Universidad del País Vasco
Euskal Herriko Unibertsitatea
The University of the Basque Country

Parallel implementation of Estimation of Distribution Algorithms based on probabilistic graphical models. Application to chemical calibration models.

by

Alexander Mendiburu-Alberro

Supervisors: José A. Lozano Alonso
José Miguel-Alonso

Dissertation submitted to the Department of Computer Architecture and Technology in partial fulfillment of the requirements for the PhD degree in Computer Science.

Donostia - San Sebastián, January 2006

Acknowledgements

This dissertation completes a first step in my research life. During the years I have been working on this dissertation, I have had two excellent advisors, Dr. Jose A. Lozano and Prof. José Miguel-Alonso, that have taught me what research is and how to proceed in this area. I have been really lucky, because instead of having a single advisor, I have been assisted by two friends, one at each side: Thank you very much.

I would like to thank Prof. Xin Yao, for inviting me to complete a stay as a visitor in CERCIA, at the University of Birmingham. It has been a remarkable experience, that has allowed me to contact with different researchers. I would also want to thank Dr. Carlos Ubide and Dr. Miren Ostra for providing the problem and datasets described in Chapter 5.

Finally, I am very grateful to my research group, Intelligent Systems Group, for accepting me in such extraordinary research-family.

The following lines are devoted to several persons that have encouraged me during my life and particularly during my years as PhD candidate. Please let me use my mother language (Basque) to complete these lines.

- Jose eta Joseani, hain lan talde bikaina osatzeagatik eta zuen adiskidetasunagatik,
- Endika, tesiaren lehenengo pausuetan laguntzeagatik eta lagun aparta delako,
- Nire bidai lagunari Txus. Gure kafetxo-elkarrizketak benetan aberasgarriak izan dira eta,
- Birminghameko unibertsitatean egin nituen lagunei, bereziki Roc eta Will. Beraiei esker egonaldia oso esperientzia polita izan delako,
- Nire familia eta lagunei, nire munduko zati garrantzitsu bat,
- Nire gurasoei, eman didaten heziketagatik eta dena zor diedalako,
- Anari, nire benetako supporter-a. Eskerrik asko ni hain ondo zaintzeagatik. Planeta honetan jarraitzeko arrazoi ederra,
- ...eta zuri amatxo, egunero nirekin zaudelako.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Evolutionary Algorithms (EAs) comprise several families of computational techniques that share a common feature: they are inspired by natural evolution of species. Based on this idea, EAs use a structure known as individual, composed by a set of variables (genes) that can take different values. The number of variables and the range of values each variable can take are problem-dependent and the objective is to find a solution in the form of a good individual using different mechanisms.

EAs create a initial set of individuals (population) and evolve some of the individuals using different breeding operators. As the number of generations increases, the quality of the individuals should also improve, approaching towards the best representation (solution) for the problem.

Among the different techniques used in EAs, we focus on Estimation of Distribution Algorithms (EDAs). Their main feature is the use of probability models to represent the characteristics of the population. The goal is to extract information about the possible relations between the variables (genes) that conform the individual.

According to this general framework, several EDA-based approaches have been proposed in the last years. These algorithms are able to successfully solve a wide range of problems [103, 143, 112]. However, the execution of an EDA may take a considerable amount of time. Two are the main reasons: (1) the EDAs that generally obtain the best results are those that consider multiple dependencies between the variables and, hence, use "complex" probabilistic models (for instance, probabilistic graphical models), and (2) like other EAs, the use of populations (sets of individuals) implies the need to evaluate each individual according to the fitness function related to the particular problem we are dealing with. This evaluation might require an important amount of time when the evaluation function is complex.

This work was proposed with the aim of improving some EDAs from the point of view of execution time. That is, our goal is to design parallel implementations of different EDAs that would efficiently use a set of processors (instead of only one), thus reducing the execution time. Additionally, we want to do so while maintaining the behavior of the sequential counterparts.

In order to design a good parallel approach, characteristics like performance and scalability are always desired. It is important to develop a parallel version that is able to efficiently use all the available processors, reducing the execution time as much as possible. The goal is usually to obtain linear speed ups (or even super-linear), trying to accelerate the program by a factor of p, being p the number of processors available. In addition, it is also interesting

to maintain a good scalability, that is, as the number of processors increases, the parallel program should be able to maintain good speed up values.

Related to the tools available to design a parallel implementation, we can cite two main parallel programming paradigms: threads and Message Passing Interface (MPI). Threads provide a way to split a program in different tasks that can be executed in a parallel way. This is mainly intended to be used in multiprocessor machines, in which the tasks can be distributed in different processors but still share common memory structures.

The other paradigm, MPI, can be described as a communication layer that allows applications to interchange data and synchronize using messages. This communication can be established between different machines (connected by a network), and also in the same machine.

Trying to evaluate the advantages that each paradigm can bring to EDAs, we decided to implement our parallel programs using a two-level scheme. Communication at the first level is supposed to be between different machines; therefore, it is performed using MPI. The second level is organized around a collection of cooperating threads (running on different processors in the same machine), which communicate via shared data structures. Experiments with different paradigm combinations have been carried out, showing that, in general, the designed parallel approaches present good efficiency and scalability properties. In addition, some additional research has been done, showing that a cluster of computers is a good platform for parallel EDAs in terms of performance and price.

The availability of parallel implementations of EDAs, together with cheap parallel computers, give researchers a new chance to apply this family of algorithms to complex problems, being able to evaluate different approaches in an affordable time. In particular, in this dissertation we have applied two parallel EDAs to solve a quantitative chemistry problem. The goal was to find a calibration model that, once trained with known data, would predict unknown initial concentration values for the species that took part in a chemical reaction. Running the designed parallel EDAs in clusters of PC-class computers has allowed us to study the problem from different points of view, getting several valid models.

Overview of the dissertation

This dissertation has been structured in six chapters. After this introduction, in Chapter 2 a general overview of EDAs is provided. The aim of this chapter is to point out the main characteristics of this family of algorithms based on probabilistic models. Even if our work has focused on a particular subset of algorithms, we have written the chapter including many different EDA approaches that have been developed in the last years. In this way, anyone interested in EDAs will obtain information and references to familiarize with this kind of algorithms.

Chapter 3 is devoted to the parallel implementations of different EDAs. We have focused on those algorithms that use probabilistic graphical models (in particular, Bayesian and Gaussian networks) in both discrete and continuous domains.

In Chapter 4, some of the proposals and experiments presented in Chapter 3 are extended. Different executions were completed looking for the best combination of programming paradigms and architectures, using a complex problem (Feature Subset Selection for classification), and measuring its efficiency and scalability.

Chapter 5 presents the results obtained by using parallel implementations of different EDAs to solve a calibration problem in a quantitative chemistry context. Different approaches have been tested, obtaining good models from the point of view of accuracy.

Finally, in Chapter 6, the main contributions of this dissertation are summarized, together with some open research lines that we plan to undertake in the future.

Chapter 2

A review of Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) are a set of new promising optimization techniques that belong to the family of Evolutionary Algorithms (EAs). The main characteristic of these algorithms is the use of probabilistic models to identify relations between variables. Before presenting the parallel approaches designed for some of these algorithms, we think interesting to present in this chapter a general and up-to-date overview of the different EDAs that have been presented in the last years.

It is out of the scope of this chapter to explain in detail the characteristics of each algorithm, and therefore several references are provided to guide the interested reader towards more detailed information. However, the algorithms parallelized in the next chapter (EBNA and EGNA) receive more attention.

2.1 Introduction

The ubiquitous presence of high-performance, PC-type computers, has encouraged the design and application of non-trivial algorithms to solve different kinds of complex optimization problems. Some of those problems can be solved via an exhaustive search of all possible solutions, but in most cases this brute force approach is unaffordable. In those situations, heuristic methods are often used (deterministic or non deterministic), searching into the space of promising solutions following a particular scheme. Several heuristic approaches have been designed to find good solutions for a specific problem, or have been presented as a general framework adaptable to many different situations.

Among this second group (general designs), there is a family of algorithms that has been widely used in the last decades: Evolutionary Algorithms (EAs). This family comprises, as main paradigms, Genetic Algorithms (GAs) [67, 86], Evolution Strategies [161], Evolutionary Programming [58] and Genetic Programming [98].

The main characteristic of these algorithms is that they use techniques inspired by the natural evolution of the species. In nature, species change across time; individuals evolve, adapting to the characteristics of their environment. This evolution leads to individuals with better characteristics.

In the same way, this idea can be translated to the world of computation, using the similar concepts:

Individual: Represents a possible solution for a problem to be solved. Each individual has a set of characteristics (genes) and a fitness value that denotes the quality of the individual.

Population: In order to look for the best solution a group of several individuals is managed. An initial population is created randomly, and will change across time, evolving towards members with different (and probably better) characteristics.

Breeding: Trying to emulate the breeding process present in the nature world, several operators can be used,mixing different individuals (crossover) or even changing a particular one (mutation), to obtain new individuals (expected to be better than the previous ones).

In the last decade, GAs have been widely used to solve different problems, improving in many cases the results obtained by previous algorithms. However, this kind of algorithms have a large number of parameters (for example, those that control the creation of new individuals) that need to be correctly tuned in order to obtain good results. Generally, only experienced users can do this correctly and, moreover, the task of selecting the best choice of values for all these parameters has been suggested to constitute itself an optimization problem [74]. In addition, GAs show a poor performance in some problems (deceptive and separable problems) in which the existing operators of crossover and mutation do not guarantee that better individuals will be obtained changing or combining existing ones.

Some authors [86] have pointed out that making use of the relations between genes can be useful to drive a more "intelligent" search through the solution space. This concept, together with the limitations of GAs, motivated the creation of a new type of algorithms grouped under the name of Estimation of Distribution Algorithms (EDAs).

EDAs were introduced in the field of Evolutionary Computation in [128], although similar approaches can be previously found in [224]. In EDAs there are neither crossover nor mutation operators. Instead, the new population of individuals is sampled from a probability distribution, which is estimated from a database that contains the selected individuals from the previous generation. Thus, the interrelations between the different variables that represent the individuals are explicitly expressed through the joint probability distribution associated with the individuals selected at each generation.

2.2 Introduction to EDAs. A simple example

In order to understand the behavior of this heuristic, a simple example will be presented. Suppose that we want to maximize the n-dimensional *OneMax* function, defined as:

$$h(\boldsymbol{x}) = \sum_{i=1}^{n} x_i \tag{2.1}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{0, 1\}$.

For this example, we will use a six-dimensional space, so the best fitness value (six in this example) will be obtained when all variables are equal to 1.

An initial and important step is to choose a good representation of the problems: how we are going to model the individuals in terms of variables and values. For this example, we will

TABLE 2.1: THE INITIAL POPULATION, D_0.

	X_1	X_2	X_3	X_4	X_5	X_6	$h(\boldsymbol{x})$
1	1	0	1	0	1	0	3
2	0	1	0	0	1	0	2
3	0	0	0	1	0	0	1
4	1	1	1	0	0	1	4
5	0	0	0	0	0	1	1
6	1	1	0	0	1	1	4
7	0	1	1	1	1	1	5
8	0	0	0	1	0	0	1
9	1	1	0	1	0	0	3
10	1	0	1	0	0	0	2
11	1	0	0	1	1	1	4
12	1	1	0	0	0	1	3
13	1	0	1	0	0	0	2
14	0	0	0	0	1	1	2
15	0	1	1	1	1	1	5
16	0	0	0	1	0	0	1
17	1	1	1	1	1	0	5
18	0	1	0	1	1	0	3
19	1	0	1	1	1	1	5
20	1	0	1	1	0	0	3

define an individual as a vector of six components, (X_1,...,X_6), where each characteristic or gene (X_i) can take one of two values (0 or 1) and represents the variable X_i of the *OneMax* function.

As the first step, an initial population will be generated randomly by sampling the following probability distribution: $p_0(\boldsymbol{x}) = \prod_{i=1}^{6} p_0(x_i)$, where $p_0(X_i = 1) = 0.5$ for $i = 1, \ldots, 6$. This means that the probability distribution from which we are sampling is factorized as a product of six univariate marginal distributions, each following a Bernoulli distribution with parameter value equal to 0.5. Table 2.1 shows the initial population as well as the fitness value of each individual (represented as $h(\boldsymbol{x})$). This value is obtained counting the number of variables whose value is equal to 1.

In the second step, we select some of the individuals from the initial population D_0. This can be done using the different standard methods presented in EAs. For this example, we use truncation selection, choosing the best half of the population (see Table 2.2).

