adjacent in $G$, orient $X_i - X_j - X_l$
        as $X_i \rightarrow X_j \leftarrow X_l$ if and only if $X_j$ is not in $Sepset(X_i, X_l)$
Step 5. repeat
          if $X_i \rightarrow X_j$, $X_j$ and $X_l$ are adjacent,
          $X_i$ and $X_l$ are not adjacent, and there is no arrowhead at $X_j$
            orient $X_j - X_l$ as $X_j \rightarrow X_l$
          if there is a directed path from $X_i$ to $X_j$
          and an edge between $X_i$ and $X_j$
            orient $X_i - X_j$ as $X_i \rightarrow X_j$
        until no more edges can be oriented
```

Fig. 9. Pseudocode for the sequential structural learning algorithm $\mathrm{EBNA_{PC}}$.

TABLE II
TIME MEASUREMENT (%) OF DIFFERENT PHASES OF THE $\mathrm{EBNA_{PC}}$ LEARNING PHASE

Individual size	*Detect dependencies*	*Orientation*
150	98.9	0.4
250	99.2	0.4
500	99.3	0.6

$Adj(G, A)$ is constantly changing as the algorithm progresses. A good review for the induction of Bayesian networks by detecting conditional (in)dependencies can be found in [33].

2) Parallel Approach—First Version: We performed a study of execution times of the sequential algorithm. As shown in Table I, the learning phase requires nearly 98% of total execution time, so this is the obvious target for parallelization. As mentioned earlier, the learning phase can be divided into two different steps: detection of conditional (in)dependencies, and orientation. Table II shows that the first step takes about 99% of the learning time, therefore, our parallel approach focuses only on this step.

Once we have explained the $\mathrm{EBNA_{PC}}$ algorithm, we propose a parallel implementation. The model induction begins with a complete undirected graph and, in the first step, all zero-order conditional independencies are searched. Each computational node can carry out this process independently. Thus, all the pairs are equally distributed between the manager and workers (since the manager must wait for the results it may as well be used as another worker). If n is the number of variables, the dependencies between $n(n-1)/2$ pairs must be checked. Thus, each worker (including the manager) take on a set of $(n(n-1)/2)/numworkers$ pairs to detect dependencies.

In the subsequent steps, the original algorithm tests sequentially the dependencies between all possible (X_i, X_j) pairs, $i = 1, \ldots, n$ and $j = 1, \ldots, n$, with all the subsets of cardinality r (from $r = 1$ until no subset can be found). The deleted edges and the state of the executions must be controlled, so the manager needs auxiliary structures where the current state is saved:

- *manage_pairs*: stores all the pairs that must be calculated along with each one's current situation: is being calculated or not, cardinality value and so on;
- *notify_changes*: stores for each worker a list with the edges that must be deleted from the G structure.

Note that the parallel version must repeat exactly the same steps as the sequential algorithm. Starting with zero-order dependencies, the pairs can be distributed between the manager and the workers without any additional control. When these computations have finished, the real "challenging" work begins. The manager waits for requests from workers asking for pairs to calculate. To determine whether a pair (X_i, X_j) is susceptible to be calculated, there cannot be another pair (X_k, X_l) being calculated such that $X_i = X_k$, $X_i = X_l$ or $X_j = X_k$. In this way, we assure that the order followed to distribute the work is exactly the same as the one used in the sequential algorithm. If there is no possibility of obtaining a pair, the manager forces the worker to wait until a pair can be selected. This process is repeated until no more dependencies can be found.

In these steps (from first order until the end), the manager will distribute the work to be done among the workers, sending the next pair to be calculated. As this calculation can change the G and *SepSet* structures, the manager receives the result of each worker's execution (must edge be removed?) and updates its G and *SepSet* structures when necessary, changing also the structure used to notify each worker of the changes made in the G graph—*notify_changes*. This is necessary because each worker needs to have an updated copy of the structure of the G graph before doing any conditional independence detection, and it is cheaper in terms of computation and communication to send the changes than to send the whole G structure each time. The *SepSet* structure is only needed in the manager because, as said earlier, the orientation process will be done sequentially.

The pseudocode of this parallel version of the $\mathrm{EBNA_{PC}}$ algorithm can be seen in Fig. 10 (manager) and Fig. 11 (workers).

3) Parallel Approach—Second Version: If we observe the operation mode of the parallel program when computing dependencies with adjacent sets of cardinality greater than zero, we can see that the manager selects the next pair to calculate, and then sends it to the worker that has asked for a job, repeating this process for all the requests. While workers process their assigned pairs, the manager stays idle, waiting. An evident improvement here is to make the manager play also the role of worker when there are not requests pending: it can take a new pair and calculate it itself.

In Fig. 12, the modifications made in the manager version of the parallel program can be seen. The worker's algorithm is not repeated again because it is the same of the first version of the $\mathrm{EBNA_{PC}}$.

$EBNA_{PC}$ **algorithm. Fist parallel version. Manager**

```
Step 1.  form the complete undirected graph $G$ on
         vertex set $V = \{X_1, \dots, X_n\}$
Step 2.  send $D$ structure to the workers
Step 3.  define the set of pairs $(X_i,X_j)$ that are adjacent in $G$
Step 4.  select a subset of pairs $(PairsSet)$ to work with
Step 5.  for each pair $(X_i,X_j)$ in $(PairsSet)$
           test conditional dependencies
           if "test fulfilled"
             delete $(i,j)$ edge
Step 6.  receive from workers the edges they have deleted and update $G$
Step 7.  send the updated $G$ to the workers
Step 8.  $r = 1$
Step 9.  repeat
           for each worker $w$ without work
             search for a pair to send
             if there is a pair $(X_i,X_j)$
               send "notify changes" order to the worker
               send edges deleted by other workers to the worker
               send $i,j$ values to the worker
             if it is not possible to send a pair (due to priority)
               do not send anything to worker
             if all pairs for all possible $r$ values have been calculated
               send "stop" order to the worker
           wait for an answer
           case request of
             "result"
               update the $manage\_pairs$ structure
               if "result=true"
                 delete from $G$ the edge that the worker $w$ has
                 calculated and update $notify\_changes$ structure
             "stop"
               take note that worker $w$ has finished its execution
         until all the workers have finished
Step 10. for each triplet of vertices $X_i, X_j, X_l$ such that
         the pair $X_i, X_j$ and the pair $X_j, X_l$ are both adjacent in $G$
         but the pair $X_i, X_l$ is not adjacent in G, orient $X_i - X_j - X_l$
         as $X_i \rightarrow X_j \leftarrow X_l$ if and only if $X_j$ is not in $Sepset(X_i, X_l)$
Step 11. repeat
           if $X_i \rightarrow X_j$, $X_j$ and $X_l$ are adjacent,
           $X_i$ and $X_l$ are not adjacent, and there is no arrowhead at $X_j$
             orient $X_j - X_l$ as $X_j \rightarrow X_l$
           if there is a directed path from $X_i$ to $X_j$
           and an edge between $X_i$ and $X_j$
             orient $X_i - X_j$ as $X_i \rightarrow X_j$
         until no more edges can be oriented
```

Fig. 10. Pseudocode for the parallel structural learning phase. First version of the EBNA$_{\text{PC}}$ algorithm for the manager.

$EBNA_{PC}$ **algorithm. First parallel version. Workers**

