Abstract— **In this paper new parallel versions of some Estimation of Distribution Algorithms are proposed. Particularly, the learning phase of the probability distribution from the selected individuals has been parallelized for different algorithms that use probabilistic graphical models -Bayesian networks and Gaussian networks-. In discrete domains, we explain the parallelization of $EBNA_{BIC}$ and $EBNA_{PC}$ algorithms, while in continuous domains the selected algorithms are $EGNA_{BIC}$ and $EGNA_{EE}$. This new implementation allows execution of the algorithms in several architectures. The parallelization was developed mixing two different options: Message Passing Interface and threads. Some experiments were executed for all the algorithms in a cluster of computers showing important performance gains comparing with the sequential versions.**

Parallel Estimation of Distribution Algorithms: New Approaches

Alexander Mendiburu, Department of Computer Architecture and Technology

Jose A. Lozano, Department of Computer Science and Artificial Intelligence

Jose Miguel, Department of Computer Architecture and Technology

The University of the Basque Country

San Sebastian, Spain

***Index Terms*— Evolutionary Computation, Estimation of Distribution Algorithms, Probabilistic Graphical Models, Cluster Computing, Performance Evaluation**

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 most 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.

This work was supported by the University of the Basque Country under the grant 9/UPV/EHU 00140.226-12084/2000 and the Ministerio de Ciencia y Tecnologia, Spain, under contract MCYT TIC2001-0591-C02-02.

In the literature a number of interesting parallel proposals have been made in the field of evolutionary computation. Efforts in the design of parallel evolutionary algorithms have focused on two areas. On the one hand, looking for algorithms where many populations evolve in parallel, and on the other, concentrating on parallelizing computation of the objective function. A good summary of different ideas for designing parallel GAs can be found in [1].

Recently, a new group of algorithms has been proposed in the evolutionary computation field: Estimation of Distribution Algorithms (EDAs) [2]. 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.

However, in this approach, it is the estimation of the joint probability distribution that causes the bottleneck. To avoid this problem, several authors have proposed simplifying assumptions on the probability model to be learnt at each step.

These assumptions concern the relations between the variables that the probability model takes into account. EDAs range from those that assume that all variables are indepen-

dent, to those that assume second order statistics and those that consider unrestricted models.

Of these, the ones that obtain the best results are those that use probability models with unrestricted dependencies [2]. In these algorithms, the probability distribution is codified by a probabilistic graphical model. 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 takes 50-99% of the total execution time of the algorithm. 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.

In the field of EDAs, previous parallel implementations can be found for some algorithms. In [3] two different parallel proposals for an algorithm that uses probabilistic graphical models in the combinatorial field ($EBNA_{BIC}$) are proposed. These implementations use threads over 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 [4] and [5] where two different parallel versions of the Bayesian Optimization Algorithm (BOA) [6] are proposed using a pipelined parallel architecture and clusters of computers respectively. Recently, in [7] the parallelization of the learning of decision trees using multithreaded techniques has been proposed.

Another way of parallelization can be found in [8] where a basic framework that facilitates the development of new multiple-deme parallel EDAs is presented.

In this paper we propose new parallel approaches 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.

In our versions we mix two parallel techniques: Message Passing Interface (MPI) [9] and threads [10]. Consequently, unlike the proposal of [3], the parallel solution can be used in a multiprocessor computer (communications are made using shared memory), in a cluster (where each computer communicates with others using messages) or any combination of both scenarios. Also, our work extends [4] and [5] allowing the learning of the Bayesian network without taking into account order restrictions, even though the objective function is not parallelized. Finally parallel approaches for algorithms of the continuous domains are also proposed, particularly: $EBNA_{BIC}$ and $EBNA_{PC}$ algorithms for discrete domains and $EGNA_{BIC}$ and $EGNA_{EE}$ for continuous domains.

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, the conclusions obtained and 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 [11], although similar approaches can be found in [12]. 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 containing the selected individuals from the previous generation. Thus, the interrelations -building blocks- between the different variables representing the individuals are explicitly expressed through the joint probability distribution associated

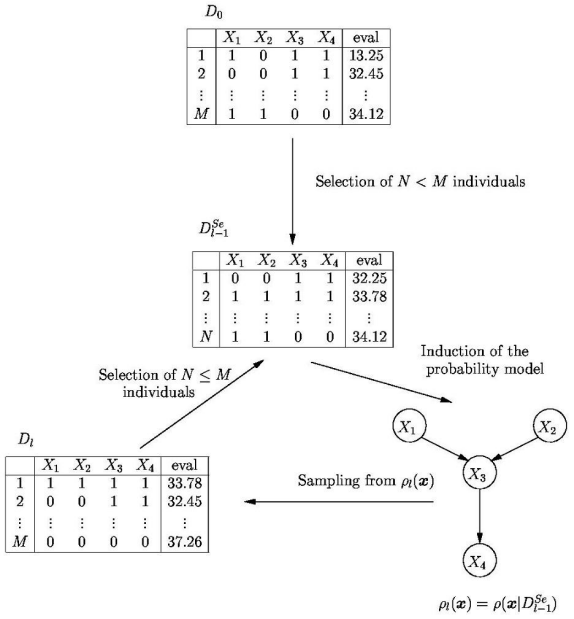

Fig. 1. EDA approach to optimization

with the individuals selected at each generation. In order to understand the behavior of this heuristic, a common model for all EDAs is introduced:

1) Generate the first population D_0 of M individuals and evaluate each of them. Usually this generation is made supposing a uniform distribution on each variable.
2) N individuals are selected from the set of M following a given selection method.
3) The n (size of the individual) dimensional probability model that shows the interdependencies among the variables best is induced.
4) Finally, a new population of M individuals is generated based on the sampling of the probability distribution learnt in the previous step.

Steps 2, 3 and 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 graphic schema of this process (for $n = 4$) can be seen in Fig. 1.

The probabilistic graphical model learnt at each step has an important influence on the behavior of the EDA algorithm (computing times and obtained results). Therefore, the algorithms are classified bellow taking into account the complexity of this probability model and, particularly, the dependencies they consider:

- 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, for example, $UMDA$ [13] or $PBIL_c$ [14].
- Bivariate dependencies: Estimation of the joint probability can be done quickly without assuming independence between the variables by taking into consideration dependencies between pairs of variables. This group includes $MIMIC$ [15], $MIMIC_c$ ([16], [17]).
- Multiple dependencies: All possible dependencies among the variables are considered without taking into account required complexity. $EBNA_{BIC}$ ([18], [19]), BOA ([6], [20], [21], [22]) or $EGNA_{EE}$ ([16], [17]) are some of the algorithms that belong to this group.

For detailed information about the characteristics and different algorithms that constitute the family of EDAs, see [2] and [23].

The algorithms of the last group, defined in the previous classification as 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 type of networks.

1) Bayesian Networks: Formally, a Bayesian network [24] over a domain $\mathbf{X} = (X_1, \ldots, X_n)$ is a pair (S,$\boldsymbol{\theta}$) representing 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:

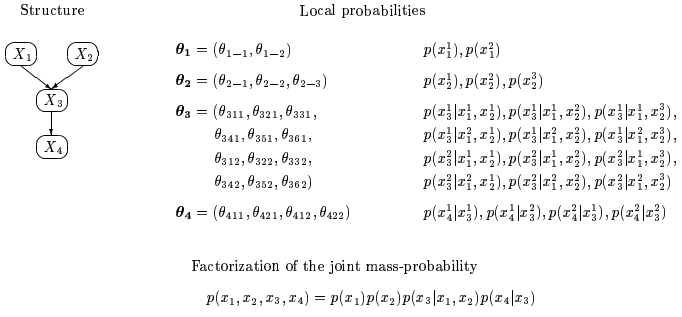

Fig. 2. Structure, local probabilities and resulting factorization for a Bayesian network with four variables (X_1, X_3 and X_4 with two possible values, and X_2 with three possible values)

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

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). On the other hand, $\boldsymbol{\theta}$ 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:

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

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, the parameter θ_{ijk} represents the conditional probability of variable X_i being in its k^{th} value, knowing that the set of its parent variables is in its j^{th} value. Therefore, the local parameters are given by $\boldsymbol{\theta}_i = (((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i}) \quad i = 1,\ldots,n$. An example of a Bayesian network can be seen in Fig. 2.

Concerning EDAs, the learning phase of the algorithm involves the learning of a Bayesian network from the selected individuals at each generation, that is, learning the structure (the graph S) and learning 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: $EBNA_{BIC}$ and $EBNA_{PC}$, respectively.

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

$$f(x_i \mid \boldsymbol{pa}_i, \boldsymbol{\theta}_i) \equiv \mathcal{N}(x_i; m_i + \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_j - m_j), v_i) \tag{3}$$

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. We call a probabilistic graphical model constructed with these local density functions a Gaussian network after [25].

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.

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

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

where $\boldsymbol{\mu}$ is the vector of means, Σ is an $n \times n$ covariance matrix, and $|\Sigma|$ denotes the determinant of Σ. The inverse of this matrix, $W = \Sigma^{-1}$, whose elements are denoted by w_{ij}, is referred to as the precision matrix.

An example of a Gaussian network can be seen in Fig. 3.