Now that we have selected some individuals, in the third step their characteristics will be expressed by means of a joint probability distribution. Among the different options that can be used (described later in this chapter), we will use the simplest one, considering that there are no dependencies between variables. It can be mathematically expressed as:

$$p_1(\boldsymbol{x}) = p_1(x_1, \ldots, x_6) = \prod_{i=1}^{6} p(x_i | D_0^{Se}) \tag{2.2}$$

With this approach, we only need six parameters to specify the model (one per variable). These six parameters will be estimated from selected data (D_0^{Se}) by means of their corresponding relative frequency, $\hat{p}(X_i = 1 | D_0^{Se})$.

TABLE 2.2: THE SELECTED INDIVIDUALS, D_0^{Se}, FROM THE INITIAL POPULATION.

	X_1	X_2	X_3	X_4	X_5	X_6
1	1	0	1	0	1	0
4	1	1	1	0	0	1
6	1	1	0	0	1	1
7	0	1	1	1	1	1
11	1	0	0	1	1	1
12	1	1	0	0	0	1
15	0	1	1	1	1	1
17	1	1	1	1	1	0
18	0	1	0	1	1	0
19	1	0	1	1	1	1

Taking into account the selected individuals (Table 2.2), the values of the parameters are:

$$\hat{p}(X_1 = 1|D_0^{Se}) = 0.7 \qquad \hat{p}(X_2 = 1|D_0^{Se}) = 0.7 \qquad \hat{p}(X_3 = 1|D_0^{Se}) = 0.6 \tag{2.3}$$

$$\hat{p}(X_4 = 1|D_0^{Se}) = 0.6 \qquad \hat{p}(X_5 = 1|D_0^{Se}) = 0.8 \qquad \hat{p}(X_6 = 1|D_0^{Se}) = 0.7 \tag{2.4}$$

Finally, once the probability distribution has been obtained, the fourth step is carried out: obtain new individuals. The new population will be created sampling the probability distribution. In this example 20 new individuals have been generated.

Table 2.3 shows the new population, which comprises only the recently sampled individuals. However, it must be noted that when creating the new population different criteria can be taken into account: use only the recently sampled individuals (like in this example), select the best individuals among the sampled and the present individuals, select some of the new individuals, and so on.

These last three steps (selection of the individuals, induction of the probability model and sampling of the new individuals) will be repeated until a particular stopping criterion is fulfilled. For example, a number of iterations (usually called generations) or a predetermined fitness value.

Obviously, the example used through these lines is a very simple version of EDAs, but it is still valid to show the general behavior of these algorithms. Figure 2.1 presents a common outline for all EDAs.

Once the general framework has been explained, a review of different EDAs is presented in the following sections, classified using two main criteria: on the one hand, EDAs can be applied to combinatorial optimization problems (discrete domain) as well as to continuous domains, and on the other hand, different probability models can be used to represent the dependencies between the variables that constitute the individuals.

2.3 EDA approaches in the discrete domain

In this section several EDA approaches for combinatorial optimization will be presented, introducing the main characteristics of each algorithm.

Table 2.3: The population of the first generation, D_1.

	X_1	X_2	X_3	X_4	X_5	X_6	$h(\boldsymbol{x})$
1	1	1	1	1	1	1	6
2	1	0	1	0	1	1	4
3	1	1	1	1	1	0	5
4	0	1	0	1	1	1	4
5	1	1	1	1	0	1	5
6	1	0	0	1	1	1	4
7	0	1	0	1	1	0	3
8	1	1	1	0	1	0	4
9	1	1	1	0	0	1	4
10	1	0	0	1	1	1	4
11	1	1	0	0	1	1	4
12	1	0	1	1	1	0	4
13	0	1	1	0	1	1	4
14	0	1	1	1	1	0	4
15	0	1	1	1	1	1	5
16	0	1	1	0	1	1	4
17	1	1	1	1	1	0	5
18	0	1	0	0	1	0	2
19	0	0	1	1	0	1	3
20	1	1	0	1	1	1	5

Step 1. Generate the first population D_0 of M individuals and evaluate each of them.
Step 2. **repeat** in each generation l
Step 3. Select N individuals (D_l^{Se}) from the D_l population following a selection method
Step 4. Induce from D_l^{Se} an n (size of the individual) dimensional probability model that shows the interdependencies between variables
Step 5. Generate a new population D_{l+1} of M individuals based on the sampling of the probability distribution $p_l(\boldsymbol{x})$ learnt in the previous step
Step 6. **if** an exit condition is fulfilled **stop**
else go to Step 2

Figure 2.1: Pseudo-code for EDAs.

Algorithms have been grouped according to the way dependencies between variables are considered: all variables are independent, pairwise dependencies, or multiple dependencies.

2.3.1 Without dependencies

All the models that belong to this category consider all variables as independent. Therefore, the joint probability distribution is factorized as a product of univariate and independent probability distributions. That is, $p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i)$.

UMDA

Univariate Marginal Distribution Algorithm (UMDA). Introduced in [123], this algorithm uses the simplest way to estimate the joint probability distribution:

$$p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se}) = \prod_{i=1}^{n} p_l(x_i) \tag{2.5}$$

where each univariate marginal distribution is estimated from marginal frequencies:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i|D_{l-1}^{Se})}{N} \tag{2.6}$$

being

$$\delta_j(X_i = x_i|D_{l-1}^{Se}) = \begin{cases} 1 & \text{if in the } j^{th} \text{ case of } D_{l-1}^{Se},\ X_i = x_i \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

UMDA has been successfully applied to different problems: search in a classifier system [164], Feature Subset Selection [6, 21], learning of Bayesian networks from data [20, 166], optimization of a composite video processing system [7], or to solve some linear and combinatorial problems using Laplace correction [142].

Other works focus on the behavior of the algorithm, performing a mathematical analysis of UMDA [126, 117], studying its convergence when UMDA is used to maximize a number of pseudo-boolean functions [70], or analyzing the genetic drift phenomenon [88, 89, 191].

Finally, several modifications have also been introduced in UMDA trying to improve its performance: modifications on the simulation phase [176, 177], use of a repair method for solving constraint satisfaction problems [76], adaptive population sizing [87], use of memory schemes for dynamic optimization problems [215], or introducing the bitwise mutation operator [77].

BSC

This approach, Bit-Based Simulated Crossover (BSC) [201] uses the value of the fitness function of the selected individuals to estimate each marginal distribution:

$$p_l(x_i) = \frac{\sum_{\{\boldsymbol{x}|\delta_j(X_i=x_i|D_{l-1}^{Se})=1\}} ef(\boldsymbol{x})}{\sum_{\{\boldsymbol{x}\epsilon D_{l-1}^{Se}\}} ef(\boldsymbol{x})} \tag{2.8}$$

where function δ_j maintains the meaning expressed in Equation 2.6.

In Equation 2.8, the numerator represents the sum of the evaluation function values of the individuals with value x_i in the variable X_i, and the denominator is the sum of evaluation values of the selected individuals.

This algorithm has been applied to problems such as feature subsect selection [94, 91] and partition clustering tasks [168].

PBIL

Population Based Incremental Learning (PBIL) [9, 10] uses a probability vector to represent the characteristics of the population:

$$p_l(\boldsymbol{x}) = (p_l(x_1), \ldots, p_l(x_i), \ldots, p_l(x_n)) \tag{2.9}$$

where $p_l(x_i)$ refers to the probability of obtaining a value of 1 in the i^{th} variable of the l^{th} population.

The vector is initialized using the first population, and then it is used to sample a new set of M individuals. From this set, only the best N individuals are selected. We denote them by:

$$\boldsymbol{x}^l_{1:M}, \ldots, \boldsymbol{x}^l_{i:M}, \ldots, \boldsymbol{x}^l_{N:M} \tag{2.10}$$

Based on the following (Hebbian inspired) rule, the probability vector is updated:

$$p_{l+1}(\boldsymbol{x}) = (1-\alpha)p_l(\boldsymbol{x}) + \alpha\frac{1}{N}\sum_{k=1}^{N}\boldsymbol{x}^l_{k:M} \tag{2.11}$$

where $\alpha \in (0, 1]$ is a parameter of the algorithm (the reader may note that when $\alpha = 1$, this algorithm performs as UMDA).

The following population will be sampled from this new (updated) probability vector. In contrast to the general EDA behavior, it must be noted that this algorithm uses the probability vector of the previous generation in addition to the recently sampled individuals to obtain the new probability vector.

PBIL has been applied to different problems, such as: optimization of parameters of a solution in the field of tactical driving [200], search for optimal weights in a neural network structure [118, 62], classifier selection [171], optimization of parameters for the simple supply chain model [71], or learning of Bayesian networks [20].

Some theoretical studies of PBIL have been completed in [85, 68].

Finally, there are some works that use some characteristics of PBIL or even modify parts of the algorithm. In [110], the Statistical and Inductive Tree Based Evolution algorithm is presented. This approach mixes ideas from PBIL (probability vector) with inductive decision trees. In general terms, it works as follows: starting with a randomly created population, individuals are split into three groups (best, mediocre, and bad) and Induction of Decision Trees is used to induct a decision tree, extracting the set of equivalent rules. Then, PBIL is used together with the rules to sample new individuals. The process is repeated until some termination criteria is fulfilled.

Related to dynamic problems, there are two different proposals: using a dual probability vector and competing with the main probability vector to generate samples [217], and using a memory scheme to store the best sample and the working probability vector [216].

In [141], a new EDA that has been applied to gene identification for molecular classification is presented. This approach, comparable to PBIL, updates the probability of each variable as:

$$p_{l+1}(x_i) = \alpha p_l(x_i) + (1-\alpha)M_l(x_i)\bar{w}(g_i) \tag{2.12}$$

where $\alpha \in (0, 1]$ is the learning rate, $M_l(x_i)$ is the empirical distribution of variable X_i, and $\bar{w}(g_i) \in [0, 1]$ is the normalized weight (by means of correlation with the classes) of each gene g_i corresponding to X_i in the dataset.

cGA

The compact Genetic Algorithm (cGA) [79] is quite similar to PBIL. It also uses a probability vector to guide the search through the space of possible solutions.

This algorithm completes the following steps: first, the probability vector is initialized (each component follows a Bernoulli distribution with parameter 0.5). Then, two individuals

Step 1. **for** $i = 1, \ldots, n$
if $(x^l_{i,1:2} \neq x^l_{i,2:2})$
if $(x^l_{i,1:2} = 1)$ $p_l(x_i) = p_{l-1}(x_i) + \frac{1}{K}$
if $(x^l_{i,1:2} = 0)$ $p_l(x_i) = p_{l-1}(x_i) - \frac{1}{K}$

Figure 2.2: Process to update the probability vector in cGA. K is a constant value fixed as a parameter.

are randomly sampled from the probability vector, and evaluated. Taking into account their fitness value, one of them will be the best ($x^l_{1:2}$) and the other the worst ($x^l_{2:2}$). The process to update the probability vector is presented in Figure 2.2.

It must be noted that the probability vector is updated in an independent way for each variable. This process of adaptation of the vector of probabilities towards the winning individual continues until the vector of probabilities has converged.

cGA has been applied to Feature Subset Selection [36], to the pruning of neural networks used in classification problems [37], or to active interval scheduling in hierarchical sensor networks [95].

A runtime analysis of cGA using different linear functions is presented in [55].

Finally, several modifications on cGA have been presented in the literature. In [60], a modified compact GA is developed for the intrinsic evolution of continuous time recurrent neural networks. In [83] a parallel cGA algorithm with a local search optimization to solve Multi-FPGA partitioning problems has been implemented using an hybrid approach. Two elitism-based cGAs: persistent elitist compact genetic algorithm (pe-cGA), and nonpersistent elitist compact genetic algorithm (ne-cGA) are presented in [2]. A family of cGAs for intrinsic evolvable hardware is presented in [61].

RELEDA

The Reinforcement Learning Estimation of Distribution Algorithm (RELEDA) was introduced in [140].

In this algorithm, an agent explores an environment perceiving its current state as well as information about the environment. Based on that information, the agent takes some decisions, making the environment change and receiving the value of this transition as a scalar reinforcement sign.

This algorithm is similar to UMDA, but the probability of each variable is updated applying a reinforcement learning method. The search of probability distributions is reduced to a number of parameters denoted by $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$ where $\theta_i \in \Re$ is a parameter related to the probability of the variable X_i through a function. The correlation between $p(x_i)$ and θ_i is expressed through the sigmoid function:

$$p(x_i) = \frac{1}{2}(1 + \tanh(\beta\theta_i)) \tag{2.13}$$

where β is the sigmoid gain.