```
Step 1.  form the complete undirected graph $G$ on
         vertex set $V = \{X_1, \dots, X_n\}$
Step 2.  receive $D$ structure from manager
Step 3.  define the set of pairs $(X_i,X_j)$ that are adjacent in $G$
Step 4.  select a subset of pairs $(PairsSet_w)$ to work with
Step 5.  for each pair $(X_i,X_j)$ in $(PairsSet_w)$
           test conditional dependencies
           if "test fulfilled" save $(i,j)$ edge in $Deleted\_edges_w$
Step 6.  send to the manager edges in $Deleted\_edges_w$
Step 7.  receive $G$ from the manager
Step 8.  wait for an order
Step 9.  case order of
           "notify changes"
             receive the edges deleted by other workers and update $G$
             receive $i,j,r$ values and test conditional dependencies
             for the $(X_i,X_j)$ pair with all possible sets
             with cardinality $r$
             send result (deleted or not) to the manager
           "stop"
             send confirmation and stop
Step 10. go to Step 8
```

Fig. 11. Pseudocode for the parallel structural learning phase. First version of the EBNA$_{\text{PC}}$ algorithm for the workers.

$EBNA_{PC}$ **algorithm. Second parallel version. Manager**

```
Step 9.  repeat
           for each worker $w$ without work
             search for a pair to send
             if there is a pair $(X_i,X_j)$
               send "notify changes" order to the worker
               send edges deleted by other workers to the worker
               send $i,j$ values to the worker
             if it is not possible to send a pair (due to priority)
               do not send anything to worker
             if all pairs for all possible $r$ values have been calculated
               send "stop" order to the worker
           if there is an answer
             case request of
               "result"
                 update the $manage\_pairs$ structure
                 if "result=true"
                   delete from $G$ the edge that the worker $w$ has
                   calculated and update $notify\_changes$
               "stop"
                 take note that the worker $w$ has finished its execution
           else
             search for a pair
             if there is a pair $(X_i,X_j)$
               test conditional dependencies
               if "test fulfilled"
                 delete $(i,j)$ edge and update $notify\_changes$
         until all the workers have finished
```

Fig. 12. Pseudocode for the parallel structural learning phase. Second version of the EBNA$_{\text{PC}}$ algorithm for the manager. Only the steps that differ from the previous version are shown.

C. Experiments for the Discrete Domain

This section reports the different experiments that have been carried out using different implementations and target computer combinations.

The results of the experiments are presented from the point of view of speed up and scalability of the parallel algorithms. Note that the parallel versions have exactly the same behavior of the sequential algorithms but run faster, allowing them to solve harder problems.

With regard to the general steps of EDAs, there is one phase, evaluation of the individuals, that uses different fitness functions depending on the particular target problem. Therefore, the time needed to evaluate all the individuals can vary significantly from one problem to other.

Trying to give a general view of the parallel approaches, we have decided to select simple problems (with very simple fitness functions) in order to study the behavior of the parallel algorithms independently of the evaluation phase.

For the discrete domain the chosen problem is the well-known OneMax function, that has a very simple objective function. This problem consists of maximizing

$$\text{OneMax}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i$$

where $x_i \in \{0, 1\}$.

Clearly, OneMax is a linear function, which is easy to optimize. The computational cost of evaluating this function is tiny.

The machine used to carry out the experiments—which is the same for discrete and continuous domains, is a cluster of 10 nodes, where each node has two AMD ATHLON MP 2000+ processors (CPU frequency 1.6 GHz), with 256 KB of cache memory per processor, and 1 GB of RAM per node. The operating system is GNU-Linux and the MPI implementation is LAM (6.5.9. version). All the computers are interconnected using a switched Gigabit Ethernet network.

As we have said in advance, two different paradigms have been used to implement the parallel algorithms: MPI and Threads. Both have been used for the experiments.

- pure MPI: All communications (intranode and internodes) are performed using only MPI.
- MPI&Threads: A MPI process runs at each node. Communication and synchronization between nodes is done via MPI. Inside each node, the MPI process creates two threads (one per processor) that use shared variables to communicate and synchronize.

We have executed several experiments for each algorithm combining different number of nodes and using for each node both processors (MPI&Threads) or only one (pure MPI). This way we obtain efficiency figures for both a cluster of dual-processors (MPI&Threads) and a cluster of single-processor machines (pure MPI).

On the subject of the parameters chosen to test program performance, we have selected a medium–big size of Bayesian network: an individual size of 500 (for both algorithms, $\mathrm{EBNA_{BIC}}$ and $\mathrm{EBNA_{PC}}$). For the remaining parameters, these are the default values: size of the population is 2000; 1999 new individuals are created each new generation; from those, the best 1000 are selected. The stopping criterion is the completion of a fixed number of generations: 15 for both algorithms. The reason for this is that, due to the variable behavior of EDAs, execution time can vary from one execution to other if the algorithm is stopped when a particular result is obtained. Therefore, fixing the number of generations we get easily comparable execution times for all program versions. As the base algorithms are the same (the sequential versions), we cannot claim (in fact, we do not want to do so) any improvement in the quality of the results, just in the execution time.

Each experiment has been repeated 30 times. Results presented here are the average of the 30 measurements. These are very representative, as the observed deviation has been less than 2%.

In the following sections, results are presented from two different points of view: time-related dimension and performance-related dimension.

1) Time-Related Dimension: The main goal of a parallelization process is to reduce the computation time required for a program to complete its work. In the following tables, execution time and speed up, as well as efficiency are presented. These last two concepts are defined as the following:

- Speed up $=$ $sequential\ time/parallel\ time$;
- Efficiency $=$ $speed\ up/number\ of\ processors$.