As explained about Bayesian networks, there are also

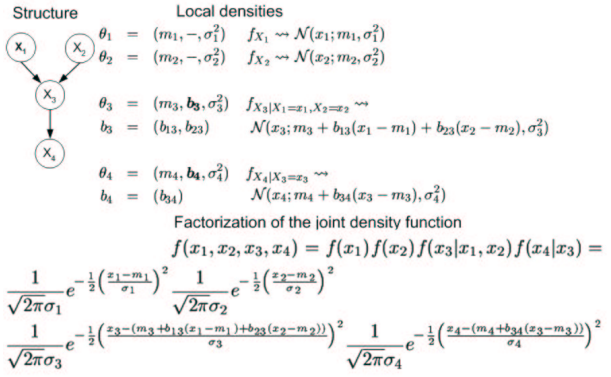

Fig. 3. Structure, local probabilities and resulting factorization for a Gaussian network with four variables

different methods to complete Gaussian network learning. The ones we point out are "score + search" and "detect conditional (in)dependencies". In sections V-A and V-B the parallelization of two algorithms that use these approaches are shown: $EGNA_{BIC}$ ("score + search") and $EGNA_{EE}$ (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 problems and 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 be focused on understanding how the learning process proceeds and finding a good parallel approach for this phase.

Before explaining the implementation, two important concepts should be explained: MPI and Threads.

TABLE I
Time measurement(%) of the learning phase for different algorithms and problems.

	individual sizes		
Algorithm, Problem	**50**	**150**	**250**
$EBNA_{BIC}$, OneMax	84.4	98.6	99.4
$EBNA_{BIC}$, EqProd	64.7	96.5	98.8
$EBNA_{PC}$, OneMax	80.9	98.0	99.1
$EBNA_{PC}$, EqProd	65.9	96.3	98.6
Algorithm, Problem	**20**	**50**	**70**
$EGNA_{BIC}$, Sphere	75.5	98.3	98.6
$EGNA_{BIC}$, Rosen.	64.0	96.2	97.4
Algorithm, Problem	**50**	**150**	**350**
$EGNA_{EE}$, Sphere	10.3	44.6	75.5
$EGNA_{EE}$, Rosen.	9.7	42.8	77.4

Message Passing Interface [9] provides a layer that can be used by application process to communicate over the network with processes that are physically executed in other computers. It was designed and standardized by the MPI forum, a group of academic and industrial experts for use in very different types of parallel computers. A wide set of procedures are available for managing, sending and receiving messages.

Concerning threads [10], most operating systems have thread support, that is, they include capabilities to spawn new processes (actually, threads) that share the same variable space. In 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.

With regard to implementations, a two-level Master-Slave scheme has been used. At the first level (communication between computers) the communication is carried out using MPI, where a computer acts as manager, coordinating communication with other computers (workers). These workers execute the orders sent by the manager and return the results to it. In some situations, the manager must wait for a long

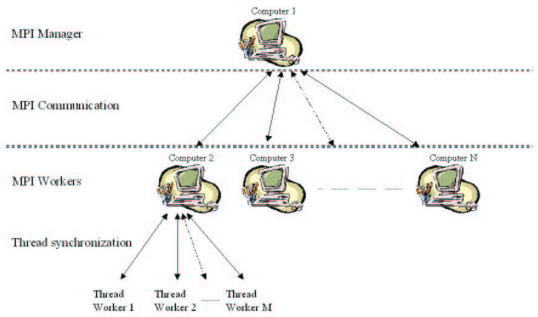

Fig. 4. Schema of the two-level Master-Slave communication

time until all the workers finish their work, and this makes it possible for the manager to act also as a worker making use of this idle time. At the second level, we present another Master-Slave schema (inside each computer), where the MPI node (computer) is the master, and the workers are the multiple threads that can be created. This is a good choice when computers are multiprocessors. This schema is depicted in Fig. 4.

Implementations of the parallel algorithms were made (except for $EBNA_{PC}$ algorithm) using both parallel techniques (MPI and threads). In many environments, machines available to run high CPU-consuming progrmas are multiprocessors, clusters of single-processor nodes or even clusters of multiprocessors. Our implementations can be easily adapted to any of these environments by selecting the right choice of mechanism for inter-node and intra-node communication and synchronization.

Finally, it should be noted that all the pseudocodes introduced in this paper for the different algorithms describe the master-slave communication for the first level (MPI communication between computers). The reason is that the schemes for the second level master-slave communication are similar to the first level (although there are no data send/receive operations because there is a single shared memory space).

It is important to remark that, throughout this paper, results for the parallel implementations are compared to an optimized sequential version of the programs, running on a single processor.

IV. Discrete domain

A. $EBNA_{BIC}$ algorithm [18], [19]

1) Algorithm description: This algorithm maintains the common schema described for EDAs.

During the learning process, 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 as a measure of its performance compared to 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.

EBNA$_{BIC}$ uses the penalized maximum likelihood score denoted by BIC (Bayesian Information Criterion) [26]. Given a structure S and a dataset D (set of selected individuals), the BIC score can be written as:

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

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 j^{th} value.
- N_{ijk} is the number of individuals in D in which variable X_i takes its k^{th} value and variables $\boldsymbol{Pa_i}$ take their j^{th} value.

An important property of this score is that it is decomposable. This means that the score 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 ($BIC(i, S, D)$):

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

where

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

The structure search algorithm used in EBNA$_{BIC}$ 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, as long as it results in a DAG (Directed Acyclic Graph) structure (note that the structure of a Bayesian network must always be a DAG). This cycle 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 $BIC(i, S, D)$ 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 remaining steps, only changes to the BIC due to arc modifications related to the variable X_i (it is assumed that in the previous step, S was updated with the arc modification (j, i)) need to be considered. 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 $n - 2$.

$EBNA_{BIC}$ **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. $EBNA_{BIC}$

We use four memory structures for this algorithm. A vector $BIC[i]$, $i = 1, 2, \ldots, n$, where $BIC[i]$ 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 last 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.

2) Parallel approach. First version: This first version is a straight-forward parallelization of the sequential algorithm. That is, the only changes made in the code are the ones necessary to implement the parallel version maintaining sequential functionality.

As parallelization is proposed only for the learning phase,

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 master 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 -regarding the computation of the decomposed BIC score and maintenance of the local S structure. Workers need their own data structures -remember that they are executed 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; and the second type of order, to maintain the integrity of the S local structure -each node addition or deletion means that the manager must notify the workers that the (i, j) edge has changed.

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 a BIC computation for a subset of variables must be executed in a worker, it can use the previously created threads. Each of them will independently compute a BIC value for a determined variable (continuing with another variable until all variables have been calculated). It can be seen that in every "MPI worker", new workers (threads) are locally created, all working together 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, with the process being transparent for the MPI communication.

The pseudocode for this algorithm is shown in Fig. 6 (manager) and Fig. 7 (workers). It must be remembered that included in the manager's pseudocode are its duties as a manager and also as a worker.

$EBNA_{BIC}$ **Algorithm. First parallel version. Manager**

Input: D, S, $paths$
Step 1. send D to the workers and set the number of variables ($NSet$) to work with
Step 2. send "calculate BIC" order to the workers
for each variable i in $NSet$
calculate $BIC[i]$
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
for each variable k in $NSet$
if $(k \neq i)$ calculate $G[k, i]$
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. **for** each variable k in $NSet$
if $(k \neq i$ **and** $k \neq j)$ calculate $G[k, i]$
Step 11. receive from workers all the changes and update G
Step 12. **go to** Step 7

Fig. 6. Pseudocode for the parallel structural learning phase. First version of the $EBNA_{BIC}$ algorithm for the manager.

3) Parallel approach. Second version: In this second version, some changes have been made with respect to the sequential algorithm. Looking at the algorithm, it can be observed that all the computations are made for each $G[i, j]$ but, while looking for the better $G[r, s]$ value, it must be verified that the Bayesian network is still a DAG (Directed Acyclic Graph).

This means that the workers complete some work that can be considered useless. Why must they compute a BIC score if it does not fulfill the DAG property?

The first idea to deal with is to send to the workers the data structure needed for them to check if the DAG property is still maintained ($paths$). However, this solution 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 in each worker (only those

$EBNA_{BIC}$ **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 $NSet$
if $(k \neq i$ **and** $k \neq j)$ 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 $EBNA_{BIC}$ algorithm for the workers.

that kept the DAG property). Therefore, great time differences can exist between the different workers (unbalanced distribution) and the manager must wait until the last of them finishes. As a consequence, it is necessary to determine previously 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 ones. Then, this set is sent to the workers, and each of them calculates the BIC criterion for its proportional subset assuring an equally distributed work load. In Fig. 8 (manager) and Fig. 9 (workers) the changes made for this second version can be seen.

B. $EBNA_{PC}$ algorithm [18], [19]

1) Algorithm description: As previous algorithm, $EBNA_{PC}$ also maintains the common schema described for EDAs, but, unlike $EBNA_{BIC}$, 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

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

Input: D, S, $paths$
Step 1. send D to the workers and set the number of variables ($NSet$) to work with
Step 2. send "calculate BIC" order to the workers
for each variable i in $NSet$
calculate $BIC[i]$
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
calculate the structures that fit the DAG property and set the number of variables ($NSetDAG$) to work with
send to the workers the set of variables ($NSetDAG$) to work with
send "calculate $G[k,i]$" order to the workers
for each variable k in $NSetDAG$
calculate $G[k,i]$
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. calculate the structures that fit the DAG property and set the number of variables ($NSetDAG$) to work with
send to the workers the set of variables ($NSetDAG$) to work with and "calculate $G[k,i]$ for (i,j)" order
Step 10. **for** each variable k in $NSet$ calculate $G[k,i]$
Step 11. receive from workers all the changes and update G
Step 12. **go to** Step 7

Fig. 8. Pseudocode for the parallel structural learning phase. Second version of the $EBNA_{BIC}$ algorithm for the manager.

the final step, a Bayesian network is created based on the detected dependencies.

$EBNA_{PC}$ receives its name from the fact that the algorithm used to detect the conditional (in)dependencies is the PC algorithm, described in [27]. Like almost all 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, 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.