In each generation, the value of the parameters θ_i is modified by a Δ_i value following:

$$\Delta\theta_i = \alpha(b_i - p(x_i))(1 - d_i) \tag{2.14}$$

Step 1. **for** $i = 1, \ldots, n$
if $(\alpha_i < 0)$ $p_i = p_i(1 - \lambda) + \lambda$
if $(\alpha_i > 0)$ $p_i = p_i(1 - \lambda)$

Figure 2.3: Process to update the probability vector in DEUM. λ is a learning rate (values between 0 and 1) fixed as a parameter.

$$b_i^{t+1} = \gamma b_i^t + (1 - \gamma) x_i \tag{2.15}$$

where b_i is the reinforcement signal (baseline), d_i is the marginal distribution of the variable X_i, x_i is the value of the variable X_i in the best individual in that generation, α is the learning rate, and γ is the baseline factor.

This algorithm has been compared in [140] to other EDAs (UMDA and PBIL) using two well-known problems: four peaks and bipolar function, showing that it requires fewer fitness evaluations to obtain an optimal solution.

DEUM

Distribution Estimation Using MRF with direct sampling (DEUM) [187]. This algorithm uses the Markov Random Field (MRF) modelling approach to update the probability vector. It can be seen as an adaptation of the PBIL approach by replacing marginal frequencies with an MRF model on a selected set of solutions.

In [32], MRF theory was used to provide a formulation of the joint probability distribution that relates solution fitness to an energy function calculated from the values of the solution variables. Mathematically:

$$p(x) = \frac{f(x)}{\sum_y f(y)} = \frac{e^{-U(x)}}{\sum_y e^{-U(y)}} \tag{2.16}$$

therefore

$$-\ln(f(x)) = U(x) \tag{2.17}$$

where $f(x)$ is the fitness function of an individual and $U(x)$ an energy function that specifies the joint probability distribution. Generally, the energy function involves interaction between variables but, for this particular approach, all the variables are considered independent. Therefore, the previous equation can be rewritten as:

$$-\ln(f(x)) = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \tag{2.18}$$

Each solution in any given population gives an equation satisfying the model. Therefore, selecting N promising solutions from a population allows us to estimate the distribution by solving $A\alpha = \boldsymbol{F}$, where A is the $N \times n$ dimensional matrix of values in the selected set, $\boldsymbol{\alpha}$ is the vector of MRF parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, and $\boldsymbol{F}$ is the N dimensional vector containing the value $-\ln(f(x))$ of the selected set of solutions.

Finally, the probability vector will be updated using the MRF parameters (see Figure 2.3).

This algorithm has been modified in [190], proposing an approach called DEUM$_d$ in which a MRF model is directly sampled to generate the new population. In [188] DEUM$_d$ was modified incorporating a simple Metropolis method, showing empirically that for linear

univariate problems the proposed univariate MRF models are very effective. The performance of DEUM$_d$ in a range of optimization problems is presented in [189].

2.3.2 Pairwise dependencies

Algorithms in this second group consider only dependencies between pairs of variables. In this way, estimation of the joint probability can still be done quickly. However, it must be noted that an additional step is required (not necessary in the previous algorithms): the construction of a structure that best represents the probabilistic model.

MIMIC

Mutual Information Maximization for Input Clustering (MIMIC) [47]. This approach searches (in each generation) for the best permutation between the variables. The goal is to find the probability distribution, $p_l^\pi(\boldsymbol{x})$, that is closest to the empirical distribution of the set of selected points when using the Kullback-Leibler distance, where

$$p_l^\pi(\boldsymbol{x}) = p_l(x_{i_1} \mid x_{i_2}) \cdot p_l(x_{i_2} \mid x_{i_3}) \cdots p_l(x_{i_{n-1}} \mid x_{i_n}) \cdot p_l(x_{i_n}) \tag{2.19}$$

and $\pi = (i_1, i_2, \ldots, i_n)$ denotes a permutation of the set of indexes $\{1, 2, \ldots, n\}$.

In addition, the Kullback-Leibler divergence between two probability distributions, $p_l(\boldsymbol{x})$ and $p_l^\pi(\boldsymbol{x})$, can be expressed as:

$$H_l^\pi(\boldsymbol{x}) = h_l(X_{i_n}) + \sum_{j=1}^{n-1} h_l(X_{i_j} \mid X_{i_{j+1}}) \tag{2.20}$$

where

$$h(X) = -\sum_x p(X = x) \log p(X = x) \tag{2.21}$$

denotes the Shannon entropy of the X variable, and

$$h(X \mid Y) = \sum_y h(X \mid Y = y) p(Y = y) \tag{2.22}$$

where

$$h(X \mid Y = y) = -\sum_x p(X = x \mid Y = \ y) \log p(X = x | Y = y) \tag{2.23}$$

denotes the mean uncertainty in X given Y.

Therefore, the problem of searching for the best $p_l^\pi(\boldsymbol{x})$ can be solved by searching for the permutation π^* that minimizes $H_l^\pi(\boldsymbol{x})$.

As a search over all the possible permutations will be unfeasible for most of the problems, a greedy search is proposed to find the π^* permutation. The process starts with the variable X_{i_n} with the smallest estimated entropy. In the following steps, the variable with the smallest average conditional entropy with respect to the variable selected in the previous step is chosen (obviously from the set of variables not yet chosen).

MIMIC has been used to solve several problems: the traveling salesman problem [165], feature subset selection [91], partial abductive inference problem in Bayesian networks [48], or learning of Bayesian networks [166].

In addition, some modifications of this algorithm have also been proposed, applying a repair method for solving constraint satisfaction problems [76], or introducing a mutation operator [77].

COMIT

Combining Optimizers with Mutual Information Trees (COMIT) [11]. This algorithms hybridizes the EDA approach with local optimizers. Estimation of the probability distribution of the selected individuals in each generation is done using a tree structured Bayesian network, learnt using the algorithm Maximum Weight Spanning Tree (MWST) proposed in [43].

In general terms, MWST looks for the probabilistic tree structure ($p_l^t(x)$) that best matches the probability distribution of the selected individuals ($p_l(x)$). To consider the quality of each possible tree, the Kullback-Leibler cross-entropy measure is used. The distance is minimized by projecting $p_l^t(x)$ on any MWST, where the weight of the branch (X_i, X_j) is defined by the mutual information measure:

$$I(X_i, X_j) = \sum_{x_i, x_j} p_{(X_i,X_j)}(x_i, x_j) \log \frac{p_{(X_i,X_j)}(x_i, x_j)}{p_{X_i}(x_i) p_{X_j}(x_j)} \tag{2.24}$$

Once the best tree has been obtained, some new individuals are sampled from it. The best individuals are selected to start a fast-search procedure (i.e. hill climbing), and some of the better solutions found during this search step will replace some of the individuals of the previous population.

BMDA

Bivariate Marginal Distribution Algorithm (BMDA) [153]. This algorithm uses a factorization of the joint probability distribution that only needs second-order statistics.

It is based on an acyclic (but not necessarily connected) dependency graph. This graph is constructed as follows: first, a variable is chosen arbitrarily and it is added as a node of the graph. This first variable is the one with the greatest dependency on the rest of the variables –measured by Pearson's χ^2 statistic.

Second, the variable with the greatest dependency between any of those previously added and the set of those not yet added is incorporated to the graph. This second step is repeated until there is no dependency surpassing a previously fixed threshold between already added variables and the rest. If this is the case, a variable is chosen at random from the set of those not yet used to create a new tree structure. The whole process is repeated until all variables are added into the dependency graph.

In each generation the factorization obtained with the BMDA is given by:

$$p_l(\boldsymbol{x}) = \prod_{X_r \in R_l} p_l(x_r) \prod_{X_i \in V \setminus R_l} p_l(x_i \mid x_{j(i)}) \tag{2.25}$$

where V denotes the set of n variables, R_l denotes the set containing the root variable – in generation l– for each of the connected components of the dependency graph, and $X_{j(i)}$ returns the variable connected to the variable X_i and added before X_i.

2.3.3 Multiple dependencies

Different works [24, 148] have shown the limitations of using simple approaches to solve difficult problems. It must be noted that in this kind of problems, different dependency relations can appear between variables and, hence, considering all of them independent or taking into account only dependencies between pairs of variables may provide a model that does not represent the problem accurately.

Several algorithms have been proposed in the literature using statistics of order greater than two to factorize the probability distribution. In this way, dependencies between variables can be expressed properly without any kind of initial restriction. However, it must be also noticed that the probability model required for some problems could be excessively complex and, sometimes, unaffordable in computational terms.

ECGA

Extended Compact Genetic Algorithm (ECGA) [78]. This algorithm divides the variables into a number of groups (clusters) which are considered independent. Therefore, in each generation, the factorization of the joint probability distribution is expressed as a product of marginal distributions of variable size. These distributions are related to the variables that are contained in the same group and to the probability distributions associated with them. In this way, the factorization of the joint probability distribution on the n variables is:

$$p_l(\boldsymbol{x}) = \prod_{c \in C_l} p_l(\boldsymbol{x}_c) \tag{2.26}$$

where C_l denotes the set of groups in the l^{th} generation, and $p_l(\boldsymbol{x}_c)$ represents the marginal distribution of the variables $\boldsymbol{X}_c$, that is, the variables that belong to the c^{th} group in the l^{th} generation.

The grouping is carried out using a greedy forward algorithm that obtains a partition between the n variables (as mention above, each group of variables is assumed to be independent of the rest).

The process starts considering n clusters (one variable in each cluster) and then continues trying to unify the pair of clusters that reduce the most a measure value. This value conjugates the sum of the entropies of the marginal distributions with a penalty for the complexity of the model based on the minimum description length principle (MLD) [163].

ECGA has been applied to feature subset selection [36] and to the pruning of neural networks used in classification problems [37].

From a theoretical point of view, in [180] empirical relations for population sizes and convergence times are presented.

Finally, some modifications of this algorithm have been proposed. In [108], a hybrid ECGA that combines crossover and mutation operators. The proposed algorithm combines the Building Blocks-wise crossover operator from ECGA with a recently proposed Building Blocks-wise mutation operator that is also based on the probabilistic model of ECGA [181]. In [179], a sub-structural niching method is proposed and applied to ECGA aiming to maintain diversity at the sub-structural level.

FDA

Factorized Distribution Algorithm (FDA) [127]. It must be noted that this algorithm differs from the others in regard to the probabilistic model. Instead of creating a new one at

Step 1. $BN_0 \leftarrow (S_0, \boldsymbol{\theta}^0)$ where S_0 is an arc-less DAG, and $\boldsymbol{\theta}^0$ is uniform $p_0(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i) = \prod_{i=1}^{n} \frac{1}{r_i}$
Step 2. $D_0 \leftarrow$ Sample M individuals from $p_0(\boldsymbol{x})$
Step 3. **for** $l = 1, 2, \ldots$ until the stopping criterion is met
$D_{l-1}^{Se} \leftarrow$ Select N individuals from D_{l-1}
$S_l^* \leftarrow$ Find the best structure according to a criterion:
conditional (in)dependence tests $\rightarrow$ EBNA$_{PC}$
penalized Bayesian score + search $\rightarrow$ EBNA$_{K2+pen}$
penalized maximum likelihood+search $\rightarrow$ EBNA$_{BIC}$
$\boldsymbol{\theta}^l \leftarrow$ Calculate θ_{ijk}^l using D_{l-1}^{Se} as the data set
$BN_l \leftarrow (S_l^*, \boldsymbol{\theta}^l)$
$D_l \leftarrow$ Sample M individuals from BN_l using PLS

Figure 2.4: Pseudo-code for the EBNA$_{PC}$, EBNA$_{K2+pen}$, and EBNA$_{BIC}$ algorithms.

each generation, the same model is used throughout the entire execution. Therefore, this algorithm needs the factorization and decomposition of the task to be given by an expert –which is not a common situation. Generally, due to this characteristic, it is intended to be applied to additively decomposable functions for which, using the running intersection property [106], a factorization of the mass-probability based on residuals, x_{b_i}, and separators, x_{c_i}, is obtained.

The joint probability distribution can be factorized as:

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{k} p_l(\boldsymbol{x}_{b_i} | \boldsymbol{x}_{c_i}) \tag{2.27}$$

As this factorization remains valid for all the iterations, the only changes are those in the estimation of probabilities.

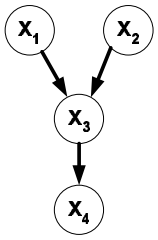
Theoretical results for FDA can be found in [127, 124, 125, 126, 223, 117]. In addition, the space complexity of the algorithm is studied by [64] using random additive functions as the prototype.

EBNA

In this section we present three different algorithms (EBNA$_{PC}$, EBNA$_{K2+pen}$, and EBNA$_{BIC}$), grouped under the name of Estimation of Bayesian Networks Algorithms (EBNAs). Introduced in [56, 100], their main characteristic is that the factorization of the joint probability distribution is encoded by a Bayesian network, learnt from the database containing the selected individuals in each generation. A common scheme for these approaches can be seen in Figure 2.4.