Tables III and IV summarize the results of the experiments for the $\mathrm{EBNA_{BIC}}$ and $\mathrm{EBNA_{PC}}$ algorithms, respectively.

TABLE III
TIME-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $\mathrm{EBNA_{BIC}}$ PARALLEL ALGORITHM

First version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 25' 42"	-	-	-	-	-
2	1h 08' 44"	2.12	1.06	1h 08' 13"	2.14	1.07
6	24' 38"	5.91	0.99	24' 23"	5.97	1.00
10	15' 21"	9.49	0.95	15' 32"	9.38	0.94
20				08' 54"	16.39	0.82

Second version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 25' 42"	-	-	-	-	-
2	59' 19"	2.46	1.23	59' 23"	2.45	1.23
6	21' 24"	6.81	1.13	21' 17"	6.85	1.14
10	13' 54"	10.48	1.05	13' 42"	10.63	1.06
20				08' 02"	18.13	0.91

TABLE IV
TIME-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $\mathrm{EBNA_{PC}}$ PARALLEL ALGORITHM

First version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 29' 40"	-	-	-	-	-
2	2h 07' 35"	1.17	0.59	2h 07' 35"	1.17	0.59
6	34' 10"	4.38	0.73	34' 23"	4.35	0.73
10	23' 48"	6.29	0.63	24' 05"	6.22	0.62
20				18' 33"	8.07	0.40

Second version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 29' 40"	-	-	-	-	-
2	58' 25"	2.56	1.28	58' 30"	2.56	1.28
6	28' 10"	5.31	0.89	28' 11"	5.31	0.88
10	22' 27"	6.66	0.67	22' 28"	6.66	0.67
20				18' 16"	8.19	0.41

It can be observed that similar results are obtained using pure MPI and MPI&Threads, a demonstration of parallel algorithms being successfully executed over clusters of single-processors, as well as over clusters of multiprocessors. In particular, it must be pointed out the excellent efficiency of both versions of the $\mathrm{EBNA_{BIC}}$ algorithm, even when using 20 CPUs. As we expected, the second parallel version reaches shorter computation times because only the scores actually needed are calculated. It could be surprising to observe efficiency reaching values over 1, but it must be taken into account that in the parallel executions more computers are used, meaning that more data fits into ultrafast cache memories.

With regard to the $\mathrm{EBNA_{PC}}$ algorithm, promising results are obtained using few CPUs, but when ten or more are used the parallel algorithm results quite inefficient. It must be pointed out that a lot of communication/synchronization is required in this algorithm, being necessary a continuous workload distribution, as well as an update of common data structures. So, the more workers are used, the more time is needed for communication nullifying the addition of more CPU power.

There are significant differences between the two parallel versions when using a small number of CPUs. This is because

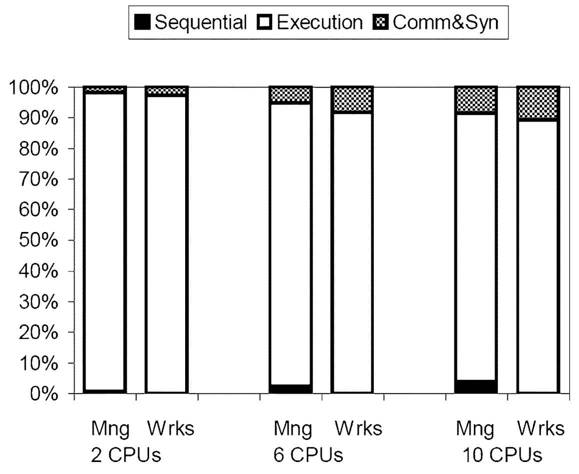


Fig. 13. Detail of the computation time for the first version of the $\mathrm{EBNA}_{\mathrm{BIC}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

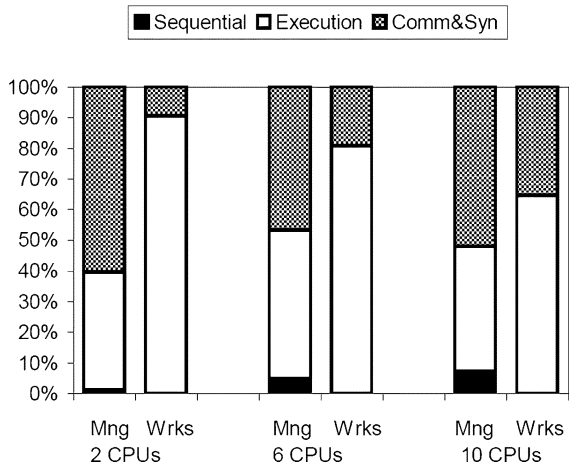


Fig. 15. Detail of the computation time for the first version of the $\mathrm{EBNA}_{\mathrm{PC}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

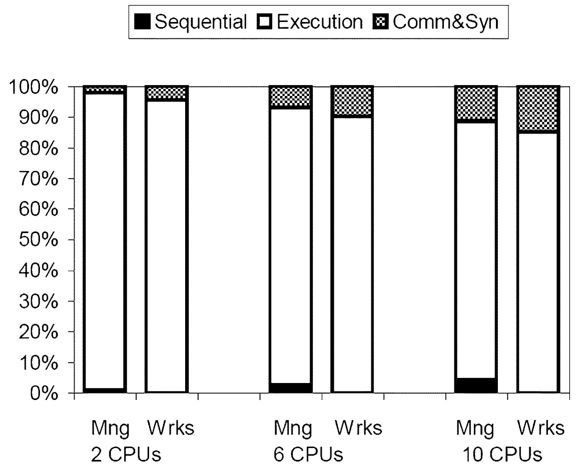


Fig. 14. Detail of the computation time for the second version of the $\mathrm{EBNA}_{\mathrm{BIC}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

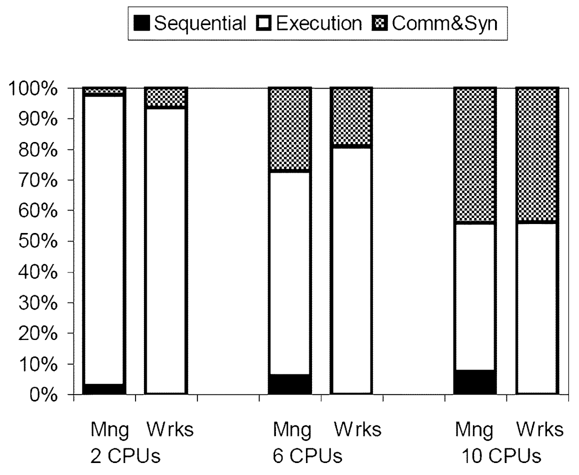


Fig. 16. Detail of the computation time for the second version of the $\mathrm{EBNA}_{\mathrm{PC}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

in the first version the manager only receives the results from the workers and sends them new work. Therefore, when few workers are available, the manager spends a significant portion of its time waiting for them to finish. In the second version, the manager makes use of this idle time to act as a worker. Logically, as the number of CPUs increases, both algorithms obtain similar efficiencies.

2) Performance-Related Dimension: When a parallel algorithm is proposed, the computational workload must be distributed correctly between all the nodes that work in parallel. Otherwise, the program is not scalable because most loaded nodes become bottlenecks.

In Figs. 13–16, total execution time has been split in parts to better observe program (master and workers) behavior. Legend "Sequential" shows the portion of time required by those parts of the code that have not been parallelized. The remaining legends differentiate time spent in computation ("Execution") and time spent in communication and synchronization ("Comm&Syn") for manager and workers. Tables V and VI extend this information, presenting actual execution times, with deviations for the case of workers. Some interesting conclusions can be presented for the $\mathrm{EBNA}_{\mathrm{BIC}}$ algorithm inspecting obtained information.

- Time required by the sequential steps is quite low, allowing a good scalability. The same occurs with the time needed for communication and synchronization between the manager and the workers.
- Work distribution can not be always exact $(numscores/numworkers)$, and this generates a slight variation in the deviation observed among the workers for the different CPU combinations.
- In the second version, the communication/synchronization time increases with the number of CPUs. This is because the manager completes an extra work, checking for those scores that must be computed and indicating to each worker which of them to compute. Therefore, there can be more idle times for the workers, while waiting for new orders.

On the other hand, for the $\mathrm{EBNA}_{\mathrm{PC}}$ we observe that:

- Time required by the sequential steps is proportionally larger, but still allows a reasonable efficiency.

- Communication and synchronization requirements make this algorithm quite inefficient as the number of CPUs increases. In the first version, as manager only distributes the work, when using few CPUs it spends a substantial time waiting for the workers requesting additional pieces of work. However, when using too many workers (10 or more) manager is not able to attend workers fast enough, so workers waste time just waiting for more to do.
- In the second version, the manager acts also as a worker making use of idle times and therefore the communication/synchronization time for the manager reduces drastically. This works fine for small numbers of workers but, as occurs with the first version, when this number is larger blocking situations arise at the worker side.

V. Continuous Domain

A. EGNA$_{\mathrm{BIC}}$ *Algorithm [4]*

1) Algorithm Description: As we have already noted, EDA algorithms can be used in discrete and also in continuous domains. In this section, we make proposals for the parallelization of the EGNA$_{\mathrm{BIC}}$ algorithm, which is similar to discrete EBNA$_{\mathrm{BIC}}$ but modified for their use in continuous domains. A Gaussian network is created instead of a Bayesian network, using a "score + search" approach. That is, the main idea under this approach consists of having a measure for each candidate Gaussian network in combination with a smart search through the space of possible structures. All the comments made concerning the learning algorithm in EBNA$_{\mathrm{BIC}}$ are also valid for this algorithm.