Concerning the data structures, it is necessary to define

$EBNA_{BIC}$ **algorithm. Second parallel version. Workers**

Step 1. create and initialize S_w 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 to the manager BIC results
"calculate $G[k,i]$"
receive the set of variables ($NSetDAG$) to work with
for each variable k in $NSetDAG$
if $(k \neq i)$ calculate $G[k,i]$
send G modifications to the manager
"calculate $G[k,i]$ for (i,j)"
receive the set of variables ($NSetDAG$) to work with
for each variable k in $NSetDAG$
if $(k \neq i$ **and** $k \neq j)$ 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. 9. Pseudocode for the parallel structural learning phase. Second version of the $EBNA_{BIC}$ algorithm for the workers.

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. Also, another structure -$SepSet$- is needed to save the independence relation between two variables and the set of variables conditioned on.

In Fig. 10 a pseudocode for the PC algorithm can be seen. $Adj(G,A)$ 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 $Adj(G,A)$ 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 [28].

2) Parallel approach. First version: Before implementing any parallel proposal, we did a study of execution times for the sequential algorithm. As can be seen in table I, in this algorithm the learning phase requires nearly 98% of the total execution time, so this is the phase that will be parallelized. As mentioned earlier, the learning phase can be divided into two different steps: detection of conditional (in)dependencies,

$EBNA_{PC}$ **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 $\mid Adj(G,X_i)\backslash\{X_j\} \mid \geq r$
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 $\mid Adj(G,X_i)\backslash\{X_j\} \mid \geq r$ 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 $\mid Adj(G,X_i)\backslash\{X_j\} \mid < r$
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. 10. Pseudocode for the sequential structural learning algorithm. $EBNA_{PC}$.

TABLE II
TIME MEASUREMENT(%) FOR DIFFERENT PROCESSES OF THE $EBNA_{PC}$ LEARNING PHASE.

	One-Max		**EqualProducts**	
Ind. size	*%Det. dep.*	*%Orient.*	*%Det. dep.*	*% Orient.*
50	98.1	0.4	97.7	0.3
150	98.9	0.4	99.0	0.3
250	99.2	0.4	99.2	0.3

and orientation. As can be seen in table II the most important action is the first one -detection of the independencies- and therefore, our parallel approach focuses only on this process.

Once we have explained the $EBNA_{PC}$ algorithm, we propose the parallel algorithm. The model induction begins with a complete undirected graph, and in the first step, all zero order conditional independencies must be searched. This process can be carried out independently by each computational node. Thus, all the pairs are equally distributed between the manager and workers (since the manager must wait for the results it will be used just as another worker). With n being the number of

variables, the dependencies between $n(n-1)/2$ pairs must be checked. Thus, each worker takes a set of $(n(n-1)/2)/comp$ pairs to detect dependencies where $comp$ is the number of computational nodes (including the manager).

In the following 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 present 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 "problematic" 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 can not 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 obliges the worker to wait until a pair can be selected. This process is repeated until no more dependencies can be found.

In those 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 (the edge must be removed or not) 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 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 made sequentially.

The pseudocode of this parallel version of the $EBNA_{PC}$ algorithm can be seen in Fig. 11 (manager) and Fig. 12 (workers).

3) Parallel approach. Second version: Observing the operation mode of the parallel proposal when calculating dependencies with adjacent sets of cardinality greater than zero, the manager selects the next pair to calculate and sends it to the worker that has asked for a job, repeating this process for all the requests. During these requests -until the workers compute the assigned pair- some time elapses and the manager does nothing.

An improvement taking advantage of this idle time, is the utilization of the manager as a worker to complete independence tests whilst waiting for another worker's request. That is, if there is a worker asking for a pair to calculate, the manager selects a new pair and sends it to the worker but, while there are no more requests, the manager selects a new pair and calculates it itself.

In Fig. 13 the modifications made in the manager version of the parallel program can be seen. The worker's schema is not repeated again because it is the same as in the first version of the $EBNA_{PC}$.

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

Step 1. form the complete undirected graph G on
vertex set $V = \{X_1, \ldots, X_n\}$
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. 11. Pseudocode for the parallel structural learning phase. First version of the $EBNA_{PC}$ algorithm for the manager.

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

Step 1. form the complete undirected graph G on
vertex set $V = \{X_1, \ldots, X_n\}$
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. 12. Pseudocode for the parallel structural learning phase. First version of the $EBNA_{PC}$ algorithm for the workers.

C. Selected problems and experiments for the discrete domain

This section reports the scenario used for the experiments, the selected problems and the results obtained.

Beginning with the computer resources, the machine used -which is the same for discrete and continuous domains-, is a cluster of 10 nodes, where each node has two AMD ATLON MP 2000+ processors (CPU frequency 1.6GHz), 512kB cache memory and 1GB of RAM. 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.

The experiments were executed combining MPI (between the nodes) and threads (two inside each node, the same as the number of processors). An exhaustive study should be made to determine the best combination of MPI nodes and threads for a particular algorithm, problem and machine, but this is outside the scope of this paper. Hence, in our tables and figures only the results obtained by using the configuration outlined in figure 14 will be taken into account.

Since we focus on the learning phase, we selected two problems that have a very simple objective function: the well-known *OneMax* and the *EqualProducts* functions.

The *OneMax* problem consists of maximizing:

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

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.

For the *EqualProducts* function a set of random real numbers $\{a_1, a_2, \ldots, a_n\}$ in the interval $[0, u]$ has to be given (in our case we used $[0, 2]$). The objective is to select

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

Step 1. form the complete undirected graph G on
vertex set $V = \{X_1, \ldots, X_n\}$
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
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
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 are 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. 13. Pseudocode for the parallel structural learning phase. Second version of the $EBNA_{PC}$ algorithm for the manager.

some numbers in such a way that the difference between the products of selected and unselected numbers is minimized. Mathematically:

$$EqualProducts(\boldsymbol{x}) = \left| \prod_{i=1}^{n} h(x_i, a_i) - \prod_{i=1}^{n} h(1 - x_i, a_i) \right|$$

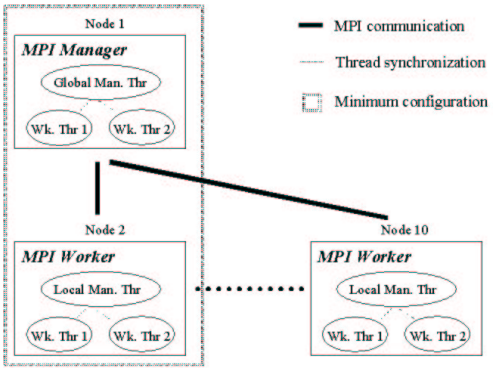

Fig. 14. Configuration used for the experiments

where function h is defined as:

$$h(x, a) = \begin{cases} 1 & \text{if } x = 0 \\ a & \text{if } x = 1 \end{cases}$$

In this problem we do not know the optimum, but the closer the solution is to zero, the better.

Regarding the experiments, several executions were done with different numbers of nodes, measuring different aspects or dimensions: time-related, solution quality-related and algorithm performance-related. Each different experiment was repeated 10 times. The results presented here are the average of all the values obtained.

1) time-related dimension: The main goal of a parallelization process is usually to reduce the computation time required for a problem to be solved. In the following tables and figures, execution time and efficiency as well as the speedup are presented. These last two concepts are defined as:

- Speed-up = $\frac{sequential\ time}{parallel\ time}$
- Efficiency = $\frac{speed-up}{numberofprocessors}$

For both the $EBNA_{BIC}$ and $EBNA_{PC}$ algorithms, the size of the population is 2,000. From this population, the best 1,000 individuals are selected at each generation and the executions are stopped when the 15th generation is reached. The size of the individual is different for each algorithm: 200

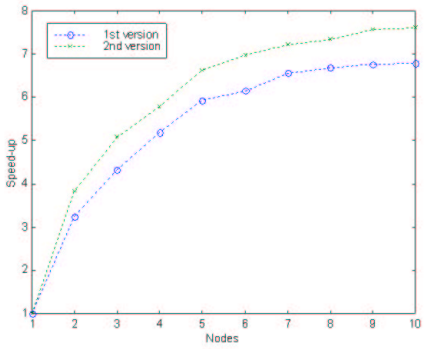

Fig. 15. Speed-up for $EBNA_{BIC}$ algorithm with the *OneMax* problem

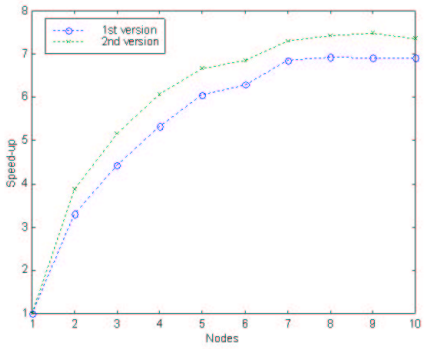

Fig. 16. Speed-up for $EBNA_{BIC}$ algorithm with the *EqualProducts* problem

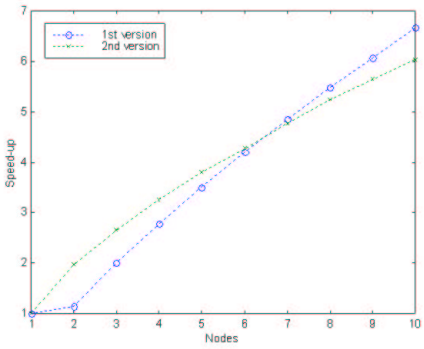

Fig. 17. Speed-up for $EBNA_{PC}$ algorithm with the *OneMax* problem

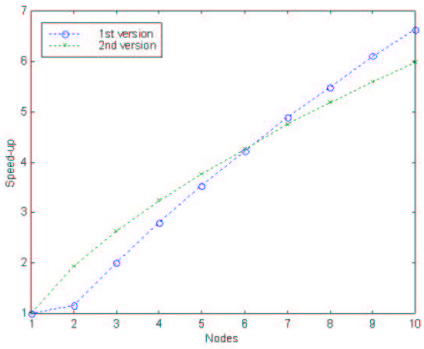

Fig. 18. Speed-up for $EBNA_{PC}$ algorithm with the *EqualProducts* problem

for $EBNA_{BIC}$ and 250 for $EBNA_{PC}$. All these parameters were selected looking for a sequential execution time close to 30 minutes, time enough to observe the behavior of the algorithms.