Before explaining the different variations of EBNAs, we proceed with a brief introduction to Bayesian networks that will be helpful to better understand the algorithms.

Bayesian networks Formally, a Bayesian network [41] over a domain $\mathbf{X} = (X_1, \ldots, X_n)$ is a pair (S,$\boldsymbol{\theta}$) that represents a graphical factorization of a probability distribution. The struc-



Structure — Local probabilities

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

Factorization of the joint mass-probability

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

Figure 2.5: 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).

ture S is a Directed Acyclic Graph (DAG) which reflects the set of conditional (in)dependencies between the variables. The factorization of the probability distribution is codified by S:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i|\boldsymbol{pa_i}) \tag{2.28}$$

where $\boldsymbol{Pa_i}$ is the set of parents of X_i (variables from which there exists an arc to X_i in the graph S). In Figure 2.5 for example, $\boldsymbol{Pa_3}$=$\{\boldsymbol{X_1},\boldsymbol{X_2}\}$ (X_1 and X_2 are the parents of X_3).

The second part of the pair, $\boldsymbol{\theta}$, is a set of parameters for the local probability distributions associated with each variable. If variable X_i can take 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:

$$p(x_i^k|\boldsymbol{pa_i^j}, \boldsymbol{\theta}_i) \equiv \theta_{ijk} \tag{2.29}$$

where $\boldsymbol{pa_i^1}, \ldots, \boldsymbol{pa_i^{q_i}}$ denote the values of $\boldsymbol{Pa_i}$ and the term q_i denotes the number of possible different instances of the parent variables of X_i. In other words, parameter θ_{ijk} represents the conditional probability of variable X_i being in its 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 Figure 2.5.

In the context of EDAs, EBNAs comprise a group of algorithms that use Bayesian networks to codify the dependencies between variables. At each generation, given a set of individuals (population), a Bayesian network must be learnt trying to reflect properly the relations between variables. After that, the Bayesian network is sampled in order to obtain the new population.

Related to the learning process, there are mainly two different methods: "score + search" and "detecting conditional (in)dependencies".

"score + search": This method uses a score (metric) to measure the quality of the Bayesian network. Among the different scores used, we can point out the Bayesian Information Criterion (BIC) [183] or the Bayesian Dirichlet equivalence (BDe) [81]. Once the Bayesian network has been assigned a score, the goal is to complete a search step, changing the structure of the Bayesian network with the aim of improving the current score.

Generally, the search step begins with an empty Bayesian network (without arcs) and, in the following steps, arcs will be added based on the score used to measure the quality of the network. In order to have an effective algorithm, it is necessary to find an adequate model as soon as possible (even if it is not optimal).

For example, Algorithm B [33] is a common method used to learn Bayesian networks. This algorithm uses a hill climbing strategy. Starting with an arc-less structure, it adds in each step the arc that maximizes the score. When no improvement can be achieved, the algorithm stops. An alternative to Algorithm B could be the use of the model created in the previous generation, instead of beginning each time with an empty structure.

Some of the algorithms that belong to this group are EBNA$_{BIC}$ and EBNA$_{K2+pen}$. Both use Algorithm B as a search method, but EBNA$_{BIC}$ uses the BIC score to measure the quality of the Bayesian network, and EBNA$_{K2+pen}$ combines the Bayesian approach to calculate the marginal likelihood [44] with a penalizing term, introduced to avoid an excessively complex Bayesian network.

"detecting conditional (in)dependencies": The techniques that belong to this group complete several tests to detect the relations between variables. These algorithms usually start with the complete undirected graph, and then independence tests are performed to remove edges. When no more edges can be removed, an orientation process is completed to create the Bayesian network. For example, EBNA$_{PC}$, one of the algorithms that belongs to this family, uses the *PC* algorithm [198] to detect the dependencies. Starting with the complete graph, it is "thinned" by removing edges with zero order conditional independence relations, "thinned" again using first order conditional relations, then second order conditional relations are taken into account, and so on. The set of variables conditioned on need only to 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 the graph.

Finally, once the Bayesian network has been learnt, new individuals are sampled to create the new population. Among the different methods, EBNAs use the Probabilistic Logic Sampling method [82]. In this method, the instances are generated one variable at a time in a forward way. That is, a variable is sampled after all its parents have already been sampled. To do that an ancestral ordering of the variables is given $(\pi(1), \ldots, \pi(n))$ where parent variables are before children variables. Once the values of $\mathbf{Pa}_{\pi(i)}$ have been assigned, we simulate a value for $X_{\pi(i)}$, using the distribution $p(x_{\pi(i)}|\boldsymbol{pa}_{\boldsymbol{\pi(i)}})$.

EBNA approaches have been applied to different problems: graph machine task [15, 14], partial abductive inference in Bayesian networks [48], feature subset selection [90, 92], feature weighting problems for K-NN [93], job scheduling problem [113], rule induction task [193], traveling salesman problem [165], partitional clustering [168], Knapsack problems [172], or software testing [173, 174].

In [76] EBNA was modified by applying a repair method for solving constraint satisfaction problems, and in [77] a mutation operator is introduced.

Some parallel approaches for EBNA$_{BIC}$ and EBNA$_{PC}$ have been presented in [120], applying a parallel and multi-objective version of EBNA$_{BIC}$ to a chemical problem [121].

In [64], the space complexity of the EBNA algorithm has been studied using random additive functions.

BOA

Bayesian Optimization Algorithm (BOA) [148, 150, 144, 145, 151, 149] uses a "score + search" method (B Algorithm) to construct the model, using as metric the Bayesian Dirichlet equivalence (BDe) [81]. In each generation, the process starts with an empty structure. In order to reduce the cardinality of the search space, the number of parents that each node can have is limited to k.

This algorithm has been extended and applied to several problems. In [146] BOA is modified in order to model hierarchical problems using a type of hybrid model called a Huffman network. In [151] it is adapted to include local structures by using decision graphs to guide the network construction.

Other extension named Mixed BOA that uses decision trees with mixed decision nodes is presented in [132]. In [129] MBOA is combined with variance adaptation in order to improve its behavior in the continuous domain.

Some theoretical studies have been completed using Bayesian networks to estimate the fitness of the individuals [154] or to reduce the number of parameters needed to execute the BOA algorithm [152].

Related to multi-objective approaches, BOA was integrated into an evolutionary multi-objective optimizer using a special selection scheme [105]; also, in [155] a multi-objective hierarchical BOA is presented, combining hBOA, NSGA-II [54], and clustering in the objective space.

The real-coded Bayesian Optimization Algorithm (rBOA) algorithm is proposed in [3], as an extension of BOA to the area of real-value optimization. It performs a Bayesian factorization of a mixture of probability distributions, and finds maximal connected graphs (substructures) of the Bayesian factorization graph (probability model). Then, it fits each substructure independently by a mixture distribution estimated for clustering results in the corresponding partial-string space. Finally, offspring is obtained by a sampling method based on independent subspaces.

Two parallel approaches have been presented for BOA using a pipelined parallel architecture [130] and clusters of computers [131]. Recently, in [133] the parallelization of the learning of decision trees using multi-threaded techniques has been proposed.

The different BOA approaches have been applied to feature subset selection [36], to the pruning of neural networks used in classification problems [37], and to two real world problems (such as Ising spin-glass systems and maximum satisfiability) [147]. In [96, 97] a comparative review of some EAs (including MBOA) is presented, evaluating them on a different number of test functions in the continuous domain.

LFDA, FDA$_L$, FDA-BC, FDA-SC

Learning Factorized Distribution Algorithm (LFDA), introduced in [125], essentially follows the same steps as EBNA$_{BIC}$. The main difference is that in the LFDA the complexity of

the model learnt is controlled by the BIC metric in conjunction with a restriction on the maximum number of parents that each variable can have in the Bayesian network.

An initial algorithm FDA_L is proposed in [136], to learn –by means of conditional (in)dependence tests– a junction tree from a database. The underlying idea is to return the junction tree that best satisfies the previous independences, once a list of dependencies and independencies between the variables is obtained.

Also, in [134], a structure learning algorithm that takes into account questions of reliability and computational cost is presented. The algorithm, called FDA-BC, studies the class of Factorized Distribution Algorithm with Bayesian networks of Bounded Complexity.

Similar ideas are introduced in the FDA-SC [135]. In this case the factorization of the joint probability distribution is done using simple structures, i.e. trees, forests or polytrees.

PADA

Polytree Approximation of Distribution Algorithms (PADA) [196]. The factorization is done using a Bayesian network with polytree structure (no more than one undirected path connecting every pair of variables). The proposed algorithm can be considered a hybrid between a method for "detecting conditional (in)dependencies" and a procedure based on "score + search".

MN-EDA

Markov Network Estimation of Distribution Algorithm (MN-EDA) [175]. The authors introduce a method that approximates probability distributions using what he calls "messy factorizations". In order to learn the factorizations, the algorithm combines a reformulation of a probability approximation procedure used in statistical physics (Kikuchi approximations), with a novel approach for selecting the initial inputs required by the procedure.

In addition, a new method for sampling solutions from the approximation is also used (Gibbs Sampling). The learning and sampling methods are the primary components of this MN-EDA.

2.3.4 Other algorithms

- An EDA in the permutation representation domain that uses Edge Histogram Based Sampling Algorithms (EHBSAs) is presented in [203]. The algorithm starts generating random permutation strings for each individual in the population. Then, individuals are evaluated and the most promising solutions are used to construct a symmetrical Edge Histogram Matrix (EHM) where an edge is a link between two variables in an individual. Finally, new individuals will be sampled from that EHM, replacing the old population. The behavior of the algorithm is tested on the traveling salesman problem.

  Further modifications of the algorithm can be consulted in [204], where an asymmetrical EHM is used to solve the flow shop scheduling problem. Also, in [205] a tag node is employed. Instead of assuming that the first variable is the first job, the tag node (virtual node) is used to indicate the positions.

- Estimation of Distribution Programming (EDP) is presented in [213]. This program is codified using a probabilistic graphical model given by a Bayesian network. The search method follows the common scheme of EDAs to solve Genetic Programming applicable

problems. This work is extended in [214], where the proposed EDP is mixed with a GP algorithm.

- Dependency Detection for Distribution Derived from *df* (DDDDD or D^5) [202]. This approach combines EDAs with linkage identifications in order to detect dependencies. It has three parts: (1) calculation of fitness differences –each variable is perturbed and then fitness difference for the perturbation is calculated–, (2) classification of individuals according to the fitness difference, and (3) estimation of the classified individuals based on entropy measures.

- The algorithm presented in [211] uses marginal frequencies to constrain the estimated probability distribution. A schema is a subset of the search space where the values of some variables are defined (fixed) and the values of the others are variable (represented by *). The order of the schema is defined by the number of *. Given a frequency distribution over the search space and a schema, the corresponding schema frequency is just the sum of the relative frequencies of the elements of that schema.

  The entropy of this distribution is subsequently maximized and the distribution is sampled to produce a new population. In this work, only contiguous order-2 schema families are used, proposing as a future work the use of higher order schemas.

- In [160] a Learning Automata based Estimation of Distribution Algorithm (LAEDA) is presented. This algorithm follows the general EDA scheme, and uses a variable structure learning automata as the probability model.

- Finally, Unsupervised Estimation of Bayesian Network Algorithm (UEBNA) is introduced in [157]. This approach uses a Bayesian network for data clustering in order to factorize the joint probability distribution of the individuals selected at each iteration. The goal of this approach is to optimize multimodal problems.

2.4 EDA approaches in the continuous domain

2.4.1 Introduction

In this section some EDA-based approaches in continuous domains will be presented. The organization is similar to that used for the discrete domain, classifying the algorithms in four different groups: those that consider all variables as independent, algorithms that take into account pairwise dependencies, those that allow for any kind of dependency relation and, finally, a set of algorithms that mix EDAs framework with ideas from other research fields.

2.4.2 Without dependencies

The simplest approaches in this continuous domain consider all variables as independent. It is usually assumed that the joint density function follows a n-dimensional normal distribution, which is factorized by a product of uni-dimensional and independent normal densities. Mathematically $\boldsymbol{X} \equiv \boldsymbol{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \sum)$, that is:

$$f_{\boldsymbol{\mathcal{N}}}(\boldsymbol{x}; \boldsymbol{\mu}, \sum) = \prod_{i=1}^{n} f_{\mathcal{N}}(x_i; \mu_i, \sigma_i^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}(\frac{x_i-\mu_i}{\sigma_i})^2} \tag{2.30}$$

UMDA$_c$

Univariate Marginal Distribution Algorithm for continuous domains (UMDA$_c$) [101, 102]. This approach assumes that all the univariate distributions are normal and hence, the parameters to be estimated at each generation for each variable are two: the mean and the standard deviation.