The score used to verify the performance of the obtained Gaussian network is the BIC. A general formulation of this criterion follows:

$$\mathrm{BIC}(S,D) = \sum_{r=1}^{N}\sum_{i=1}^{n}\left[-\frac{1}{2}\ln(2\pi v_i) - \frac{1}{2v_i} \times \left(x_{ir} - m_i - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j)\right)^2\right] - \frac{1}{2}\log(N)\left(2n + \sum_{i=1}^{n}|\boldsymbol{Pa}_i|\right) \qquad (8)$$

where

- n is the number of variables in the Gaussian network;
- N is the number of selected individuals;
- v_i is the conditional variance for the variable X_i given $\boldsymbol{Pa_i}$;
- m_i is the mean of the variable X_i;
- b_{ji} is the regression coefficient for variable X_j in $\boldsymbol{Pa_i}$.

Like for EBNA$_{\mathrm{BIC}}$ algorithm, this score can also be broken down to separately calculate the score for each variable. Accordingly, each variable X_i has associated with it a local BIC score $(\mathrm{BIC}(i,S,D))$

$$\mathrm{BIC}(S,D) = \sum_{i=1}^{n}\mathrm{BIC}(i,S,D) \qquad (9)$$

TABLE V
Algorithm Performance-Related Experimental Results for the EBNA$_{\mathrm{BIC}}$ Algorithm. Execution Times for Manager and Workers

First version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	35"	1h 06' 50"	1' 19"	1h 06' 15"	1' 54"
6	35"	22' 46"	1' 17"	22' 04"±07"	1' 59"
10	35"	13' 28"	1' 18"	13' 11"±01"	1' 35"

Second version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	35"	57' 33"	1' 11"	56' 10"	2' 34"
6	35"	19' 21"	1' 28"	18' 48"±07"	2' 01"
10	35"	11' 45"	1' 34"	11' 21"±05"	1' 58"

TABLE VI
Algorithm Performance-Related Experimental Results for the EBNA$_{\mathrm{PC}}$ Algorithm. Execution Times for Manager and Workers

First version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	1' 40"	48' 30"	1h 18' 14"	1h 54' 31"	12' 13"
6	1' 40"	16' 10"	15' 57"	27' 14"±26"	15' 57"
10	1' 40"	9' 42"	12' 21"	15' 18"±38"	8' 25"

Second version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	1' 40"	55' 29"	1' 17"	53' 08"	3' 38"
6	1' 40"	18' 49"	7' 40"	21' 25"±28"	5' 04"
10	1' 40"	10' 51"	9' 54"	13' 54"±36"	10' 51"

where

$$\mathrm{BIC}(i,S,D) = \sum_{r=1}^{N}\left[-\frac{1}{2}\ln(2\pi v_i) - \frac{1}{2v_i} \times \left(x_{ir} - mi - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j)\right)^2\right] - \frac{1}{2}\log(N)\left(2 + |\boldsymbol{Pa}_i|\right). \qquad (10)$$

Consequently, the steps followed to parallelize this algorithm are like those for EBNA$_{\mathrm{BIC}}$, decomposing the BIC criterion and sending each piece of the score to a different worker.

2) Parallel Approach: As parallel approaches, we maintain the two proposals made for EBNA$_{\mathrm{BIC}}$. The first one, where workers calculate all possible arc modifications without considering whether the DAG property is fulfilled; and the second, where the manager performs a preliminary step to obtain the arc changes that maintain the DAG property, sending afterwards to each worker a subset of those possible modifications.

Due to the similarity of this algorithm to the discrete one, (EBNA$_{\mathrm{BIC}}$) we consider unnecessary to explain the entire process again. To obtain a complete view of the characteristics of the algorithm and the parallel solution, see Section IV-A.

B. EGNA$_{\mathrm{EE}}$ *Algorithm [17], [18]*

1) Algorithm Description: In this algorithm, the structure learning of the Gaussian network follows a "detecting conditional (in)dependencies" method. In particular, this method begins with a complete graph, where there is a connection from each variable X_i, $i = 1, \ldots, n$ to each variable X_j, $j = i + 1, \ldots, n$, and then a statistical test is completed for each edge, deleting the edge if the null hypothesis is fulfilled.

To complete this test, the likelihood ratio test is used–borrowed from [34]. The statistic to exclude the edge between X_1 and X_2 from a graphical Gaussian model is $T_{lik} = -nlog(1 - r_{ij|rest}^2)$, where $r_{ij|rest}$ is the sample partial correlation of X_i and X_j adjusted for the remaining variables. The latter can be expressed [35] in terms of the maximum-likelihood estimates of the elements of the precision matrix as $r_{ij|rest} = \hat{w}_{ij}(\hat{w}_{ii}\hat{w}_{jj})^{-1/2}$, where the precision matrix W is the inverse of the covariance matrix Σ.

[34] obtain the density and distribution functions of the likelihood ratio test statistic under the null hypothesis. These expressions are of the form

$$f_{lik}(t) = g_{\chi}(t) + \frac{1}{4}(t-1)(2n+1)g_{\chi}(t)N^{-1} + O(N^{-2}) \quad (11)$$

$$F_{lik}(x) = G_{\chi}(x) - \frac{1}{2}(2n+1)xg_{\chi}(x)N^{-1} + O(N^{-2}) \quad (12)$$

where $g_{\chi}(t)$ and $G_{\chi}(x)$ are the density and distribution functions, respectively, of a χ_1^2 variable.

Once the structure has been obtained, its parameters have to be learnt to complete the Gaussian network. This can be done using the following formulas:

$$\hat{m}_i = \overline{X}_i \quad (13)$$

$$\hat{b}_{ji} = \frac{\hat{\sigma}_{ji}}{\hat{\sigma}_{jj}^2} \quad (14)$$

$$\hat{v}_i = \hat{\sigma}_{ii}^2 - \sum_{X_j \in \boldsymbol{Pa}_i} \frac{\hat{\sigma}_{ji}}{\hat{\sigma}_{jj}^2} + 2 \sum_{X_j \in \boldsymbol{Pa}_i} \sum_{X_k \in \boldsymbol{Pa}_{i_{k>j}}} \frac{\hat{\sigma}_{jk}\hat{\sigma}_{ji}\hat{\sigma}_{ki}}{\hat{\sigma}_{jj}^2 \hat{\sigma}_{kk}^2} \quad (15)$$

where m is the estimated means, v the estimated conditional variances, and b the estimated regression coefficients.

Six major structures are needed to implement this algorithm. The graph structure is stored as $S[i]$, $i = 1, \ldots, n$, where for each node its parents are represented in an adjacency list. Two $n \times n$ structures Σ and W where the covariances matrix and its inverse are stored, respectively. Two vectors, $m[i]$, $i = 1, \ldots, n$, and $v[i]$, $i = 1, \ldots, n$ for means and conditional variances. Finally, an $n \times n$ matrix, $b[i, j]$, $i = 1, \ldots, n$, and $j = 1, \ldots, n$, where the regression coefficients will be saved.

A pseudocode for the sequential structure learning algorithm is presented in Fig. 17.

2) Parallel Approach—First Version: Table I (presented in Section III) tells us that the learning phase is not as time-consuming as in previous algorithms. Due to this, a deeper analysis has been done to know how the computation time is distributed among the different phases. Table VII shows the most important procedures that are executed by the present algorithm: learning and "sampling and evaluation" of new individuals.

As a first parallel approach, the parallelization of the learning phase is presented. Due to the execution time distribution, this version will be completed—in the second approach—with the parallelization of the "sampling and evaluation" phase.