Tables III and IV summarize the results of the experiments with *OneMax* and *EqualProducts* functions for both algorithms. It can be seen that there is a dramatic time reduction in all cases as the number of workers increases.

Particularly, for $EBNA_{BIC}$ algorithm, the times obtained by the second parallel version are better than those of the first, and it is significant that when 7 or more nodes are used the computation time decreases more slightly -see Fig. 15 and 16. Due to the parallelization process, communication between the manager and the workers is necessary to maintain data integrity, but when the number of workers is excessive, the time spent to communicate and update the structures is greater than the computation reduction obtained.

Before explaining the results obtained for the $EBNA_{PC}$ algorithm, it is necessary to point out that for this particular algorithm only the first level parallelization (MPI) was used. Remembering the pseudocode (Fig. 11), notice that workers manage one pair each time, so it is not worth it to create second level workers (threads) to have each one compute part of the work.

Once this has been clarified, we can see in Fig. 17 and 18 the different tendencies of the two parallel versions of the $EBNA_{PC}$ algorithm. In the first one (the manager waits for the workers to finish without doing anything except distributing pairs) when there are few workers the waiting-time is very high, while in the second version this idle time is used by the manager to act as a worker, yielding better results for the second parallel version. However, when the number of workers increases (for 6 or more nodes), there are more requests from workers asking for a pair and therefore the manager has less

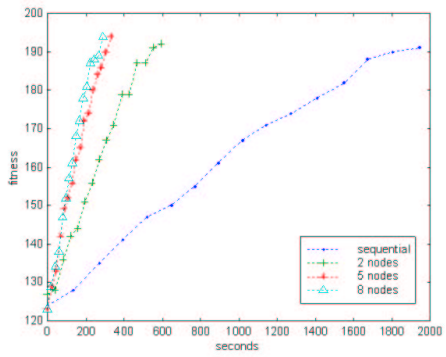

Fig. 19. Solution evolution for the first version of the $EBNA_{BIC}$ algorithm with the *OneMax* problem

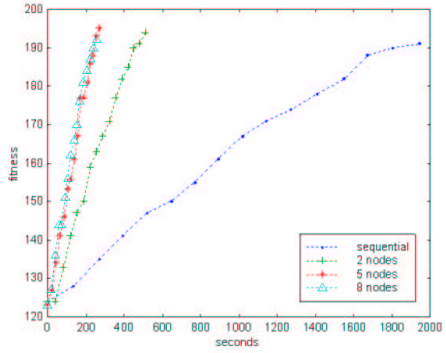

Fig. 20. Solution evolution for the second version of the $EBNA_{BIC}$ algorithm with the *OneMax* problem

idle time. If the manager also computes pairs, this forces the workers to wait, increasing the waiting-time.

2) quality-related dimension: As mentioned before, all the algorithm versions have exactly the same behavior: that of the sequential version. The stopping criteria for those EDA algorithms is diverse. In some cases, algorithms stop when a given quality of the solution is reached (without a-priory knowledge of the execution time required to reach it). In many others, we look for the best possible solution given a fixed execution time (common in real environments).

In Fig. 19, 20, 21, 22, 23, 24, 25 and 26 we show the evolution of the solution's quality with time. It can be seen that when execution time is fixed, quality of parallel solutions are considerably better.

3) performance-related dimension: When a parallel solution for an algorithm is proposed, the work must be distributed

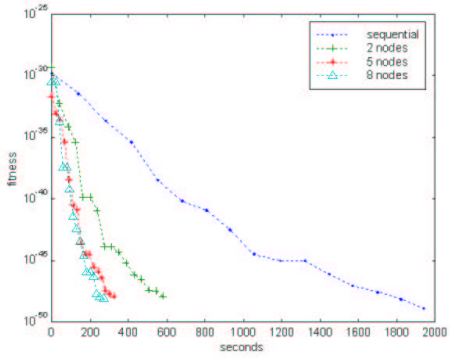

Fig. 21. Solution evolution for the first version of the $EBNA_{BIC}$ algorithm with the *Equal Products* problem

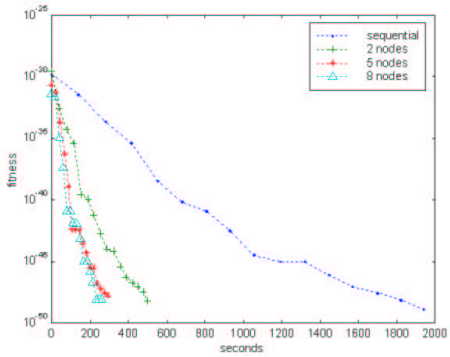

Fig. 22. Solution evolution for the second version of the $EBNA_{BIC}$ algorithm with the *Equal Products* problem

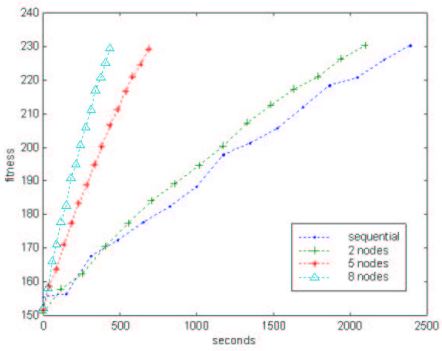

Fig. 23. Solution evolution for the fist version of the $EBNA_{PC}$ algorithm with the *OneMax* problem

correctly between all the nodes that will work in parallel. The execution time for each node is shown in tables V, VI, VII and VIII ($EBNA_{BIC}$ algorithm with *OneMax* and *Equal-Products* problems) and tables IX, X, XI and XII ($EBNA_{PC}$ algorithm with *OneMax* and *EqualProducts* problems). The node with the highest computation times is obviously the manager, since it is the one that centralizes all the processes,

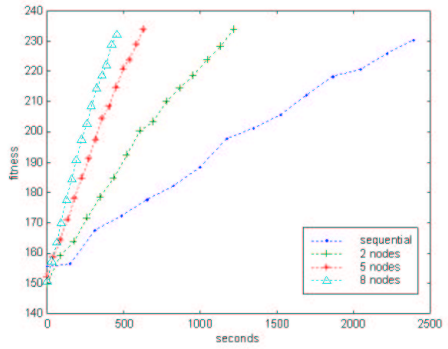

Fig. 24. Solution evolution for the second version of the $EBNA_{PC}$ algorithm with the *OneMax* problem

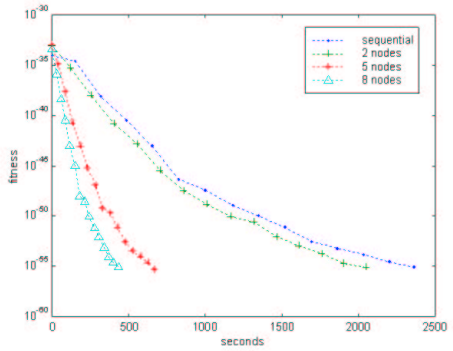

Fig. 25. Solution evolution for the first version of the $EBNA_{PC}$ algorithm with the *Equal Products* problem

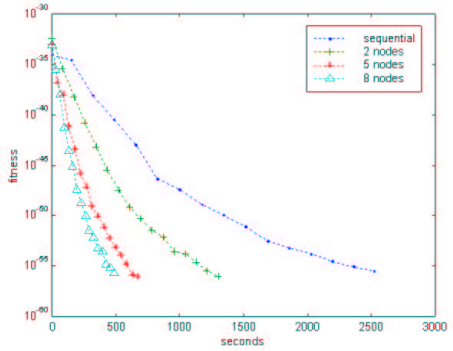

Fig. 26. Solution evolution for the second version of the $EBNA_{PC}$ algorithm with the *Equal Products* problem

and as we can see the work is equally distributed among the workers. There are slight differences due to the work distribution (it is not always an exact division) and the communication connections managed by the MPI implementation, but distribution of the work is properly balanced.