However, this algorithm could be extended performing statistical tests for each variable at each generation, in order to find an appropriate density function. A structure identification approach is carried out, identifying the density components of the model using hypothesis tests. Once these densities have been obtained, the estimation of the parameters is performed by their maximum likelihood estimates.

UMDA$_c$ has been applied to the learning of Bayesian networks in the space of orderings [166]. Some theoretical studies have been completed about the behavior of this algorithm in [69]. In addition, a theory that predicts the behavior of UMDA$_c$ with truncation selection on monotonous fitness functions has been developed in [72].

SHCLVND

Stochastic Hill Climbing with Learning by Vectors of Normal Distributions (SHCLVND) [169]. This proposal estimates the joint density function as a product of uni-dimensional and independent normal densities. The vector of means $\boldsymbol{\mu} = (\mu_1, ..., \mu_i, ..., \mu_n)$ is adapted by means of the Hebbian rule:

$$\boldsymbol{\mu}^{l+1} = \boldsymbol{\mu}^l + \alpha \cdot (\boldsymbol{b}^l - \boldsymbol{\mu}^l) \tag{2.31}$$

where $\boldsymbol{\mu}^{l+1}$ denotes the vector of means in the generation $l+1$, α denotes the learning rate, and $\boldsymbol{b}^l$ denotes the barycenter of the B (an amount fixed at the beginning) best individuals in the l^{th} generation. Adaptation of the vector of variances $\boldsymbol{\sigma}$ is carried out using a reduction policy in the following way:

$$\boldsymbol{\sigma}^{l+1} = \boldsymbol{\sigma}^l \cdot \beta \tag{2.32}$$

where β denotes a previously fixed constant ($0 < \beta < 1$).

PBIL$_c$

This is an extension of the discrete PBIL algorithm to continuous domains [184]. This algorithm also assumes a joint density function that follows a Gaussian distribution, factorizable as a product of uni-dimensional and independent marginal densities. The adaptation of each component of the vector of means is carried out using the following formula:

$$\mu_i^{l+1} = (1 - \alpha) \cdot \mu_i^l + \alpha \cdot (x_{i,1:N}^l + x_{i,2:N}^l - x_{i,N:N}^l) \tag{2.33}$$

where μ_i^{l+1} represents the i^{th} component of the mean $\boldsymbol{\mu}^{l+1}$ at generation $l+1$, α is a constant, and $x_{i,1:N}^l$, $x_{i,2:N}^l$, and $x_{i,N:N}^l$ denote the best, second best, and the worst individual of the generation l respectively.

Related to the vector of variances $\boldsymbol{\sigma}$, several methods for updating it are discussed. The simplest one is just to fix all σ_i at some constant value.

Based on this initial approach, some modifications have been proposed. In [185], the authors introduce a implementation of a real-coded version of PBIL. They assume that each

variable is constrained to some interval $(a_i{}^l, b_i{}^l)$, and present $z_i{}^l$ $(i = 1, \ldots, n)$, representing the probability of the i^{th} variable to be greater than $\frac{(a_i{}^l + b_i{}^l)}{2}$. The probabilities z_i are updated at each generation and, when their value is close to a limit (upper or low), the interval is halved in the appropriate direction, z_i is reset to 0.5, and the algorithm continues.

In [219] an extension of PBIL_c is presented based on a new updating rule and a self-adaptive learning rate. Instead of using only the two best and the worst individuals to update the mean, this new rule uses all the individuals by means of the product of the learning rate and the sum of the difference between the mean vector and each individual, weighted by the ratio between the fitness value of that individual and the sum of the fitness values of all individuals. In summary, the contribution of each individual is simply decided by its position and the fitness value.

Related to the self-adaptive learning rate, they propose a separate rate for each variable. The idea is that when the mean vector oscillates, this suggests that there may be multiple attractors in the neighboring area; therefore, the algorithm should be cautious by using a small learning rate. However, when the algorithm moves in the same direction within consecutive generations, this may imply that there is only one major attractor and, thus, it will be safe to gradually increase the learning rate to speed up the movement.

2.4.3 Bivariate dependencies

MIMIC_c^G

This algorithm is an extension of the MIMIC algorithm (proposed in [47]) to the continuous domain. Introduced in [101, 102], it assumes that the underlying probability model for every pair of variables is a bivariate Gaussian density function. The joint density function is factorized by a chain structure, fitting the model as close as possible to the empirical data by using one univariate marginal density and $n - 1$ pairwise conditional density functions.

In [166] this algorithm was applied to the learning of Bayesian networks in the space of orderings.

2.4.4 Multiple dependencies

The models presented in this section learn density functions without restrictions. This can be done by using non restricted multivariate normal densities, or Gaussian networks.

EMNA

Estimation of Multivariate Normal Algorithm (EMNA) [103]. In this approach, the factorization of selected individuals in each generation is performed by a multivariate normal density function. The estimation of the vector of means and the covariance matrix is performed by their maximum likelihood estimates.

In addition to this algorithm (known as EMNA_{global}), two variations have been proposed:

EMNA_a: This is an adaptive version. Once the first model has been obtained, it has a flow similar to a steady-state genetic algorithm. After a new individual is simulated from the current density model, its fitness value is compared with the worst fitness value of the population. If the simulated individual improves the goodness value, it will replace the worst individual. After that replacement, it is necessary to update the parameters of the multivariate normal density function.

EMNA$_i$: This is an incremental version. As the previously presented one, this also generates only one individual from each model. The difference is that in this approach, every new individual is added to the population. Therefore, the number of individuals in the population increases as the algorithm evolves.

EDA$_{mvg}$

Estimation of Distribution Algorithm with multivariate Gaussian model (EDA$_{mvg}$) [222]. This approach (similar to EMNA), uses a multivariate Gaussian distribution to model selected individuals and generate new individuals (using the Choleski decomposition). The new population is created selecting the best individuals from the union of the two populations (the previous and the recently sampled).

The target application in [222] was the design of superconductive magnet configurations in magnetic resonance imaging systems.

In addition, the algorithm has also been tested on a benchmark suite with 25 problems in [220].

RECEDA

Real-Coded Estimation of Distribution Algorithm (RECEDA) [139]. This approach calculates means (μ) and covariances (Σ) of the variables from the selected individuals of a population and produces new offspring by sampling those means and covariances.

All the variables are considered multivariate normal distributions and their covariance matrix is decomposed using the Cholesky decomposition: $LL^T = \Sigma$. Then, a vector of independent normal deviates (Z) is obtained, sampling the new individuals using the following equation:

$$X = \mu + LZ \tag{2.34}$$

Finally, the new population is created replacing the old population with some offspring.

EGNA

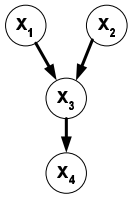
Estimation of Gaussian Networks Algorithms (EGNAs) [101, 102, 104] use Gaussian networks to complete the factorization of the selected individuals in each generation. Their general behavior is illustrated in Figure 2.7.

Before explaining the different variants of EGNAs, an introduction to Gaussian networks is given in the following subsection.

Gaussian networks This probabilistic model assumes that the joint density function follows a multivariate Gaussian density [209]. In this case, each variable $X_i \in \boldsymbol{X}$ is continuous and each local density function follows the linear-regression model:

$$f(x_i \mid \boldsymbol{pa}_i, \boldsymbol{\theta}_i) \equiv \mathcal{N}(x_i; m_i + \sum_{x_j \in \boldsymbol{Pa}_i} b_{ji}(x_j - m_j), v_i) \tag{2.35}$$

where $\mathcal{N}(x; \mu, \sigma^2)$ is a univariate normal distribution with mean μ and variance σ^2. Given this form, a missing arc from X_j to X_i implies that $b_{ji} = 0$ in the former linear-regression model. The local parameters are given by $\boldsymbol{\theta}_i = (m_i, \boldsymbol{b}_i, v_i)$, where $\boldsymbol{b}_i = (b_{1i}, \ldots, b_{i-1i})^t$ is a column vector. A probabilistic graphical model constructed with these local density functions is a Gaussian network [186].



Structure

Local densities

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

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

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

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

Factorization of the joint density function

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

Figure 2.6: Structure, local probabilities and resulting factorization for a Gaussian network with four variables.

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

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

$$f(\boldsymbol{x}) \equiv \boldsymbol{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) \equiv (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{t} \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})} \tag{2.36}$$

where $\boldsymbol{\mu}$ is the vector of means, Σ is an $n \times n$ covariance matrix, and $|\Sigma|$ denotes the determinant of Σ. An example of a Gaussian network can be seen in Figure 2.6.

In the context of EDAs, the algorithms known as EGNAs use Gaussian networks to codify the dependencies between variables. At each generation, the Gaussian network that best represents the relation between variables is sought. Finally, this Gaussian network will be used to sample new individuals.

The learning step is similar to that of Bayesian networks, with the same two main approaches: "search + score" and "detecting conditional (in)dependencies". EGNA$_{BGe}$ and EGNA$_{BIC}$ are algorithms that belong to the first group ("score + search"). EGNA$_{BGe}$ constructs the Gaussian network (starting in each generation from scratch) by means of a Bayesian score (BGe metric) [66] and the algorithm B [33]. EGNA$_{BIC}$ uses a penalized maximum likelihood score based on the BIC criterion [183]. In the second group, "detecting conditional (in)dependencies", we can point out the EGNA$_{EE}$ algorithm, which starts with a complete Gaussian network and performs edge exclusion tests based on the likelihood ratio test [195] in order to remove the unnecessary links (edges).

Finally, related to the creation of new individuals (sampling), a method introduced in [162] is used. This method generates instances of $\boldsymbol{X}$ by computing X_1, then X_2 conditional on X_1, and so on. This technique is similar to Probabilistic Logic Samplig (used for the simulation of Bayesian networks). For the simulation of a univariate normal distribution, a simple method based on the sum of 12 uniform variables can be applied.

Step 1. $D_0 \leftarrow$ Sample M individuals from a uniform distribution
Step 2. **for** $l = 1, 2, \ldots$ until the stopping criterion is met
$D_{l-1}^{Se} \leftarrow$ Select Se individuals from D_{l-1}
$\hat{S}_l \leftarrow$ Structural learning via:
edge exclusion tests $\rightarrow$ EGNA$_{EE}$
Bayesian score + search $\rightarrow$ EGNA$_{BGe}$
penalized maximum likelihood+search $\rightarrow$ EGNA$_{BIC}$
$\hat{\boldsymbol{\theta}}^l \leftarrow$ Calculate the estimates for the parameters of $\hat{S}_l$
$M_l \leftarrow (\hat{S}_l, \hat{\boldsymbol{\theta}}^l)$
$D_l \leftarrow$ Sample M individuals from M_l
using the continuous version of the PLS algorithm

Figure 2.7: Pseudo-code for the EGNA$_{EE}$, EGNA$_{BGe}$, and EGNA$_{BIC}$ algorithms.

EGNAs have been applied to tasks of adjusting weights for artificial neural networks [46], optimization problems [16], job shop scheduling [113], or the traveling salesman problem [165].

IDEA

Iterated Density Evolutionary Algorithms (IDEA) [23, 27, 22]. This is a general framework suitable to be used in both discrete and continuous domains.

Two are the main characteristics of IDEA: (1) in each generation individuals are sampled from a truncated distribution (where the truncation point is given by the worst individual found in the previous generation), and (2) only a part of the population is replaced in each generation.

Some experiments in continuous optimization using IDEA are presented in [29, 25, 26]. The density models used are multivariate normal, the histogram distribution, and the normal kernel distribution with a diagonal covariance matrix. While in the first two references the model is searched using a greedy search with the Kullback-Leibler divergence as score, in the last reference the model is obtained by a hypothesis test approach.

An extension of IDEA to learn mixtures of multivariate normal distributions in each generation by means of two fast clustering algorithms, leader and k-means, was presented in [28].

Finally. in [96, 97] a comparative review of some EAs (including IDEA) is presented, evaluating them on a different number of test functions in the continuous domain.

2.4.5 Other algorithms

- A finite Adaptive Gaussian Mixture model (*AMix*) density estimator was introduced in [63, 62]. In *AMix*, the density function, $f_l(\boldsymbol{x})$, is factorized as a product of univariate densities:

$$f_l(\boldsymbol{x}) = \prod_{i=1}^{n} f_l(x_i) \tag{2.37}$$

where $f_l(x_i)$ is represented at each generation by the sum of component distributions:

$$f_l(x_i) = \sum_{j=1}^{K_l} \pi_{l,j} f_l(x_i \mid j) \tag{2.38}$$

being K_l the number of mixture components, $\pi_{l,j}$ the mixing coefficient for the j^{th} component, and $f_l(x_i \mid j)$ the univariate Gaussian density corresponding to the j^{th} clustering, that is $f_l(x_i \mid j) \equiv \mathcal{N}(x_i; \mu_{i,j}^l, \sigma_{i,j}^{2,l})$.