```
$EGNA_{EE}$ algorithm. Sequential version.

Input: $D$
Step 1. calculate the means $m$ and the covariance matrix $\Sigma$
Step 2. initialize the structure of the Gaussian network $S$
Step 3. calculate the inverse of $\Sigma$
Step 4. for $i = 1, \ldots, n$
            for $j = i + 1, \ldots, n$
                calculate $r_{ij|rest}$
                if "test fulfilled"
                    update $S$ with $(i, j)$ arc deletion
Step 5. calculate the parameters $m$, $b$ and $v$
```

Fig. 17. Pseudocode for the sequential learning algorithm $\mathrm{EGNA}_{\mathrm{EE}}$.

TABLE VII
TIME MEASUREMENT (%) OF DIFFERENT PHASES OF THE $\mathrm{EGNA}_{\mathrm{EE}}$ ALGORITHM

Individual size	*Learning*	*Sampling and evaluation*
500	51.3	48.2
1,000	58.8	40.6
1,500	69.6	30.0

TABLE VIII
TIME MEASUREMENT (%) OF DIFFERENT PHASES OF THE $\mathrm{EGNA}_{\mathrm{EE}}$ LEARNING PHASE

Individual size	Σ	*Precision matrix*	*Test*	*Parameter learning*
500	59.2	12.5	0.3	27.9
1,000	46.2	9.7	0.1	44.0
1,500	29.6	6.1	0.1	64.2

The learning phase has several independent procedures, and we can descend one level in depth to obtain the exact computation time spent on each of these processes. As explained for the sequential algorithm, the learning phase completes the following steps.

Step 1) Calculate means and Σ from the selected individuals.
Step 2) Obtain the precision matrix.
Step 3) Compute the tests.
Step 4) Carry out the parameter learning.

Table VIII shows the time consumed by each of these processes. Different measures have been obtained taking into account different individual sizes and, although the percentages vary depending on the size of the individual, it can be observed that the most computationally expensive steps are the first (where means and Σ are calculated) and the fourth (where the parameters are learnt). In the second step—calculate the inverse of the covariance matrix—the LU decomposition method is used and execution is very fast. The third step (test) is also rapidly executed because it uses the values computed in the previous phases to obtain the r_{ij} values. Therefore, our parallelizing efforts focus on Steps 1) and 4).

To calculate Σ, it is first necessary to obtain the means for each variable, so the database of cases is sent to each worker and the number of means to be calculated (number of variables) is distributed among all the workers (manager included). If n is the number of variables and $numworkers$ the number of workers, each worker will have $n/numworkers$ variables to compute.

```
$EGNA_{EE}$ algorithm. First parallel version. Manager

Input: $D$
Step 1.  initialize the structure of the Gaussian network $S$
Step 2.  send $D$ structure to the workers
         set the number of variables ($NSet$) to work with
Step 3.  send "calculate means" order to the workers
Step 4.  Synchronize with all workers and update $m$
Step 5.  send "calculate covariances" order to the workers
Step 6.  Synchronize with all workers and update $\Sigma$
Step 7.  calculate the inverse of $\Sigma$
Step 8.  for $i = 1, \ldots, n$
             for $j = i + 1, \ldots, n$
                 calculate $r_{ij|rest}$
                 if "test fulfilled"
                     update $S$ with $(i, j)$ arc deletion
Step 9.  send "calculate parameters" order to the workers
Step 10. send $S$ to the workers
Step 11. receive from the workers all the changes and update $b$ and $v$
Step 12. send "stop" order to the workers
```

Fig. 18. Pseudocode for the parallel learning phase. First version of the $\mathrm{EGNA}_{\mathrm{EE}}$ algorithm for the manager.

```
$EGNA_{EE}$ algorithm. First parallel version. Workers

Step 1. receive $D$ structure from the manager
        set the number of variables ($NSet$) to work with
Step 2. wait for an order
Step 3. case order of
            "calculate means"
                for each variable $i$ in $NSet$
                    calculate the means $m_i$
                Synchronize with manager and workers and update $m$
            "calculate covariances"
                set the number of pairs ($NSet_p$) to work with
                for each pair $(i, j)$ in $NSet_p$
                    calculate the covariance $\Sigma[i, j]$
                Synchronize with manager and workers and update $\Sigma$
                learn distribution
            "calculate parameters"
                receive $S$ from the manager
                set the number of variables ($NSet_l$) to work with
                for each variable $i$ in $NSet_l$
                    calculate parameters $b_i$ and $v_i$
                send all $b$ and $v$ modifications to the manager
            "stop"
                stop
Step 4. go to Step 2
```

Fig. 19. Pseudocode for the parallel learning phase. First version of the $\mathrm{EGNA}_{\mathrm{EE}}$ algorithm for the workers.

Once the means have been calculated, all the nodes must send their results and receive the ones computed by the others. Thus, all the nodes have an updated *means* structure, which is needed for the next step: compute Σ matrix. The idea is the same, where each worker computes $(n(n-1)/2)/numworkers$ values of the matrix. Next, each worker sends its results to the manager and it updates the Σ structure, continuing with the sequential processes: calculate the precision matrix (inverse of Σ), and compute the tests.

Finally, the parameters must be learnt and, again, a workload distribution must be done. It this case, the distribution is not as simple as in the previous steps. Due to the initial structure of the Gaussian network, the variables with larger index have more dependencies and, therefore, need more time to compute its respective $\boldsymbol{b}$ and $\boldsymbol{v}$ values. To solve this, a simple algorithm has been used: a distribution of the variables in $numworkers$ sets, trying to balance global execution times.

Figs. 18 and 19 show the pseudocode for the parallel version.

3) Parallel Approach—Second Version: In the previous algorithms, we observed that the most time-consuming part of the programs was the learning phase, and we focused our efforts on it. However, for $\mathrm{EGNA}_{\mathrm{EE}}$, the "sampling and evaluation" phase also consumes a significant portion of time, and it is compulsory to work on it if we want an efficient and scalable parallel program. In fact, this situation can be observed in all those cases when solving problems that use a complex fitness function to evaluate individuals or require a large population and, therefore, this idea can be adapted to any of the algorithms presented in this paper.

Following the general structure of EDAs, an initial population is created, the best N individuals are selected, a probabilistic model is induced (learning phase) and, finally, a new population is generated based on the induced model. For this last step, an adaptation of the probabilistic logic sampling (PLS) proposed in [36] is used. In this method, the instances are generated one variable at a time in a forward way. That is, a variable is sampled after all its parents have already been sampled. To do that an ancestral ordering of the variables is given $(\pi(1), \ldots, \pi(n))$, where parent variables are before children variables. Once the values of $\mathbf{Pa}_{\pi(i)}$ have been assigned, we simulate a value for $X_{\pi(i)}$, using the distribution $f(x_{\pi(i)}|\boldsymbol{pa}_{\pi(i)})$. For the simulation of a univariate normal distribution, a simple method based on the sum of 12 uniform variables is applied [37].

```
$EGNA_{EE}$ sampling algorithm. Sequential version.