TABLE III

TIME-RELATED EXPERIMENTAL RESULTS FOR *OneMax* AND *Equal Products* USING THE TWO $EBNA_{BIC}$ PARALLEL VERSIONS

	OneMax				***EqualProducts***			
	1st version		***2nd version***		***1st version***		***2nd version***	
Nod	**secs**	**eff**	**secs**	**eff**	**secs**	**eff**	**secs**	**eff**
seq.	1946	1.00	1946	1.00	1929	1.00	1929	1.00
2	600	0.81	508	0.96	585	0.82	499	0.97
3	450	0.72	383	0.85	436	0.74	373	0.86
4	375	0.65	336	0.72	362	0.67	318	0.76
5	328	0.59	294	0.66	319	0.60	289	0.67
6	316	0.51	279	0.58	306	0.53	281	0.57
7	296	0.47	270	0.51	281	0.49	264	0.52
8	291	0.42	265	0.46	278	0.43	260	0.46
9	288	0.38	257	0.42	279	0.38	258	0.42
10	287	0.34	256	0.38	279	0.35	262	0.37

TABLE IV

TIME-RELATED EXPERIMENTAL RESULTS FOR *OneMax* AND *Equal Products* USING THE TWO $EBNA_{PC}$ PARALLEL VERSIONS

	OneMax				***EqualProducts***			
	1st version		***2nd version***		***1st version***		***2nd version***	
Nod	**secs**	**eff.**	**secs**	**eff.**	**secs**	**eff.**	**secs**	**eff.**
seq.	2566	1.00	2566	1.00	2525	1.00	2525	1.00
2	2250	0.57	1307	0.98	2184	0.58	1300	0.97
3	1286	0.67	968	0.88	1254	0.67	961	0.88
4	926	0.69	789	0.81	904	0.70	782	0.81
5	734	0.70	677	0.76	714	0.71	671	0.75
6	610	0.70	599	0.71	599	0.70	593	0.71
7	528	0.69	539	0.68	516	0.70	532	0.68
8	467	0.69	490	0.65	461	0.68	487	0.65
9	423	0.67	454	0.63	413	0.68	451	0.62
10	385	0.67	425	0.60	381	0.66	422	0.60

TABLE V

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{BIC}$ FIRST VERSION WITH *OneMax* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
588	538	-	-	-	-	-	-	-	-
431	369	361	-	-	-	-	-	-	-
371	292	291	290	-	-	-	-	-	-
307	243	232	232	231	-	-	-	-	-
280	230	201	199	199	198	-	-	-	-
246	198	184	173	167	167	166	-	-	-
233	182	167	166	156	155	155	156	-	-
227	167	147	146	139	137	136	136	135	-
203	157	131	130	121	122	123	121	121	124

TABLE VI

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{BIC}$ SECOND VERSION WITH *OneMax* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
464	418	-	-	-	-	-	-	-	-
373	310	309	-	-	-	-	-	-	-
317	244	242	240	-	-	-	-	-	-
247	189	179	178	175	-	-	-	-	-
242	192	168	166	166	166	-	-	-	-
221	173	158	147	146	146	146	-	-	-
199	154	140	141	131	130	129	128	-	-
196	134	122	122	112	112	112	112	110	-
183	138	113	112	104	103	104	102	101	102

TABLE VII

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{BIC}$ FIRST VERSION WITH *EqualProducts* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
574	522	-	-	-	-	-	-	-	-
430	365	358	-	-	-	-	-	-	-
346	273	272	271	-	-	-	-	-	-
299	235	225	224	223	-	-	-	-	-
268	216	189	189	187	188	-	-	-	-
244	193	180	169	164	164	163	-	-	-
218	167	154	154	144	145	145	146	-	-
220	159	140	141	130	131	132	132	131	-
205	157	132	132	122	123	123	123	123	124

TABLE VIII

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{BIC}$ SECOND VERSION WITH *EqualProducts* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
495	446	-	-	-	-	-	-	-	-
368	305	304	-	-	-	-	-	-	-
304	233	232	229	-	-	-	-	-	-
270	207	198	196	194	-	-	-	-	-
239	190	167	168	165	165	-	-	-	-
215	165	151	140	138	139	138	-	-	-
204	156	142	141	132	131	130	130	-	-
199	137	123	125	114	114	115	114	111	-
193	143	116	116	106	106	106	106	104	105

TABLE IX

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{PC}$ FIRST VERSION WITH *OneMax* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
983	2149	-	-	-	-	-	-	-	-
690	1206	1204	-	-	-	-	-	-	-
544	852	851	852	-	-	-	-	-	-
458	665	663	663	663	-	-	-	-	-
399	542	540	540	540	538	-	-	-	-
358	461	459	458	459	458	456	-	-	-
328	400	397	397	397	396	396	395	-	-
305	354	352	351	351	351	350	349	351	-
286	315	313	312	311	311	311	311	313	311

TABLE X

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{PC}$ SECOND VERSION WITH *OneMax* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
1308	1042	-	-	-	-	-	-	-	-
969	715	717	-	-	-	-	-	-	-
790	567	566	563	-	-	-	-	-	-
677	468	467	465	463	-	-	-	-	-
599	403	400	399	398	396	-	-	-	-
540	353	351	350	348	348	344	-	-	-
491	315	312	310	311	308	306	306	-	-
454	286	283	281	282	279	279	279	277	-
425	262	259	257	258	257	255	254	254	255

TABLE XI

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{PC}$ FIRST VERSION WITH *EqualProducts* PROBLEM.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
983	2081	-	-	-	-	-	-	-	-
691	1173	1170	-	-	-	-	-	-	-
545	829	827	827	-	-	-	-	-	-
458	641	640	640	640	-	-	-	-	-
400	528	527	527	527	525	-	-	-	-
359	446	443	443	443	443	441	-	-	-
329	389	388	387	387	385	385	385	-	-
305	340	338	337	338	337	336	336	338	-
287	307	306	304	305	302	302	303	306	303

TABLE XII
ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{PC}$ SECOND VERSION WITH *EqualProducts* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
1301	1034	-	-	-	-	-	-	-	-
962	722	719	-	-	-	-	-	-	-
782	560	559	558	-	-	-	-	-	-
671	462	462	460	459	-	-	-	-	-
594	398	395	393	394	391	-	-	-	-
533	349	346	344	344	342	340	-	-	-
488	311	309	307	307	306	303	304	-	-
452	281	281	281	279	275	276	274	276	-
422	259	256	255	255	253	252	251	251	251

V. CONTINUOUS DOMAIN

A. $EGNA_{BIC}$ algorithm [2]

1) Algorithm description: As we have already noted, EDA algorithms can be used in discrete and continuous domains. In this section we propose the parallelization of the $EGNA_{BIC}$ algorithm, which is similar to discrete $EBNA_{BIC}$, but modified for 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 $EBNA_{BIC}$ are also valid for this algorithm.

The score used to verify the performance of the Gaussian network obtained is the Bayesian Information Criterion (BIC). A general formula for this criterion is as follows:

$$\begin{aligned} &BIC(S,D) = \\ &\sum_{r=1}^{N}\sum_{i=1}^{n}[-\ln(\sqrt{2\pi v_i}) \\ &-\frac{1}{2v_i}(x_{ir} - m_i - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j))^2] \\ &-\frac{1}{2}\log(N)(2n + \sum_{i=1}^{n} \mid \boldsymbol{Pa}_i \mid) \end{aligned} \tag{8}$$

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 $\boldsymbol{Pa_i}$.

As was explained for $EBNA_{BIC}$ algorithm, this score can also be broken down to calculate separately the score for each variable. Accordingly, each variable X_i has associated with it a local BIC score ($BIC(i,S,D)$):

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

where

$$\begin{aligned} &BIC(i,S,D) = \\ &\sum_{r=1}^{N}[-\ln(\sqrt{2\pi v_i}) - \frac{1}{2v_i}(x_{ir} - mi - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j))^2] \\ &-\frac{1}{2}\log(N)(2+ \mid \boldsymbol{Pa}_i \mid) \end{aligned} \tag{10}$$

Consequently, the steps followed to parallelize this algorithm are similar to those seen for the $EBNA_{BIC}$, 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 the $EBNA_{BIC}$ algorithm. The first one, where workers calculate all possible arc modifications without considering whether the DAG property is fulfilled, and the second, where the manager makes a preprocess step to obtain the arc changes that maintain the DAG property, then sending to each worker a subset of those possible modifications.

Due to the similarity of this algorithm to the discrete one ($EBNA_{BIC}$) it is 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. $EGNA_{EE}$ algorithm [16], [17]

1) Algorithm description: In this algorithm the structure learning of the Gaussian network follows a "detecting conditional (in)dependencies" method.

Particularly, this method begins with a completed 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 [29]. The statistic to exclude the edge between X_1 and X_2 from a graphical Gaussian model, is $T_{lik} = -nlog(1 - r^2_{ij|rest})$ 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 [30] in terms of the maximum likelihood estimates of the elements of the precision matrix as $r_{ij|rest} = \hat{w}_{ij}(\hat{w}_{ii}\hat{w}_{jj})^{-\frac{1}{2}}$, where the precision matrix W is the inverse of the covariance matrix Σ.

[29] 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^2_1 variable.

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

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

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

Input: D
Step 1. calculate the means m and the covariance matrix Σ
Step 2. initialize the structure of the Gaussian network S
Step 3. calculate the inverse of Σ
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. 27. Pseudocode for the sequential learning algorithm. $EGNA_{EE}$.

TABLE XIII
TIME MEASUREMENT FOR DIFFERENT PROCESSES OF THE $EGNA_{EE}$ ALGORITHM.

	Sphere model		**Rosenbrock gener.**	
Ind. size	*%Learn.*	*%Sampl.*	*%Learn.*	*%Sampl.*
50	10.3	89.6	9.7	90.3
150	44.6	55.1	42.8	57.1
350	75.5	24.3	77.4	22.4

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

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

where $\mathbf{m}$ is the estimated means, $\mathbf{v}$ the estimated conditional variances and $\mathbf{b}$ the estimated regression coefficients.

Six main 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 and 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. 27.

2) Parallel approach. First version: As can be seen in table I (presented in section III), the learning phase does not take as long as with previous algorithms. Due to this, a deeper

TABLE XIV
TIME MEASUREMENT FOR DIFFERENT PROCESSES OF THE $EGNA_{EE}$ LEARNING PHASE.

	$\mathbf{EGNA_{EE}}$, Sphere model			
Ind. size	*%Σ*	*%Prec. matr.*	*%test*	*%Parm. learn.*
50	92.0	2.0	0.5	5.3
150	71.1	2.5	0.8	25.5
350	65.3	5.7	1.2	27.8
	$\mathbf{EGNA_{EE}$,Rosenbrock generalized			
Ind. size	*%Σ*	*%Prec. matr.*	*%test*	*%Parm. learn.*
50	92.0	2.0	0.5	5.4
150	69.0	2.6	0.7	27.6
350	66.5	5.3	1.0	27.2

analysis has been done to know how the computation time is distributed among the different procedures. Table XIII shows the most important procedures that are executed by the present algorithm: learning and sampling of the new individuals.