In [62] it is also proposed to model each variable by a finite Gaussian kernel density estimator.

- A model based on conditional Gaussian networks is introduced in [156] where each component of the mixture is a Gaussian network whose structure is a tree augmented network and where the weight of each component is obtained by an adaptation of the EM algorithm. It is intended to be applied to multimodal problems, taking advantage of data clustering in order to efficiently discover all the global optima of a given optimization problem.

- Induce Chromosome Elements Exchanger (ICE) [30] is a new tool for finding and using structure of permutation problems by estimating marginal product factorized probability distributions in the space of permutations.

- Voronoi-based Estimation of Distribution Algorithm for Multi-objective Optimization (VEDA) algorithm [138]. VEDA employs Voronoi diagrams to generate stochastic models, which cover the parameter space. A Voronoi diagram is a method for partitioning a space related to the nearest-neighbor partition. Clustering techniques have been used to divide the selected individuals into a number of groups and then one Voronoi diagram (one model) will be generated for one group. Meanwhile, principal component analysis (PCA) has been employed to reduce the dimensionality. Finally, new offspring will be generated by sampling from the generated models.

- In [158], the author presents a new EDA designed to solve real-valued optimization problems. This proposal uses a distribution tree. This distribution tree is built recursively searching for the best axis-parallel split, using the set of selected individuals. The leaf nodes of the tree fully cover the search space, and the model can be seen as a mixture of multivariate uniform distributions. Finally, new individuals are sampled from the distribution tree.

- Evolutionary Continuous Optimization by Distribution Estimation with Variational Bayesian Independent Component Analyzers Mixture Model (vbICA-MM) [42]. This algorithm tries to overcome the drawbacks of Gaussian mixture or kernel-based approaches breaking the requirement of each component being Gaussian. It uses a sophisticated technique, based on variational Bayesian independent component analyzers mixture model. In this way, the algorithm is able to model non-Gaussian clusters and also determines the local dimensionality if each cluster by employing Bayesian inference.

- PolyEDA, introduced in [73], is an algorithm that combines Estimation of Distribution Algorithms and Linear Inequality Constraints. This proposal has been created to

solve constrained optimization problems. Typically, when facing this type of problems, it is necessary to use some techniques in order to handle invalid solutions. In this approach, some changes are introduced into the probabilistic model and the sampling method in such a way that only feasible solutions are obtained. The probabilistic model used is based on factorizations of multivariate truncated normal distributions. In this model, all solutions that are feasible under the linear inequality constraints have positive probabilities, whilst solutions that are infeasible have a probability of zero. For the sampling of feasible solutions from this model, a Gibbs sampler is used. This sampling method allows the generation of random values from the multivariate truncated distribution without calculating their density.

- In [221], the authors present a new type of EDA for continuous domains. They propose a three-step algorithm that uses EAs with diversity maintenance technique, clustering, and a simple EDA based on a multivariate Gaussian distribution.

- Finally, two new EDAs were presented in [115]. These algorithms take advantage of data clustering to break the single Gaussian distribution assumption. CEGNA$_{BGe}$ combines a clustering technique, Rival Penalized competitive learning (RPCL) [212], with EGNA$_{BGe}$. CEGDA uses adaptive RPCL as both a clustering and estimation approach.

2.5 EDAs and parallelism

The advances in hardware and software during the last decades have provided new tools to the research community. Nowadays, there are new development and execution platforms which are really helpful and allow researches to propose new approaches that improve existing algorithms.

In particular, the availability of powerful computational resources (such as clusters of computers, computational grids and so on) has encouraged the development of parallel and distributed applications that can make use of these resources. These parallel applications may reduce execution time, improve solution accuracy, or manage larger problems compared with their sequential counterparts.

The modularity of EAs makes this family of algorithms suitable to be parallelized. In the literature we can find different proposals with this objective, which can be classified in two groups: (1) those that create different sub-populations (Island models) and exchange information among them trying to improve the behavior of the sequential algorithm (mainly in terms of the quality of the obtained solution), and (2) those whose behavior is exactly the same of the sequential version (parallelization of the sequential version), being the main goal the reduction of execution time and the applicability to larger problems.

2.5.1 Island models

Even if the idea of parallelism in EAs was discussed in different papers long time ago [31, 86], it has been only in the last two decades when the availability of hardware and software has made possible the design of usable parallel solutions.

Starting from the idea of the sequential algorithm, where a unique population exists and each of the individuals can potentially mate with any other, two new proposals appear: distributed EAs (dEAs) and cellular EAs (cEAs).

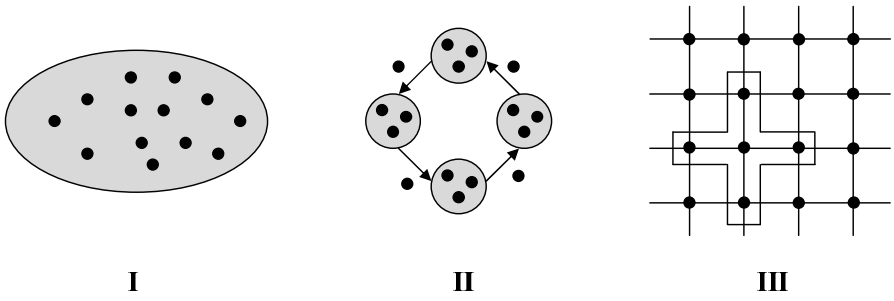


Figure 2.8: (I) sequential EA where all individuals belong to the same population, (II) dEAs, where individuals are distributed in several sub-populations, and (III) cEAs, with individuals distributed in a grid.

dEAs are known as coarse grain Parallel EAs. This kind of algorithms use several sub-populations (islands), instead of the usual single population. These populations evolve in a quasi-independent way and, with some fixed frequency (usually a number of generations), some individuals are exchanged between islands. Different topologies have been tested for different problems, including for example rings and stars.

Related to cEAs (fine grain Parallel EAs), individuals are generally placed on a toroidal n-dimensional grid (one at each position). Every individual has a neighborhood, and the breeding operators will be applied between the individuals in the neighborhood. Figure 2.8 shows the distribution of the populations in sequential EAs, dEAs and cEAs.

These ideas have been widely applied to GAs, and a summary of different parallel techniques for Parallel GAs can be consulted in [35]. In addition, [5] presents a review about parallelism and EAs, and in [4] an up-to-date reference on parallel metaheuristic issues can be consulted.

In the last years, these ideas and implementations have been applied also to EDAs. In [116], an asynchronous distributed framework for EDAs is presented (implemented for the UMDA algorithm), where the population is distributed in different islands (processors), completing migrations of individuals among them. In addition, as the main characteristic of EDAs is the use of probabilistic models, in [49] the authors design different island-based (dEAs) topologies, and carry out some experiments using two different types of migration: individuals and probabilistic models. Related to the latter contribution (exchange of probabilistic models), a univariate EDA for the discrete domain is used. In the migration step, each island mixes its probabilistic model with the received one. In that preliminary work, the researchers propose a convex combination of the models, while in a later paper [50], the model combination is improved using a local search. The results obtained from the experiments show that, in general, better solutions are obtained by exporting the probabilistic models rather than exchanging individuals. Furthermore, solutions provided by the best island-based model are often superior than those obtained with the sequential version. It can be observed (as seen for GAs), that having different populations allows the algorithm to search in different parts of the solution space. In addition, it helps to maintain a higher diversity, avoiding premature convergence.

These ideas have also been adapted to the continuous domain [51], translating the proposals to the $UMDA_c$ algorithm. In [84], the authors propose the Distributed Probabilistic Model Building GA (DPMBGA), a new EDA approach that uses Principal Component Analysis to detect dependencies between the variables and uses normal distributions to cre-

ate new individuals. The model uses a ring topology and exchanges individuals.

This algorithm has been extended [192] introducing the penalty method and the pulling back method in order to improve the characteristics of the previous version when solving constraint problems.

In summary, this is a new field of research inside the EDA family, and there are still some aspects that need further work. Firstly, these approaches need additional parameters, and it is not clear which is the best combination of values in order to properly tune an island-based model. For example, it is necessary to decide the topology to be used, the number of islands, the migration scheme, and so on. According to the results, some general ideas can be obtained but, generally, the values of the parameters seem to be problem dependent. Secondly, these experiments have been carried out using algorithm with simple (univariate) probabilistic models. These models are able to obtain good results but, in problems where there are dependencies between the variables, they usually behave worse than more complex probabilistic models. Therefore, it could be interesting to study the way these ideas can be extended to algorithms that use pairwise or multi-dependencies models.

2.5.2 Parallelization of sequential versions

We refer as "parallelization of the sequential version", the process of developing a parallel implementation of the original (sequential) EDA, without major changes, with the goal of reducing the execution time while maintaining its behavior.

Several EDAs have been proven successful to solve a range of complex problems. However, their application may become unfeasible if the individual size grows to a few hundred variables, or the fitness function is too complex. A fast, parallel EDA would not improve the quality of the solution, but would allow the utilization of these approaches to solve larger, more complex problems –in reasonable execution times. This is the approach we have followed in this work. We present it, applied to EDAs based on probabilistic graphical models (Bayesian networks in EBNAs, Gaussian networks in EGNAs), in the following chapter.

2.6 Summary

In this chapter, a family of algorithms that belong to the field of Evolutionary Algorithms has been introduced: Estimation of Distribution Algorithms (EDAs).

In the last few years, several algorithms have been proposed in the discrete domain as well as in the continuous domain. It has been the aim of this chapter to introduce, firstly, the common scheme of EDAs, explaining latter the different proposals and implementations.

Algorithms have been classified in several groups attending to how dependencies between variables are considered: no dependencies, pairwise dependencies, and multiple dependencies. Finally, a brief review of parallelism and EDAs has been presented as an introduction to the following chapter.

Chapter 3

Parallel implementation of EDAs based on probabilistic graphical models

As introduced in the previous chapter, EDAs are a family of algorithms that have been successfully used to solve a variety of complex optimization problems. The range of EDAs is large, but they can be aggregated taking into account the way dependencies between variables are considered. Those EDAs that look for multi-dependencies are the ones that usually have the best behavior (obtain the best results) but, unfortunately, are also the ones that need the most computation effort (execution time).

In this chapter we proceed with the parallel implementation of four algorithms that belong to this group. Two of them in the discrete domain (EBNA_{BIC} and EBNA_{PC}) and the other two in the continuous domain (EGNA_{BIC} and EGNA_{EE}). This study concludes that parallel approaches are useful, allowing the algorithms to be used in large problems, where sequential versions are useless due to the required execution time.

3.1 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 yielding very good results. However, there is still a significant drawback, which, in many situations, makes the use of such algorithms prohibitive in real environments: computation time. That is, the algorithms are able to solve a complex problem but not as quickly as we require, with execution times that may take days or even weeks.

Related to EDAs, the estimation of the joint probability distribution can easily become a computational bottleneck. Even if there are (as seen in the previous chapter) some approaches that try to solve this disadvantage simplifying the probabilistic model, it must be noted that, generally, the algorithms that provide the best results are those that use unrestricted probabilistic graphical models. Therefore, in order to make these approaches suitable to be used in large problems, we decided to apply parallelization techniques to reduce the computation time.

Excluding the execution time required to compute the objective function, the main computational cost of the algorithms is the time consumed in learning the probabilistic graphical model at each step. 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.

It must be also taken into account that, depending on the particular problem to be solved, there is another phase that could require an important amount of time during the execution: "sampling and evaluation". In this chapter, the parallelization of the "sampling and evaluation" step is presented only for the EGNA$_{EE}$ algorithm, although it can easily be adapted to any of the others –as it will be done in the next chapter for the parallel EBNA$_{BIC}$ algorithm.

Concerning parallel approaches to implement EDAs, approximations for some particular algorithms can be found in the literature. In [114] two different parallel proposals for an algorithm that uses probabilistic graphical models in the combinatorial field (EBNA$_{BIC}$) are proposed. These implementations use threads on a multiprocessor with shared memory, so that all data structures are shared between the different processes, with each of them computing a piece of work.

Also interesting is the work done in [130, 131] where two different parallel versions of the *BOA* algorithm are proposed using a pipelined parallel architecture and clusters of computers respectively. More recently, the parallelization of the learning of decision trees using multi-threaded techniques has been proposed in [133].

Another parallelization approach can be found in [1] where a basic framework that facilitates the development of new multiple-deme parallel EDAs is presented.