Input: $S$, $Order(\pi(1), \ldots, \pi(n))$, $m$, $b$, $v$
Step 1. for each $i$ in $NewPopSet$
            calculate a new individual
            add the individual to the population
```

Fig. 20. Pseudocode for the sequential sampling algorithm $\mathrm{EGNA}_{\mathrm{EE}}$.

A pseudocode for this sampling process can be seen in Fig. 20.

The parallelization approach is as follows: if $NewPopSet$ is the amount of new individuals to be created and $numworkers$ is the number of workers, the manager sends to each worker the order to create $NewPopSet/numworkers$ new individuals (the manager also creates new individuals). Once all these new individuals have been created, they are evaluated (taking advantage of the parallelism), and finally returned to the manager, which adds them to the population and continues with the algorithm.

In Fig. 21 (manager) and Fig. 22 (workers), the parallel proposal for the sampling phase can be seen.

C. Experiments for the Continuous Domain

The scenario used for the continuous domain is the same used for the discrete one, so detailed information can be obtained in Section IV-C.

```
EGNA_EE sampling algorithm. Parallel version. Manager

Input: $S$,$Order(\pi(1),\ldots,\pi(n))$,$m$,$b$,$v$
Step 1.  send $S$, $m$, $b$ and $v$ structures to the workers
Step 2.  receive the new individuals from the workers
         update the population
```

Fig. 21. Pseudocode for the parallel sampling phase. $\mathrm{EGNA_{EE}}$ algorithm for the manager.