In this first parallel approach, the parallelization of the learning phase is presented, and due to the execution time distribution, this version will be completed -in the second approach- with the parallelization of the creation of new individuals.

In this particular case, 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:

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

Table XIV shows the times required by each of these processes. We can see that the two most computationally expensive phases are the first one, where means and Σ are calculated, and the fourth one, 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 quickly executed because it uses the values computed in the previous phases to obtain the r_{ij} values. Clearly, this distribution of the time is conditioned by the parameters of the particular problem. That is, if the size of the population grows or the size of the individuals changes the computation distribution between the four steps can be quite different. Concerning our problem, our parallelizing efforts must focus on steps one and four.

To calculate Σ it is necessary first 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 *comp* the number of nodes, each node will have $n/comp$ variables to compute. 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 node computes $(n(n-1)/2)/comp$ 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 distribution similar to the previous one is made, with each node computing a piece of $\boldsymbol{b}$ and $\boldsymbol{v}$.

Fig. 28 and Fig. 29 show the pseudocode for the parallel version.

3) Parallel approach. Second version: Due to the fact that the learning process is not as computationally weighty as in the previous algorithms, the results obtained are not as good as in the other programs. If we look at table XIII, we can see that there is another process that consumes a large portion of the total execution time: creation of the new individuals (sampling).

For this reason, we present as a second version, the parallelization of both phases: learning and sampling. Remembering

$EGNA_{EE}$ **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 and
define the set of variables $NSet$ to work with
Step 3. send "calculate means" order to the workers
Step 4. **for** each variable i in $NSet$
calculate the mean m_i
Step 5. Synchronize with all workers and update m
Step 6. send "calculate covariances" order to the workers
Step 7. define the set of pairs $NSet_p$ to work with
Step 8. **for** each pair i, j in $NSet_p$
calculate the covariance $\Sigma[i, j]$
Step 9. Synchronize with all workers and update Σ
Step 10. calculate the inverse of Σ
Step 11. **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 12. send "calculate parameters" order to the workers
Step 13. send S to the workers
Step 14. **for** each variable i in $NSet$
calculate the parameters b_i and v_i
Step 15. receive from the workers all the changes and update b and v
Step 16. send "stop" order to the workers

Fig. 28. Pseudocode for the parallel learning phase. First version of the $EGNA_{EE}$ algorithm for the manager.

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

Step 1. receive D structure from the manager and
calculate the set 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"
calculate the set 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 Σ
learn distribution
"calculate parameters"
receive S from the manager
for each variable i in $NSet$
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. 29. Pseudocode for the parallel learning phase. First version of the $EGNA_{EE}$ algorithm for the workers.

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. We use an adaptation of the Probabilistic Logic Sampling (PLS) proposed in [31].

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 $(\pi(i), \ldots, \pi(n))$ 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 $f(x_{\pi(i)}|\boldsymbol{pa}_{\boldsymbol{\pi(i)}})$. For the simulation of a univariate normal distribution, a simple method based on the sum of 12 uniform variables is applied [32].

A pseudocode for this sampling process can be seen in Fig. 30.

The parallelization approach is the following: If $NewPopSet$ is the amount of new individuals to be created and $comp$ the number of computational nodes, the manager sends to each worker the order to create $NewPopSet/comp$

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

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

Fig. 30. Pseudocode for the sequential sampling algorithm. $EGNA_{EE}$.

new individuals (the manager also creates new individuals). Once all these new individuals have been created, they are sent to the manager which adds them to the population and continues with the algorithm.

In Fig. 31 (manager) and Fig. 32 (workers) the parallel proposal for the sampling phase can be seen.

$EGNA_{EE}$ **sampling algorithm. Parallel version. Manager**

Input: S,$Order(\pi(i), ldots, \pi(n)$,$m$,$b$,$v$
Step 1. send m, b and v structures to the workers
Step 2. **for** each i in $NewPopSet$
calculate a new individual
Step 3. receive the new individuals from the workers
and update the population

Fig. 31. Pseudocode for the parallel sampling phase. $EGNA_{EE}$ algorithm for the manager.

$EGNA_{EE}$ **sampling algorithm. Parallel version. Workers**

Step 1. receive S, $Order(\pi(i), ldots, \pi(n)$, 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. 32. Pseudocode for the parallel sampling phase. $EGNA_{EE}$ algorithm for the workers.

C. Selected problems and experiments for the continuous domain

The scenario used for the continuous domain is the same as used for the discrete one, so this information can be obtained in section IV-C

In order to test the different proposals, we have chosen two standard functions broadly used in the literature for comparing optimization techniques.

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:

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

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

The second function is *Rosenbrock generalized*, a minimization problem proposed by [33], extending to n dimensions the original proposal [34].

It is defined as:

$$Rosenbrock\ generalized(\mathbf{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{17}$$

The optimum value is 0, which is obtained again for the individual whose components are all set to 1. The range of values is of $-10 \leq x_i \leq 10, \;\; i = 1, \ldots, n$.

Concerning analysis of the experiments, the same dimensions as for discrete domain algorithms have been taken into account for these continuous algorithms. Therefore we analyze different dimensions such as time-related, performance-related and solution quality-related.

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

	Sphere model				***Rosenbrock gener.***			
	1st version		***2nd version***		***1st version***		***2nd version***	
Nod	**secs**	**eff**	**secs**	**eff**	**secs**	**eff**	**secs**	**eff**
seq.	2973	1.00	2973	1.00	2771	1.00	2771	1.00
2	825	0.90	626	1.19	790	0.88	612	1.13
3	579	0.86	450	1.10	557	0.83	439	1.05
4	477	0.78	365	1.02	457	0.76	353	0.98
5	384	0.77	334	0.89	370	0.75	325	0.85
6	335	0.74	303	0.82	323	0.71	293	0.79
7	329	0.65	299	0.71	318	0.62	299	0.66
8	287	0.65	307	0.61	277	0.63	303	0.57
9	282	0.59	267	0.62	274	0.56	263	0.59
10	249	0.60	265	0.56	242	0.57	258	0.54

1) time-related dimension: In this section, the execution time as well as efficiency and speed-up are presented. The experiments were executed with a population of 2,000, with the number of individuals selected at each generation set at 1,000. As particular parameters, $EGNA_{BIC}$ was executed with an individual size of 60 and the number of generations was limited to 15 while for $EGNA_{EE}$ algorithm, the size of the individuals was 300 and the execution was stopped when the 300th generation was reached (looking for an execution time of the sequential algorithm close to 30 minutes).

The results obtained for $EGNA_{BIC}$ algorithm can be seen in Fig. 33 and 34 and table XV.

It can be observed that the parallel algorithm represents a great improvement over the sequential version. In particular, the results obtained with the second parallel version are really good: for six or less nodes, efficiency is always over 0.8, being even more than 1 when 2,3 or 4 nodes are used. It should be remembered that the program is executed in different machines (using MPI) and therefore, depending on the characteristics of the algorithm and the available architecture, results like

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

	Sphere model				*Rosenbrock gener.*			
	1st version		*2nd version*		*1st version*		*2nd version*	
Nod	**secs**	**eff**	**secs**	**eff**	**secs**	**eff**	**secs**	**eff**
seq.	1850	1.00	1850	1.00	1879	1.00	1879	1.00
2	877	0.53	762	0.61	897	0.52	785	0.60
3	807	0.38	568	0.54	822	0.38	580	0.54
4	790	0.29	500	0.46	805	0.29	510	0.46
5	789	0.23	465	0.40	804	0.23	478	0.39
6	787	0.20	451	0.34	801	0.20	458	0.34
7	782	0.17	434	0.30	796	0.17	443	0.30
8	790	0.15	433	0.27	805	0.15	443	0.27
9	797	0.13	436	0.24	809	0.13	446	0.23
10	812	0.11	457	0.20	825	0.11	468	0.20

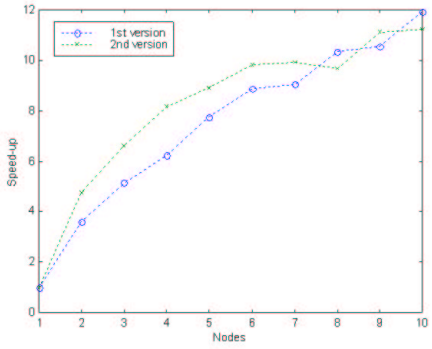

Fig. 33. Speed-up for $EGNA_{BIC}$ algorithm with the *Sphere model* problem

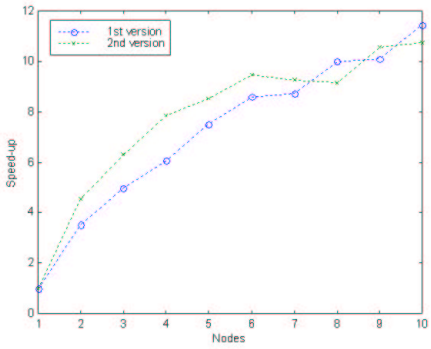

Fig. 34. Speed-up for $EGNA_{BIC}$ algorithm with the *Rosenbrock generalized* problem

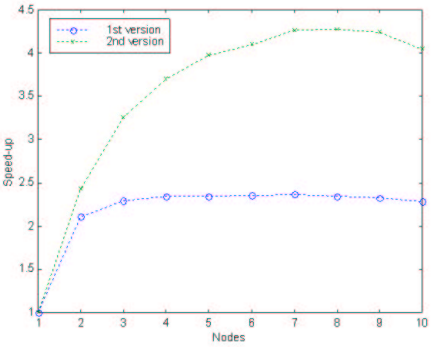

Fig. 35. Speed-up for $EGNA_{EE}$ algorithm with the *Sphere model* problem

these are not unusual. On the other hand, it can be seen that the difference on speed-up between the first and second version increases with the number of nodes up to four. After that number, difference decreases becoming similar and even better, in some cases, for the first version of the parallel algorithm. This is due to the work distribution -depends on the number of workers(nodes)- and the communication-computation relation.