As we stated before, in this chapter we propose new parallel versions of certain EDAs that use probabilistic graphical models to learn the probability distribution of the selected individuals, in both discrete and continuous domains. Specifically, we have chosen one algorithm for each learning method of probabilistic graphical models used in EDAs. Our main goal has been to implement different parallel versions for each algorithm, trying to reduce the execution time while maintaining exactly the same behavior of the sequential version. In this way, we provide parallel algorithms that are able to be applied to problems that require excessive execution times when using sequential EDAs. Implementation has been carried out mixing two parallel application programming interfaces: Message Passing Interface (MPI) [8] and POSIX threads [34].

In the discrete domain, the algorithms chosen are EBNA$_{BIC}$ and EBNA$_{PC}$. Our parallel version of the EBNA$_{BIC}$ algorithm extends the version presented in [114], being suitable for a multiprocessor computer, a cluster, or any combination of both scenarios. Also, our work extends [130] and [131] allowing the learning of the Bayesian network without taking into account order restrictions between the variables. In the continuous domain, parallel versions of EGNA$_{BIC}$ and EGNA$_{EE}$ algorithms are proposed.

In addition to the experiments presented in this chapter, the performance analysis of the parallel EBNA$_{BIC}$ algorithm will be extended in Chapter 4, measuring execution times and efficiency on different target machines (clusters and multiprocessors) for a particular problem, more complex than those dealt with in this chapter.

The rest of this chapter is organized as follows: in Section 3.2, general considerations for the parallel proposals are presented. In Sections 3.3 and 3.4 the parallel solution for the four different algorithms is explained following this scheme: a brief explanation of the algorithm, parallel implementation(s) and an evaluation of the results obtained. Finally, conclusions are presented in Section 3.5.

Table 3.1: Time measurement (%) of the learning phase for different algorithms and problems.

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

3.2 General considerations for parallel proposals

Before undertaking the implementation of a parallel alternative to an existing sequential algorithm, it is essential to study the structure and functionality of the sequential solution and find the parts amenable to parallelization. To this end, we have executed the sequential version of the algorithms with different individual sizes in order to measure the computation time that each part of the algorithm requires. As we have said in the introduction and can be seen in Table 3.1[1](showing the time required to complete the learning step in percentages), the most time-consuming process is the learning phase, which generally accounts for 50-99% of total execution time. Our effort will therefore focus on understanding how the learning process proceeds and finding a good parallel approach for this phase.

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

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

Message Passing Interface (MPI) [8] is an Application Programming Interface (API) that can be used by an application process to communicate with other process that may be running in the same or in different computers. In the latter case, message exchange is performed using a network. It was designed and standardized by the MPI forum, a group of academic and industrial experts, to be used in very different types of parallel computers. A wide set of functions are available for managing, sending and receiving messages.

In relation to POSIX threads [34], most operating systems have thread support, that is, they include capabilities to spawn new "lightweight processes" (actually, threads) that share the same variable space. With this paradigm, a set of processes collaborating to perform a common task are implemented as threads that share the same memory space, so they can communicate directly using a set of global variables. The advantage of using this approach is that communication is fast and efficient, although the synchronization between threads

[1]For a complete description of OneMax and Sphere Model problems the reader is referred to Sections 3.3.3 and 3.4.3 respectively.

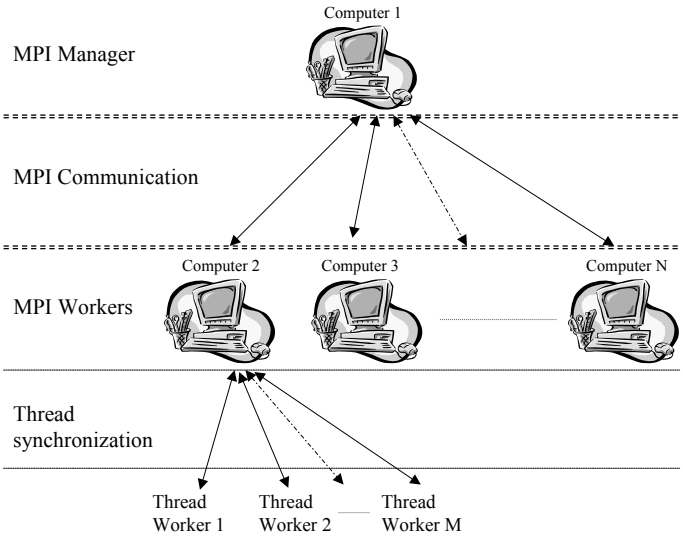


Figure 3.1: Two-level Manager-Worker scheme.

has to be explicitly implemented.

For the implementation of the programs we have chosen a two-level Manager-Worker scheme (Figure 3.1). At the first level (communication between computers) the communication is carried out using MPI, where a computer acts as a manager that distributes workload among the remaining computers (workers). These workers execute orders sent by the manager and return results to it. In some situations the manager may stay idle for a long time, waiting until all the workers finish their work. To make good use of this time, it is possible to make the manager act as another worker. At the second level, we present another Manager-Worker scheme (inside each computer), where the MPI process (one per computer) is the manager, and the workers are the multiple threads that can be created. This is a good choice when computers are multiprocessors. For reasons that will be explained later, the implementation of the parallel EBNA$_{PC}$ algorithm is made using a single-level Manager-Worker scheme (based on MPI).

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

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

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

3.3 Discrete domain

3.3.1 EBNA$_{BIC}$ algorithm

Algorithm description

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

EBNA$_{BIC}$ uses the penalized maximum likelihood score denoted by BIC (Bayesian Information Criterion) [183]. Given a structure S and a dataset D (set of selected individuals), the BIC score can be written as:

$$BIC(S,D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{1}{2}\log(N)\sum_{i=1}^{n} q_i(r_i-1) \tag{3.1}$$

where:

- N is the number of selected individuals.
- n is the number of variables of the Bayesian network (size of the individual).
- q_i is the number of different values that the parent variables of X_i, $\boldsymbol{Pa_i}$, can take.
- r_i is the number of different values that variable X_i can take.
- 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.
- N_{ij} is the number of individuals in D in which variables $\boldsymbol{Pa_i}$ take their j^{th} value.

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

$$BIC(S,D) = \sum_{i=1}^{n} BIC(i,S,D) \tag{3.2}$$

where

$$BIC(i,S,D) = \sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{1}{2}\log(N)q_i(r_i-1) \tag{3.3}$$

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, provided that it results in a Directed Acyclic Graph (DAG) structure (note that the structure of a Bayesian network must always be a DAG). This process continues until

```
Step 1.  for i = 1, ..., n calculate BIC[i]
Step 2.  for i = 1, ..., n and j = 1, ..., n
             if (i ≠ j) calculate G[j, i]
Step 3.  Find (j, i) such that paths[i, j] = 0 and G[j, i] ≥ G[r, s]
         for each r, s = 1, ..., n such that paths[s, r] = 0
Step 4.  if G[j, i] > 0
             update S with arc modification (j, i)
             update paths
         else stop
Step 5.  for k = 1, ..., n
             if (k ≠ i and k ≠ j) calculate G[k, i]
Step 6.  go to Step 3
```

Figure 3.2: Pseudo-code for the sequential structural learning phase, EBNA$_{BIC}$.

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), that is $j \rightarrow i$, 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 BIC score is calculated for each node and then the change in the BIC score is calculated for each possible arc modification. Thus, we have to calculate $n(n-1)$ terms, as there are $n(n-1)$ possible arc modifications. The arc modification that maximizes the gain of the BIC score, whilst maintaining the DAG structure, is applied to S. In the remaining steps, only changes to the BIC score due to arc modifications related to the variable X_i need to be considered (it is assumed that in the previous step, S was updated with the arc modification (j, i)). Other arc modifications have not changed its value because of the decomposable property of the score. In this case, the number of terms to be calculated is $n-2$.

We use four 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).

- A matrix $paths[i, j]$, $i, j = 1, 2 \ldots, n$, of dimension $n \times n$ that represents the number of paths between each pair of vertices (variables). This structure is used to check if an arc modification produces a DAG structure. For instance, it is possible to add the arc (j, i) to the structure if the number of paths between i and j is equal to 0, that is, $paths[i, j] = 0$.

A pseudo-code for the sequential structure learning algorithm can be seen in Figure 3.2.

Parallel approach. First version

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

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

The pseudo-code for this algorithm is shown in Figures 3.3 (manager) and 3.4 (workers).

Parallel approach. Second version

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

The first idea is to send to the workers the data structure needed for them to check if the DAG property is still maintained (*paths*). However, this alternative is not efficient, because even if work is distributed among the workers as $numvariables/numworkers$, it is impossible to know how many BICs must be calculated at each worker (only those that kept the DAG property). Therefore, notable differences in computational effort could exist between different workers (unbalanced distribution) and the manager must wait until the last of them finishes.

For this reason, it is necessary to find out in advance the number of edges that fulfill the restriction and, therefore, before sending the work, the manager tests for all the possible changes that maintain the DAG condition, creating a set with those. Then, this set is sent to the workers, and each of them calculates the BIC criterion for its proportional subset, guaranteeing an equally distributed workload. It must be noted that this second version

```
Step 1.  Send D to the workers
         set the number of variables (NSet) to work with
Step 2.  Send "calculate BIC" order to the workers
Step 3.  Receive BIC results from workers
Step 4.  for i = 1, ..., n
             send "calculate G[k, i]" order to the workers
             send i, BIC[i] to the workers
Step 5.  Receive from workers all the changes and update G
Step 6.  Find (j, i) such that paths[i, j] = 0 and G[j, i] ≥ G[r, s]
         for each r, s = 1, ..., n such that paths[s, r] = 0
Step 7.  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 8.  Send "calculate G[k, i] for (j,i)" order to the workers
Step 9.  Receive from workers all the changes and update G
Step 10. go to Step 6
```

Figure 3.3: Pseudo-code for the parallel structural learning phase. First version of the EBNA$_{BIC}$ algorithm for the manager.

requires an additional effort in the manager: look for the valid movements (those that maintain the DAG property) and send to the workers the set of valid movements. Therefore, this version may take a longer execution time than the previous one if the structure of the created Bayesian network is very simple.

This modifications have been also introduced in the sequential version, creating a different sequential program. This is necessary in order to fairly measure the efficiency and scalability of the parallel algorithm compared to the sequential version.

Figure 3.5 (manager) shows the changes made for this second version. Code for the workers is not shown due to its similarity with the first version: the single difference is that in this version the workers explicitly receive the set of variables to work with, instead of calculating the set themselves.

3.3.2 EBNA$_{PC}$ algorithm

Algorithm description

Like the previous algorithm, EBNA$_{PC}$ also follows the common scheme 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 the final step, a Bayesian network is created based on them.

EBNA$_{PC}$ receives its name from the fact that the algorithm used to detect the conditional (in)dependencies is the *PC* algorithm, described in [198]. Like most structure recovery algorithms based on independence detections, the PC algorithm starts by creating the complete undirected graph G, then "thins" that graph by removing edges with zero order conditional independence relations, "thins" again with first order conditional relations, then with second

```
Step 1. Receive D
        set the number 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]"
               receive i, BIC[i]
               for each variable k in NSet
                  if (k ≠ i) calculate G[k,i]
               send G modifications to the manager
            "calculate G[k,i] for (j,i)"
               for each variable k in NSet
                  if (k ≠ i and k ≠ j)calculate G[k,i]
               send G modifications to the manager
            "change arc (j,i)"
               Update S with (j,i) arc modification
            "stop" stop
Step 4. go to Step 2
```

Figure 3.4: Pseudo-code for the parallel structural learning phase. First version of the EBNA$_{BIC}$ algorithm for the workers.

```
Step 4. for i = 1,...,n
           calculate the structures that fit the DAG property
           set the vector of variables (NSetDAG) to work with
           send "calculate G[k,i]" order to the workers
           send i, BIC[i] to the workers
           send NSetDAG to the workers
Step 8. Calculate the structures that fit the DAG property
        set the vector of variables (NSetDAG) to work with
        send "calculate G[k,i] for (j,i)" order
        send NSetDAG to the workers
```

Figure 3.5: Pseudo-code for the parallel structural learning phase. Second version of the EBNA$_{BIC}$ algorithm for the manager. Only the steps that differ from the previous version are shown.

order conditional relations, and so on. The set of variables conditioned on need only to be a subset of the set of variables adjacent to one of the variables of the pair. The independence test is performed based on the χ^2 distribution. When there are no more tests to do, the orientation process begins, giving a direction to each edge in G.

The data structures used in this program are:

- A matrix G to store the undirected graph structure, where $G[i, j]$ is *true* when there is an edge connecting i and j nodes.
- A structure *SepSet* to save the independence relation between two variables and the

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
save $S(X_i, X_j)$ 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

Figure 3.6: Pseudo-code for the sequential structural learning phase, EBNA$_{PC}$.

set of variables conditioned on.