```
EGNA_EE sampling algorithm. Parallel version. Workers

Step 1.  receive $S$, $m$, $b$ and $v$ structures
Step 2.  for each $i$ in $NewPopSet$
             calculate a new individual
Step 3.  send the new individuals to the manager
```

Fig. 22. Pseudocode for the parallel sampling phase. $\mathrm{EGNA_{EE}}$ algorithm for the workers.

As for the discrete domain, we have chosen a very simple, well-known problem: the *Sphere model*. With problems like this, the evaluation of the individuals (fitness function) does not require a large portion of computation time and, therefore, the efficiency of the parallel algorithm can be measured without taking into account the execution time of this function. However, it is important to note that the conclusions might change if more complex functions were used. In these cases, in order to improve the general performance of the algorithm, it would be necessary to parallelize also the evaluation of the individuals (we have followed this approach in the next algorithm $\mathrm{EGNA_{EE}}$).

The *Sphere model* is a simple minimization problem. It is defined so that $-600 \leq x_i \leq 600 \; i = 1, \ldots, n$, and the fitness value for an individual is as follows:

$$\mathrm{Sphere}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2. \tag{16}$$

As the reader can see, the fittest individual is the one whose components are all 0, which corresponds to the fitness value 0.

Regarding the individual and population sizes, different values have been selected for each algorithm, such that the execution times of the sequential algorithms are large enough to take into consideration their parallel execution.

- $\mathrm{EGNA_{BIC}}$: The size of the individual is 100. The population size is 2000, 1999 new individuals are created in each generation, and the best 1000 are selected. Execution is stopped when the tenth generation is reached.
- $\mathrm{EGNA_{EE}}$: 1500 has been selected as individual size, the population has 6000 individuals, and 5999 new ones being created in each generation. Then, the best 3000 are selected. Execution is stopped when the 15th generation is reached.

1) Time-Related Dimension: In Tables IX and X, the execution times, speed up, and efficiency are presented.

Results on $\mathrm{EGNA_{BIC}}$ show good levels of speed up. Scalability maintains an acceptable level even when 20 CPUs are used. However, compared with the discrete version, as evaluating the BIC score requires more time, the fact that work divisions are not exactly identical do reflect in an unbalanced workload distribution: more waiting times in manager and workers, and larger deviations.

TABLE IX
TIME-RELATED EXPERIMENTAL RESULTS FOR THE TWO $\mathrm{EGNA_{BIC}}$ PARALLEL VERSIONS

First version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 57' 47"	-	-	-	-	-
2	1h 48' 24"	1.64	0.82	1h 52' 03"	1.59	0.79
6	39' 01"	4.56	0.76	40' 21"	4.41	0.73
10	24' 12"	7.35	0.73	25' 09"	7.07	0.71
20				14' 07"	12.59	0.63

Second version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 57' 47"	-	-	-	-	-
2	1h 20' 13"	2.22	1.11	1h 21' 56"	2.17	1.08
6	30' 06"	5.91	0.98	30' 12"	5.89	0.98
10	20' 37"	8.62	0.86	19' 56"	8.92	0.89
20				12' 54"	13.78	0.69

TABLE X
TIME-RELATED EXPERIMENTAL RESULTS FOR THE TWO $\mathrm{EGNA_{EE}}$ PARALLEL VERSIONS

First version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	3h 23' 17"	-	-	-	-	-
2	2h 17' 38"	1.48	0.74	2h 18' 00"	1.47	0.74
6	1h 36' 35"	2.10	0.35	1h 35' 38"	2.13	0.35
10	1h 27' 17"	2.33	0.23	1h 26' 36"	2.35	0.23
20				1h 20' 40"	2.52	0.13

Second version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	3h 23' 17"	-	-	-	-	-
2	1h 47' 08"	1.90	0.95	1h 50' 08"	1.85	0.92
6	47' 56"	4.24	0.71	47' 26"	4.29	0.71
10	36' 29"	5.57	0.56	35' 43"	5.69	0.57
20				27' 20"	7.44	0.37

For the first version of $\mathrm{EGNA_{EE}}$, it can be observed that parallelizing only the learning phase is not enough to obtain an efficient parallel algorithm. As the "sampling and evaluation" phase requires around the 30% of the total execution time, when six or more CPUs are used, this phase becomes an obvious bottleneck. In the second version, the mentioned "sampling and evaluation" phase is also parallelized, and the efficiency improves noticeably.

2) Performance-Related Dimension: Performance results are summarized in Figs. 23–26 and Tables XI and XII. They show detailed information about execution times, exposing the percentage that each different section requires, as well as absolute timing values and deviations for workers.

As can be observed, the behavior of the $\mathrm{EGNA_{BIC}}$ algorithm is similar to $\mathrm{EBNA_{BIC}}$ algorithm and, therefore, conclusions are the same. This conclusions can be consulted in Section IV-C2.

For both versions of $\mathrm{EGNA_{EE}}$ algorithm, we could reach to these conclusions.

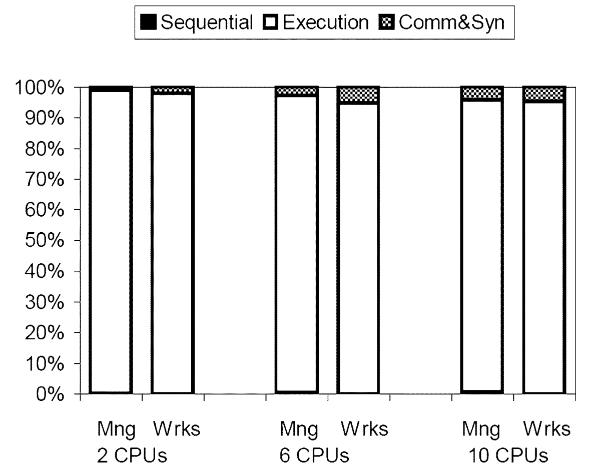


Fig. 23. Detail of the computation time for the first version of the $\mathrm{EGNA_{BIC}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

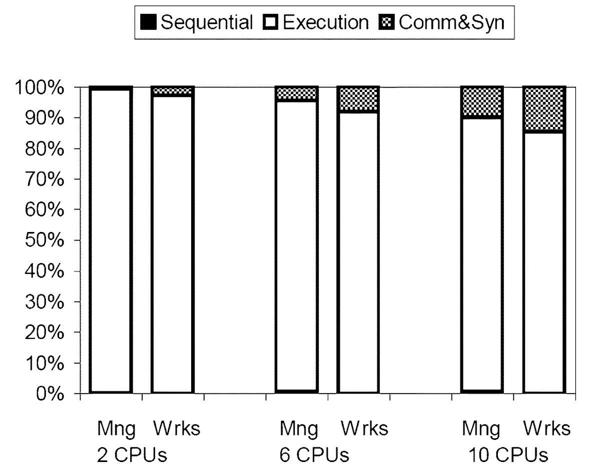


Fig. 24. Detail of the computation time for the second version of the $\mathrm{EGNA_{BIC}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

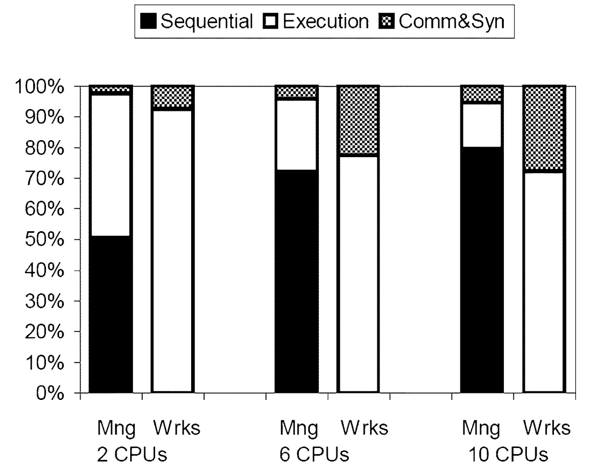


Fig. 25. Detail of the computation time for the first version of the $\mathrm{EGNA_{EE}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

- In the first version, where the "sampling and evaluation" phase has not been parallelized, the efficiency decreases notably when more CPUs are used, as it could be expected from the distribution of execution times for the different phases of the sequential algorithm. This is the reason why the second version (where this phase has been parallelized) performs better.
- In general, the scalability of the algorithm is quite poor. This is due to the high communication requirements—different calculations that require continuous synchronization—and to the portion of the algorithm that has not been parallelized (the sequential part), which becomes more prominent when the number of processors grows.

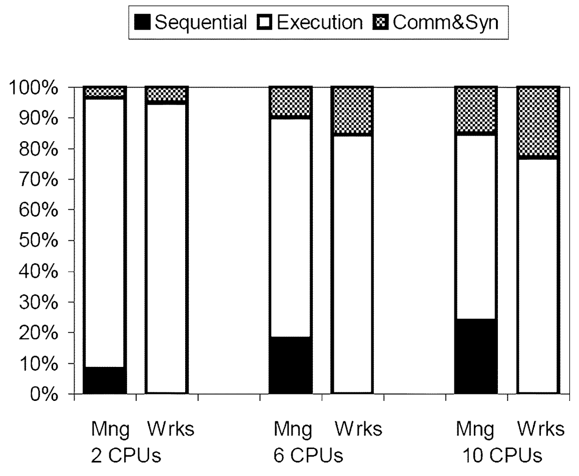


Fig. 26. Detail of the computation time for the second version of the $\mathrm{EGNA_{EE}}$ algorithm, using a pure MPI implementation—2, 6, and 10 CPUs have been used.

TABLE XI
ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR THE $\mathrm{EGNA_{BIC}}$ ALGORITHM. EXECUTION TIMES FOR MANAGER AND WORKERS

First version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	11"	1h 47' 10"	1' 03"	1h 46' 07"	2' 06"
6	11"	37' 42"	1' 08"	36' 47"±62"	2' 03"
10	11"	22' 58"	1' 03"	22' 53"±10"	1' 08"

Second version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	11"	1h 19' 30"	32"	1h 17' 49"	2' 13"
6	11"	28' 33"	1' 22"	27' 29"±35"	2' 26"
10	11"	18' 24"	2' 02"	17' 28"±35"	2' 58"

TABLE XII
ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR THE $\mathrm{EGNA_{EE}}$ ALGORITHM. EXECUTION TIMES FOR MANAGER AND WORKERS

First version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	1h 09' 28"	1h 04' 47"	3' 23"	1h 03' 06"	5' 04"
6	1h 09' 28"	23' 03"	4' 04"	21' 00"±16"	6' 07"
10	1h 09' 28"	13' 06"	4' 43"	12' 52"±10"	4' 57"

Second version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	8' 42"	1h 45' 38"	7' 14"	1h 32' 08"	20' 44"
6	8' 42"	37' 50"	4' 50"	32' 54"±17"	9' 46"
10	8' 42"	23' 40"	5' 31"	21' 18"±05"	7' 53"

VI. Conclusion and Future Work

In this paper, several parallel solutions have been proposed for different EDAs. For each one, a previous study of the execution times of the procedures that conform the sequential algorithm was made. In most of the cases, the step that requires the most substantial part of the total execution time is the learning phase, where a probabilistic graphical model is induced from the database of individuals. In the final section, we have seen that, depending on the algorithm, there may be other steps that also take up much computational time: for example, the sampling and creation of new individuals once the structure has been learnt.

The experiments have been made using a cluster of ten dual-processor computers, changing the number of nodes from two to ten (MPI communication). Two different parallel implementations have been presented (pure MPI and MPI&Threads), using one or two CPUs per node, showing in this way the ability of the algorithms to run in different target computers.

Looking at the obtained results, it can be seen that parallelization of the learning phase notably improves the performance of the algorithms. This suggests that applying parallelization techniques to EDAs to solve complex problems can bring them nearer to practical use. However, it is important to realize that for problems with complex fitness functions (evaluation of the individuals), it could be necessary to parallelize also the "sampling and evaluation" phase.