Concerning the parallelization of the $EGNA_{EE}$ algorithm, it can be seen that the results are not as good as in previous algorithms. In the explanation given for this algorithm, the learning phase is not as important computationally as in the other algorithms, and therefore the improvement obtained can not be excellent: for the first version, there is nearly naught time reduction when three or more nodes are used. However, when combining the parallelization of learning with the parallelization of the sampling phase -the second parallel version of $EGNA_{EE}$-, the results are better, reaching a reasonable speed-up. It can also be seen -for this second parallel version- that the best results are obtained when seven or eight nodes are used, decreasing the efficiency with nine or more. It is important to keep in mind that a good balance between communication and computation must be maintained, because if too many nodes are used (last cases) the process of work distribution and recollection supposes an important delay.

These results can be seen in Fig. 35 and 36 and table XVI.

2) quality-related dimension: As said before, the proposed parallel version follows exactly the same steps as the sequential version, so it is interesting to compare the objective function values that are obtained by the different parallel versions over the time. Fig. 37, 38, 39 and 40 show the

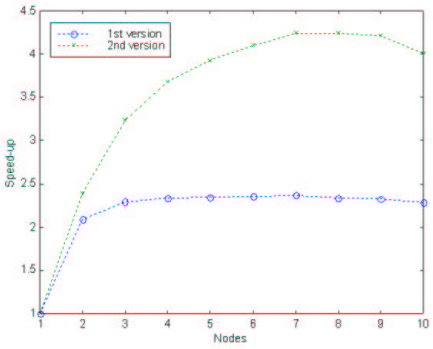

Fig. 36. Speed-up for $EGNA_{EE}$ algorithm with the *Rosenbrock generalized* problem

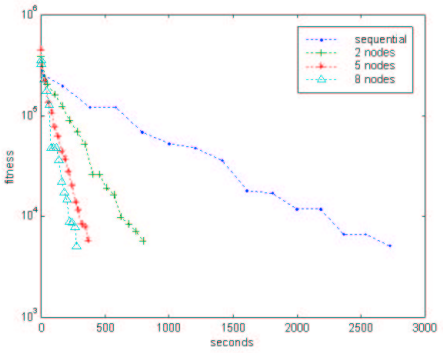

Fig. 39. Solution produced by the first version of the $EGNA_{BIC}$ algorithm throughout the time with the *Rosenbrock generalized* problem

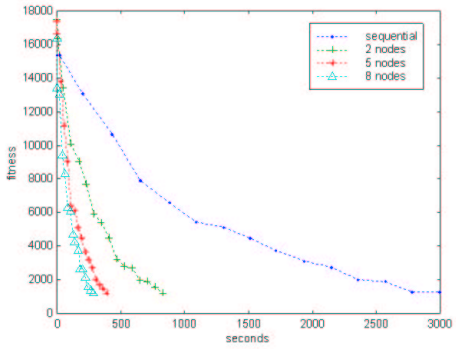

Fig. 37. Solution produced by the first version of the $EGNA_{BIC}$ algorithm throughout the time with the *Sphere model* problem

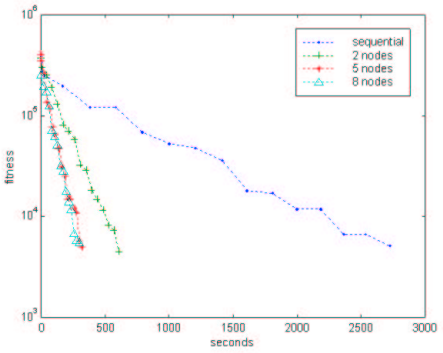

Fig. 40. Solution produced by the second version of the $EGNA_{BIC}$ algorithm throughout the time with the *Rosenbrock generalized* problem

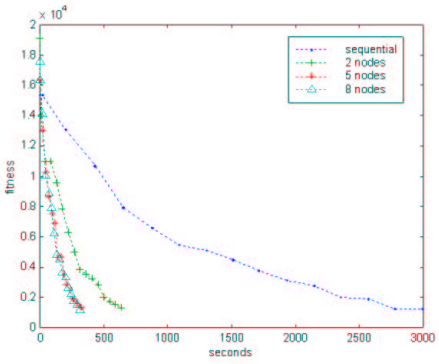

Fig. 38. Solution produced by the second version of the $EGNA_{BIC}$ algorithm throughout the time with the *Sphere model* problem

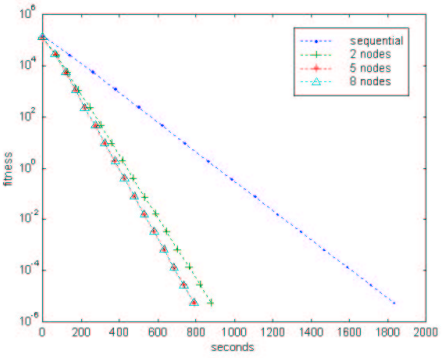

Fig. 41. Solution produced by the first version of the $EGNA_{EE}$ algorithm throughout the time with the *Sphere model* problem

difference of the sequential version compared to the parallel ones for the $EGNA_{BIC}$ algorithm, where better objective function values can be obtained in a short time.

Concerning to $EGNA_{EE}$ algorithm, take note that in the first version, the difference when three or more nodes are used is really small, being slightly more appreciable for the second parallel version (see Fig. 41, 42, 43 and 44.

3) performance-related dimension: The last results are those obtained from the execution times of manager and workers. These show how much work is executed by each type of node (in this case manager or worker) and can also be used to verify whether the load is equally distributed between the workers. Fig. XVII, XVIII, XIX and XX show the detailed information obtained from the execution of the

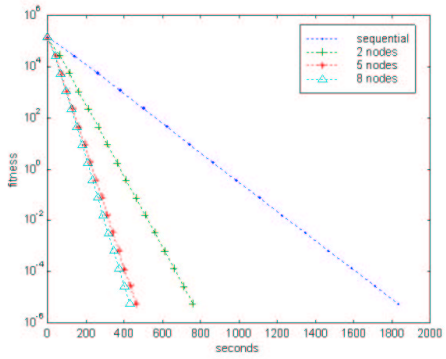

Fig. 42. Solution produced by the second version of the $EGNA_{EE}$ algorithm throughout the time with the *Sphere model* problem

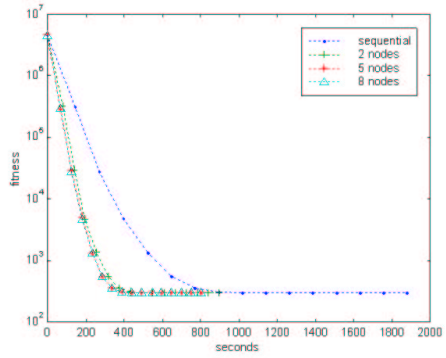

Fig. 43. Solution produced by the first version of the $EGNA_{EE}$ algorithm throughout the time with the *Rosenbrock generalized* problem

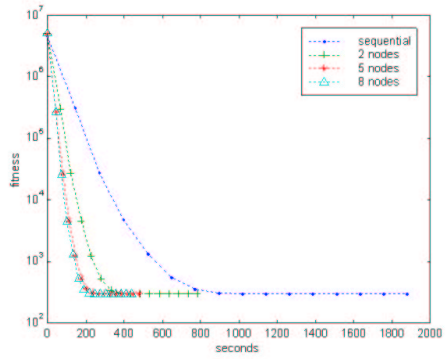

Fig. 44. Solution produced by the second version of the $EGNA_{EE}$ algorithm throughout the time with the *Rosenbrock generalized* problem

$EGNA_{BIC}$ algorithm with the two problems (*Sphere* model and *Rosenbrock generalized*). In most of the cases, manager and workers require a similar amount of computation time (remember that the manager carries out the sequential steps as well as the communications) and, if we focus on the workers, we see that load is quite well balanced. Particularly, it can be observed that the first workers consume generally more

TABLE XVII
ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{BIC}$ FIRST VERSION WITH *Sphere model* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
808	787	-	-	-	-	-	-	-	-
567	543	538	-	-	-	-	-	-	-
445	420	419	417	-	-	-	-	-	-
369	346	343	343	341	-	-	-	-	-
318	298	295	294	294	294	-	-	-	-
293	273	272	268	246	246	245	-	-	-
269	249	249	246	228	225	219	218	-	-
247	227	225	221	218	218	199	197	197	-
221	203	199	201	200	198	197	198	199	196

TABLE XVIII
ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{BIC}$ SECOND VERSION WITH *Sphere model* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
612	581	-	-	-	-	-	-	-	-
438	413	401	-	-	-	-	-	-	-
348	324	316	314	-	-	-	-	-	-
295	273	265	263	254	-	-	-	-	-
260	242	234	233	228	224	-	-	-	-
235	217	213	208	200	198	194	-	-	-
208	196	189	188	181	179	175	176	-	-
198	181	179	179	172	168	163	164	157	-
181	166	162	162	157	154	152	152	147	146

computational time than the others; this is mainly due to the work division: it is not always possible to evenly distribute the variables among the workers, so some of them (the first one's) may be receiving a bit more work.

For the first version of the $EGNA_{EE}$ algorithm, the behavior is different. As the learning phase does not take much more than half of the total execution time, it can be seen that the work distribution between the manager and the workers is very uneven, becoming more evident as the number of workers increases. In the second parallel version, more code has been parallelized, so the manager-worker duties are better distributed and therefore better results are obtained. Even so, the differences between the execution times of manager and workers are greater than in the previous parallel algorithms.

TABLE XIX

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{BIC}$ FIRST VERSION WITH *Rosenbrock generalized* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
769	746	-	-	-	-	-	-	-	-
533	508	509	-	-	-	-	-	-	-
422	399	396	396	-	-	-	-	-	-
352	330	327	330	326	-	-	-	-	-
305	285	282	285	285	284	-	-	-	-
283	261	261	262	233	237	233	-	-	-
260	237	236	240	213	215	211	214	-	-
233	211	209	213	212	214	187	190	189	-
209	191	190	194	190	193	190	191	188	192

TABLE XX

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EBNA_{BIC}$ SECOND VERSION WITH *Rosenbrock generalized* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
588	556	-	-	-	-	-	-	-	-
427	403	391	-	-	-	-	-	-	-
335	312	304	300	-	-	-	-	-	-
284	259	256	252	244	-	-	-	-	-
249	229	222	222	216	214	-	-	-	-
229	207	203	200	194	194	189	-	-	-
205	189	185	184	177	176	171	171	-	-
188	175	170	169	162	161	155	155	151	-
176	163	158	159	154	153	150	149	145	143

TABLE XXI

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{EE}$ FIRST VERSION WITH *Sphere model* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
861	167	-	-	-	-	-	-	-	-
795	116	116	-	-	-	-	-	-	-
776	92	91	91	-	-	-	-	-	-
765	80	76	76	75	-	-	-	-	-
751	75	66	66	65	65	-	-	-	-
741	69	62	58	59	59	57	-	-	-
738	62	55	57	51	52	52	52	-	-
737	58	52	52	47	47	48	48	47	-
733	60	47	48	43	44	45	44	45	45

TABLE XXII

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{EE}$ SECOND VERSION WITH *Sphere model* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
741	432	-	-	-	-	-	-	-	-
553	277	276	-	-	-	-	-	-	-
420	213	213	213	-	-	-	-	-	-
433	179	174	176	174	-	-	-	-	-
403	160	148	148	148	148	-	-	-	-
378	141	134	131	130	131	128	-	-	-
362	126	120	120	115	115	115	115	-	-
356	116	108	108	104	104	104	105	105	-
343	113	99	99	95	95	96	96	96	96

TABLE XXIII

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{EE}$ FIRST VERSION WITH *Rosenbrock generalized* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
880	173	-	-	-	-	-	-	-	-
810	121	120	-	-	-	-	-	-	-
790	94	94	94	-	-	-	-	-	-
777	83	78	79	79	-	-	-	-	-
765	78	68	68	68	67	-	-	-	-
754	71	65	61	60	61	59	-	-	-
750	65	57	59	53	53	53	54	-	-
715	61	54	53	49	49	48	50	49	-
748	63	50	50	45	45	45	46	46	46

TABLE XXIV

ALGORITHM PERFORMANCE-RELATED EXPERIMENTAL RESULTS FOR $EGNA_{EE}$ SECOND VERSION WITH *Rosenbrock generalized* PROBLEM. EXECUTION TIMES FOR MANAGER AND WORKERS.

M.	***Wkrs.***								
	1	2	3	4	5	6	7	8	9
768	398	-	-	-	-	-	-	-	-
566	273	272	-	-	-	-	-	-	-
491	209	210	211	-	-	-	-	-	-
445	176	171	172	171	-	-	-	-	-
410	157	146	146	146	145	-	-	-	-
389	138	132	128	128	128	126	-	-	-
373	124	118	119	113	113	113	113	-	-
366	113	107	107	102	102	102	102	102	-
355	110	96	98	94	93	94	94	94	95

VI. Conclusions 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 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 creation of new individuals once the structure has been learnt.

Different parallel approaches have been developed for each sequential algorithm, mixing MPI and thread techniques. The experiments have been made using a cluster of ten two-processor computers, changing the number of nodes from two to ten (MPI communication) where in each node two threads have been created.

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 in a practical situation not only the parallelization of the learning or sampling phases has to be taken into account but, the parallelization of the objective function. Moreover, if an accurate and efficient solution is wanted, it would be necessary to find a specific parallelization of the objective function in used.

Finally, we have began a deeper study of the advantage that supposes mixing MPI and threads, measuring different times and characteristics (communication times, synchronization, work distribution) of the resulting algorithm combined with more complex problems.

References

[1] E. Cantú-Paz, *Efficient and accurate parallel genetic algorithms*. Kluwer Academic Publishers, 2000.

[2] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[3] 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.

[4] J. Ocenasek and J. Schwarz, "The parallel bayesian optimization algorithm," in *Proceedings of the European Symposium on Computational Intelligence*, pp. 61–67, 2000.

[5] J. Ocenasek and J. Schwarz, "The distributed bayesian optimization algorithm for combinatorial optimization," in *EUROGEN - Evolutionary Methods for Design, Optimisation and Control, CIMNE*, 2001.

[6] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Orlando FL* (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds.), vol. 1, (San Francisco, CA), pp. 525–532, Morgan Kaufmann Publishers, 1999.

[7] J. Ocenasek, J. Schwarz, and M. Pelikan, "Design of multithreaded estimation of distribution algorithms," in *Lecture Notes in Computer Science 2724: Genetic and Evolutionary Computation - GECCO 2003*, pp. 1247–1258, Springer, 2003.

[8] C. W. Ahn, D. E. Goldberg, and R. Ramakrishna, "Multiple-deme parallel estimation of distribution algorithms: Basic framework and application," tech. rep., Kwang-Ju institute of Science and Technology (K-JIST), 2003.

[9] M. P. I. Forum, "MPI: A message-passing interface standard," tech. rep., International Journal of Supercomputer Applications, Los Angeles, 1994.

[10] D. R. Butenhof, *Programming with POSIX Threads*. Addison-Wesley Professional Computing Series, 1997.

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

[12] A. A. Zhigljavsky, *Theory of Global Random Search*. Kluwer Academic Publishers, 1991.

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

[14] M. Sebag and A. Ducoulombier, *Extending Population-Based Incremental Learning to Continuous Search Spaces*. Parallel Problem Solving from Nature - PPSN V , 418–427, Berlin, Springer-Verlag, 1998.

[15] J. S. De Bonet, C. L. Isbell, and P. Viola, "MIMIC: Finding optima by estimating probability densities," in *Advances in Neural Information Processing Systems*, vol. 9, M. Mozer, M. Jordan and Th. Petsche eds., 1997.

[16] P. Larrañaga, R. Etxeberria, J. Lozano, and J. Peña, "Optimization by learning and simulation of Bayesian and Gaussian networks," Tech. Rep. KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.

[17] P. Larrañaga, R. Etxeberria, J. Lozano, and J. Peña, "Optimization in continuous domains by learning and simulation of Gaussian networks," in *Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the 2000 Genetic and Evolutionary Computation Conference, GECCO 2000*, (Las Vegas, Nevada, USA), pp. 201–204, 2000.

[18] R. Etxeberria and P. Larrañaga, "Global optimization with Bayesian networks," in *Special Session on Distributions and Evolutionary Optimization*, pp. 332–339, II Symposium on Artificial Intelligence, CIMAF99, 1999.

[19] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña, "Combinatorial optimization by learning and simulation of Bayesian networks.," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI 2000*, (Stanford, CA, USA), pp. 343–352, 2000.

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

[21] M. Pelikan and D. E. Goldberg, "Hierarchical problem solving and the Bayesian optimization algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference* (D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H.-G. Beyer, eds.), vol. 1, (San Francisco, CA), pp. 267–274, Morgan Kaufmann Publishers, 2000.

[22] M. Pelikan and D. E. Goldberg, "Research on the Bayesian optimization algorithm," in *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program* (A. Wu, ed.), vol. 1, pp. 212–215, 2000.

[23] M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002. Also IlliGAL Report No. 99018.

[24] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*. New York: Springer-Verlag, 1997.

[25] R. Shachter and C. Kenley, "Gaussian influence diagrams," *Management Science*, vol. 35, pp. 527–550, 1989.

[26] G. Schwarz, "Estimating the dimension of a model," *Annals of Statistics*, vol. 7, no. 2, pp. 461–464, 1978.

[27] P. Spirtes, C. Glymour, and R. Scheines, "An algorithm for fast recovery of sparse causal graphs," *Social Science Computing Reviews*, vol. 9, pp. 62–72, 1991.

[28] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search*. Lecture Notes in Statistics 81, Springer-Verlag., 1993.

[29] P. W. F. Smith and J. Whittaker, "Edge exclusion tests for graphical gaussian models," in *Learning in Graphical Models*, pp. 555–574, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[30] J. Whittaker, *Graphical models in applied multivariate statistics*. John Wiley and Sons, 1990.

[31] M. Henrion, "Propagating uncertainty in Bayesian networks by probabilistic logic sampling," *Uncertainty in Artificial Intelligence*, vol. 2, pp. 149–163, 1988. J.F. Lemmer and L.N. Kanal eds., North-Holland, Amsterdam.

[32] B. D. Ripley, *Stochastic Simulation*. John Wiley and Sons, 1987.

[33] R. Salomon, "Evolutionary algorithms and gradient search: similarities and differences," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 2, pp. 45–55, 1998.

[34] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, pp. 175–184, 1960.