In terms of future work, we want to carry out these tasks.

- Improvements of the presented algorithms, taking advantage of the knowledge acquired during this work. For example, for the $\mathrm{EBNA}_{\mathrm{PC}}$ algorithm, a workload distribution using independent sets of pairs could be applied avoiding sending/receiving operations for each pair. Additionally, for the second version of the $\mathrm{EGNA}_{\mathrm{EE}}$ algorithm, we are looking for methods to reduce the sequential part, using parallel libraries to solve some large matrix-related computations.
- Complete a deeper study over the possible advantages of using multiple threads per node (intranode communication) versus using a pure MPI implementation.
- Analyze the viability of this parallel algorithms when used to solve more complex problems. Particularly, we plan to apply $\mathrm{EBNA}_{\mathrm{BIC}}$ to feature subset selection (FSS) problems, parallelizing also the "sampling and evaluation" phase of this algorithm following the schema presented for $\mathrm{EGNA}_{\mathrm{EE}}$ algorithm.
- Make use of the ideas and conclusions obtained from this work to parallelize other algorithms in the family of EDAs.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments on the earlier version of this paper.

References

[1] E. Cantù-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA: Kluwer, 2000.

[2] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 443–462, Oct. 2002.

[3] H. Mühlenbein and G. Paaß, "From recombination Of genes to the estimation of distributions i. binary parameters," in *Lecture Notes in Computer Science*, 1996, vol. 1411, Proc. Parallel Problem Solving from Nature—PPSN IV, pp. 178–187.

[4] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer, 2002.

[5] J. Lozano, R. Sagarna, and P. Larrañaga, "Parallel estimation of distribution algorithms," in *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, P. Larrañaga and J. A. Lozano, Eds: Kluwer Academic Publishers, 2002, pp. 129–145.

[6] J. Ocenasek and J. Schwarz, "The parallel Bayesian optimization algorithm," in *Proc. Eur. Symp. Comput. Intell.*, 2000, pp. 61–67.

[7] ——, "The distributed bayesian optimization algorithm for combinatorial optimization," *EUROGEN—Evol. Methods r Design, Optimization and Control, CIMNE*, pp. 115–120, 2001.

[8] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf.*, vol. I, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., Orlando, FL, 1999, pp. 525–532.

[9] J. Ocenasek, J. Schwarz, and M. Pelikan, "Design of multithreaded estimation of distribution algorithms," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2003, vol. 2724, Proc. Genetic Evol. Comput., pp. 1247–1258.

[10] C. W. Ahn, D. E. Goldberg, and R. Ramakrishna, "Multiple-deme parallel estimation of distribution algorithms: Basic framework and application," in *Lecture Notes in Computer Science*: Springer, 2004, vol. 3019, Proc. Parallel Process. Appl. Math., pp. 544–551.

[11] Message Passing Interface Forum, "MPI: A message-passing interface standard," Int. J. Supercomput. Applicat., Tech. Rep., Los Angeles, CA, 1994.

[12] D. R. Butenhof, *Programming With POSIX Threads*, ser. Professional Computing. Reading, MA: Addison-Wesley, 1997.

[13] A. A. Zhigljaysky, *Theory of Global Random Search*. Norwell, MA: Kluwer, 1991.

[14] H. Mühlenbein, "The equation for response to selection and its use for prediction," *Evol. Comput.*, vol. 5, pp. 303–346, 1998.

[15] M. Sebag and A. Ducoulombier, "Extending population-based incremental learning to continuous search spaces," in *Parallel Problem Solving from Nature—PPSN V*. Berlin, Germany: Springer-Verlag, 1998, pp. 418–427.

[16] J. S. De Bonet, C. L. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," *Adv. Neural Inf. Process. Syst.*, vol. 9, . , 1996.

[17] P. Larrañaga, R. Etxeberria, J. Lozano, and J. Peña, "Optimization by learning and simulation of Bayesian and Gaussian networks," Dept. Comput. Sci. Artif. Intell., Univ. Basque Country, Guipuzcoa, Spain, Tech. Rep. KZZA-IK-4-99, 1999.

[18] ——, "Optimization in continuous domains by learning and simulation of Gaussian networks," in *Proc. Workshop Optim. Building Using Probabilistic Models, Workshop Within the 2000 Genetic Evol. Comput. Conf.*, Las Vegas, NV, 2000, pp. 201–204.

[19] R. Etxeberria and P. Larrañaga, "Global optimization with Bayesian networks," in *Proc. II Symp. Artif. Intell. CIMAF99 (Special Session Distrib. Evol. Optim.)*, La Habana, Cuba, 1999, pp. 332–339.

[20] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña, "Combinatorial optimization by learning and simulation of Bayesian networks," in *Proc. Conf. Uncertainty Artif. Intell.*, Stanford, CA, 2000, pp. 343–352.

[21] M. Pelikan and D. E. Goldberg, "Genetic algorithms, clustering, and the breaking of symmetry," in *Lecture Notes in Computer Science*, 2000, vol. 1917, Proc. Parallel Problem Solving from Nature—PPSN VI.

[22] ——, "Hierarchical problem solving and the Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf.*, vol. 1, D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmée, and H.-G. Beyer, Eds., San Francisco, CA, 2000, pp. 267–274.

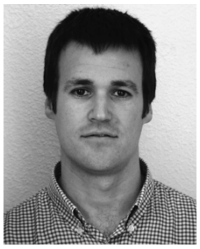
[23] ——, "Research on the Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf. Workshop Program*, vol. 1, A. Wu, Ed., 2000, pp. 212–215.

[24] H. Mühlenbein and T. Mahning, "FDA—a scalable evolutionary algorithm for the optimization of additively decomposed functions," *Evol. Comput.*, vol. 7, no. 4, pp. 353–376, 1999.

[25] M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Comput. Optim. Applicat.*, vol. 21, no. 1, pp. 5–20, 2002.

[26] Q. Zhang, "On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 80–93, Feb. 2004.

[27] Q. Zhang and H. Mühlenbein, "On the convergence of a class of estimation of distribution algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 127–136, Apr. 2004.
[28] C. Gonzalez, J. Lozano, and P. Larrañaga, "Mathematical modeling of discrete estimation of distribution algorithms," in *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, P. Larrañaga and J. A. Lozano, Eds. Norwell, MA: Kluwer, 2002, pp. 147–166.
[29] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*. New York: Springer-Verlag, 1997.
[30] R. Shachter and C. Kenley, "Gaussian influence diagrams," *Manage. Sci.*, vol. 35, pp. 527–550, 1989.
[31] G. Schwarz, "Estimating the dimension of a model," *Ann. Statist.*, vol. 7, no. 2, pp. 461–464, 1978.
[32] P. Spirtes, C. Glymour, and R. Scheines, "An algorithm for fast recovery of sparse causal graphs," *Soc. Sci. Comput. Rev.*, vol. 9, pp. 62–72, 1991.
[33] ——, *Causation, Prediction, and Search*. New York: Springer-Verlag, 1993, Lecture Notes in Statistics 81.
[34] P. W. F. Smith and J. Whittaker, "Edge exclusion tests for graphical gaussian models," in *Learning in Graphical Models*. Dordrecht, The Netherlands: Kluwer, 1998, pp. 555–574.
[35] J. Whittaker, *Graphical Models in Applied Multivariate Statistics*. New York: Wiley, 1990.
[36] M. Henrion, "Propagating uncertainty in Bayesian networks by probabilistic logic sampling," in *Uncertainly in Artificial Intelligence*, J. F. Lemmer and L. N. Kanal, Eds. Amsterdam, The Netherlands: North-Holland, 1988, vol. 2, pp. 149–163.
[37] B. D. Ripley, *Stochastic Simulation*. New York: Wiley, 1987.



Alexander Mendiburu received the B.S. degree in computer science from The University of the Basque Country, Guipuzcoa, Spain, in 1995.

Since 1999, he has been a Lecturer at the Department of Computer Architecture and Technology, University of the Basque Country. His main research areas are evolutionary computation, probabilistic graphical models, and parallel computing.

Jose A. Lozano received the B.S. degrees in mathematics and computer science and the Ph.D. degree from the University of the Basque Country, Guipuzcoa, Spain, in 1991, 1992, and 1998, respectively.

Since 1999, he has been an Associate Professor of Computer Science at the University of the Basque Country. He has edited three books and has published over 20 refereed journal papers. His main research interests are evolutionary computation, machine learning, probabilistic graphical models, and bioinformatics.

Prof. Lozano is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and was a Co-Chair of the 8th International Conference on Parallel Problem Solving from Nature (PPSN'04).

José Miguel-Alonso received the Ph.D. degree in computer science from the University of the Basque Country, Guipuzcoa, Spain, in 1996.

He is a Full Professor at the Department of Computer Architecture and Technology, University of the Basque Country. His research interests include parallel computing, networks for parallel systems, and network (cluster, grid) computing.

Dr. Miguel-Alonso is a member of the IEEE Computer Society.