Figure 3.6 shows the pseudo-code for the PC algorithm. $Adj(G, A)$ represents the set of vertices adjacent to the vertex A in the undirected graph G, and $I(X_i, X_j | S(X_i, X_j))$ indicates that X_i and X_j are independent given a subset of adjacent variables $S(X_i, X_j)$. Note that the graph G is continually updated, so $Adj(G, A)$ 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 [199].

Parallel approach. First version

We performed a study of execution times of the sequential algorithm. As shown in Table 3.1, the learning phase requires nearly 98% of total execution time, so this is the obvious target for parallelization. As mentioned earlier, the learning phase can be divided into two different steps: detection of conditional (in)dependencies, and orientation. Table 3.2 shows that the first step takes about 99% of the learning time, therefore our parallel approach focuses only on this step.

TABLE 3.2: TIME MEASUREMENT (%) OF DIFFERENT STEPS OF THE EBNA$_{PC}$ LEARNING PHASE, FOR DIFFERENT SIZES OF THE *OneMax* PROBLEM.

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

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

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

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

Note that the parallel version must repeat exactly the same steps as the sequential algorithm. Starting with zero order dependencies, the pairs can be distributed between the manager and the workers without any additional control. When these computations have finished, the real "challenging" work begins. The manager waits for requests from workers asking for pairs to calculate. To determine whether a pair (X_i, X_j) is susceptible to be calculated, there 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 forces the worker to wait until a pair can be selected. This process is repeated until no more dependencies can be found.

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

The pseudo-code of this parallel version of the EBNA$_{PC}$ algorithm can be seen in Figure 3.7 (manager) and Figure 3.8 (workers).

Parallel approach. Second version

If we observe the operation mode of the parallel program when computing dependencies with adjacent sets of cardinality greater than zero, we can see that the manager selects the next pair to calculate and then sends it to the worker that has asked for a job, repeating this process for all the requests. While workers process their assigned pairs, the manager stays idle, waiting. An evident improvement here is to make the manager play also the role of a worker when there are not requests pending: the manager can take a new pair and calculate it itself. However, it must be noted that this approach can cause a reduction of the efficiency when the number of workers is increased. It could occur that the manager starts computing a pair (because there were no requests from workers) and, then, workers finish their jobs while the manager is still computing. In this situation, the manager would be blocking one or more workers, avoiding a efficient use of the available CPUs.

In Figure 3.9 the modifications made in the manager version of the parallel program can be seen. The worker's algorithm is not repeated again because it is the same of the first version.

3.3.3 Experiments in the discrete domain

This section reports the different experiments that have been carried out to assess the performance of the different implementations of the algorithms, on several computing platforms. The results of the experiments are presented from the point of view of speed up and scalability of the parallel algorithms.

With regard to the general steps of EDAs, there is one phase, evaluation of the individuals, which uses different fitness functions depending on the particular target problem. Therefore, the time needed to evaluate all the individuals can vary significantly from one problem to other. As our purpose is to give a general view of the parallel approaches, we have decided to select simple problems (with very simple fitness functions) in order to study the behavior of the parallel algorithms independently of the evaluation phase. As mentioned before, a deeper analysis of the EBNA$_{BIC}$ algorithm using a more complex problem will be presented in the following chapter.

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

$$OneMax(\boldsymbol{x}) = \sum_{i=1}^{n} x_i \tag{3.4}$$

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

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

The machine used to carry out the experiments –which is the same for discrete and continuous domains–, is a cluster of 10 nodes, where each node has two AMD ATHLON MP 2000+ processors (CPU frequency 1.67GHz), with 256KB of cache memory per processor, and 1GB of RAM per node. The operating system is GNU-Linux and the MPI implementation is LAM (version 6.5.9). All the computers are interconnected using a switched Gigabit Ethernet network.

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 be sent
        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, r values to the worker
        if it is not possible to send a pair (due to priority)
            do not send anything to the 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

Figure 3.7: Pseudo-code for the parallel structural learning phase. First version of the EBNA$_{PC}$ algorithm for the manager.

```
Step 1.  Form the complete undirected graph G on
         vertex set V = {X_1, ..., 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) to work with
Step 5.  for each pair (X_i,X_j) in PairsSet
             test conditional dependencies
             if "test fulfilled" save (i, j) edge in Deleted_edges
Step 6.  Send to the manager edges in Deleted_edges
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
                 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
```

Figure 3.8: Pseudo-code for the parallel structural learning phase. First version of the EBNA$_{PC}$ algorithm for the workers.

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

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

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

On the subject of the parameters chosen to test program performance, we have selected a medium-large size Bayesian network: an individual size of 500 (for both algorithms, EBNA$_{BIC}$ and EBNA$_{PC}$). For the remaining parameters, these are the default values: size of the population is 2,000; 1,999 new individuals are created each new generation; from those, the best 1,000 are selected. The stopping criterion is the completion of a fixed number of generations: 15 for both algorithms. The reason for this is that, due to the variable behavior of EDAs, execution time can vary from one execution to other if the algorithm is stopped when a particular result is obtained. Therefore, fixing the number of generations we get easily comparable execution times for all program versions. As the base algorithms are the same (the sequential versions) we cannot claim (in fact, we do not want to do so) any improvement in the quality of the results, just in the execution time.

```
Step 9. repeat
            for each worker w without work
                search for a pair to be sent
                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, r values to the worker
                if it is not possible to send a pair (due to priority)
                    do not send anything to the 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
```

Figure 3.9: Pseudo-code for the parallel structural learning phase. Second version of the EBNA$_{PC}$ algorithm for the manager. Only the steps that differ from the previous version are shown.

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

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

Time-related dimension

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

- Speed up = $\frac{sequential\ time}{parallel\ time}$
- Efficiency = $\frac{speed\ up}{number\ of\ processors}$

TABLE 3.3: TIME-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $EBNA_{BIC}$ PARALLEL ALGORITHM.

First version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	2h 51' 40"	-	-	-	-	-
2	1h 08' 44"	2.50	1.25	1h 08' 13"	2.52	1.26
6	24' 38"	6.97	1.16	24' 23"	7.04	1.17
10	15' 21"	11.18	1.12	15' 32"	11.05	1.11
20				08' 54"	19.29	0.96

Second version

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

Tables 3.3[2] and 3.4 summarize the results of the experiments for the $EBNA_{BIC}$ and $EBNA_{PC}$ algorithms respectively.

It can be observed that results obtained using pure MPI and MPI&Threads are very similar. This shows that these algorithms can run efficiently on clusters of single-processors as well as on clusters of multiprocessors. In particular, it must be pointed out the excellent efficiency of both versions of the $EBNA_{BIC}$ algorithm, even when using 20 CPUs. As we expected, the second parallel version reaches shorter computation times because only the scores actually needed are calculated. It could be surprising to observe efficiency reaching values over 1, but it must be taken into account that in the parallel executions more processors are used, meaning that more data structures fit into their ultra-fast, on-chip cache memories.

With regard to the $EBNA_{PC}$ algorithm, promising results are obtained using few CPUs, but when 10 or more processors are used the parallel algorithm shows itself quite inefficient. It must be pointed out that a lot of communication/synchronization is required in this algorithm, being necessary a continuous work distribution as well as an update of common data structures. So, the more workers are used, the more time is needed for communication nullifying the addition of more CPU capacity.

There are significant differences between the two $EBNA_{PC}$ parallel versions when using a small number of CPUs. This is because in the first version the manager only receives the results from the workers and sends them new work. Therefore, when few workers are available, the manager spends a significant portion of its time waiting for them to finish. In the second version the manager makes use of this idle time to act as a worker. Logically, as the number of CPUs increases, the difference between both versions reduces.

[2]Take note that both $EBNA_{BIC}$ sequential versions are different and, therefore, their execution times differ too.

TABLE 3.4: TIME-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE EBNA$_{PC}$ PARALLEL ALGORITHM.

First version

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

Second version

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

Efficiency-related dimension

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

In Figures 3.10, 3.11, 3.12 and 3.13 total execution time has been split in parts to better observe program (manager and workers) behavior. Legend "Sequential" shows the portion of time required by those parts of the code that have not been parallelized. The remaining legends differentiate time spent in computation ("Execution") from time spent in communication and synchronization ("Comm&Syn"), for manager and workers. Tables 3.5 and 3.6 extend this information, presenting actual execution times, with deviations for the case of workers. Some interesting conclusions can be presented for the EBNA$_{BIC}$ algorithm inspecting these figures and tables:

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

For the EBNA$_{PC}$ algorithm, we observe that:

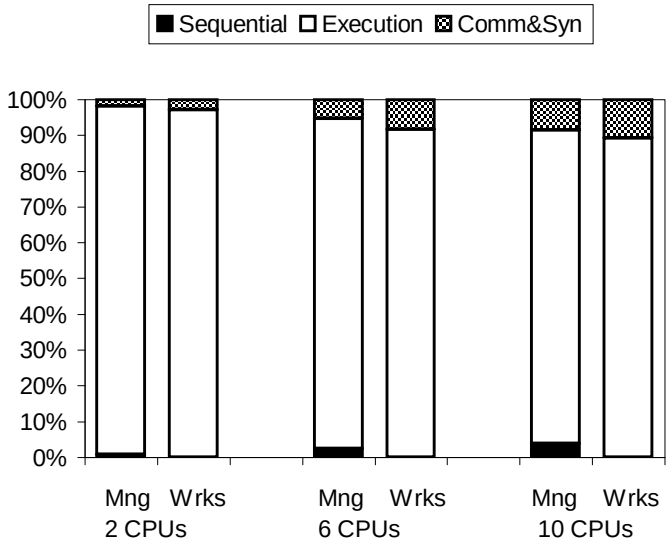


Figure 3.10: Detail of the computation time for the first version of the EBNA$_{BIC}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

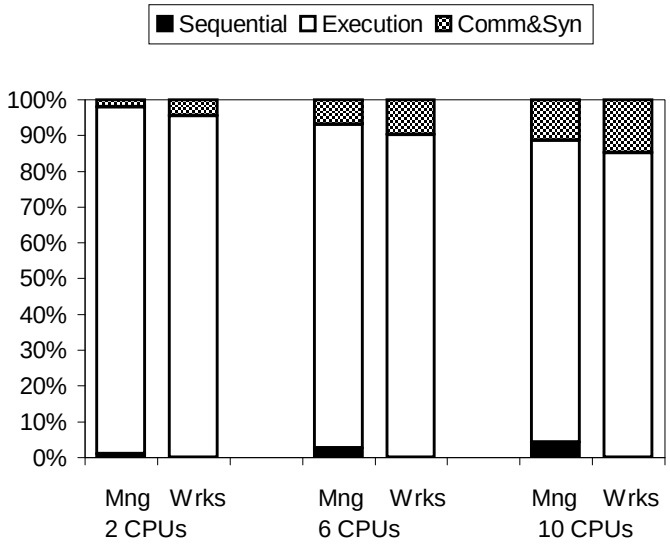


Figure 3.11: Detail of the computation time for the second version of the EBNA$_{BIC}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

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

- Communication and synchronization requirements make this algorithm quite inefficient as the number of CPUs increases. In the first version, as the manager only distributes the work, when using few CPUs it spends a substantial time waiting for the workers to request additional pieces of work. However, when using too many workers (10 or

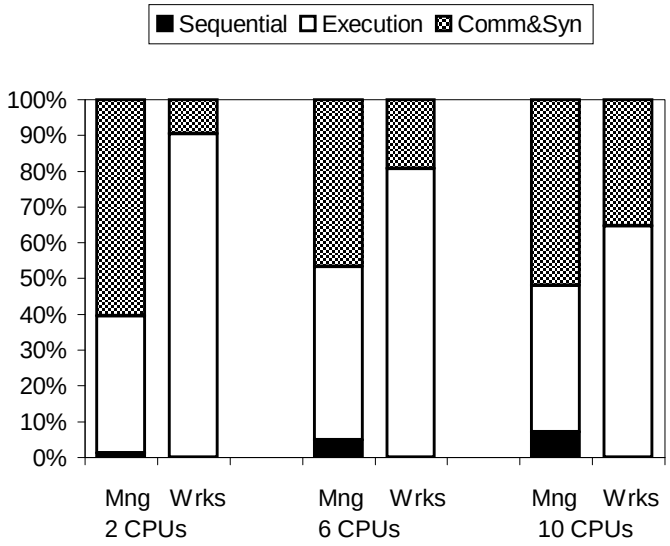


Figure 3.12: Detail of the computation time for the first version of the EBNA$_{PC}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

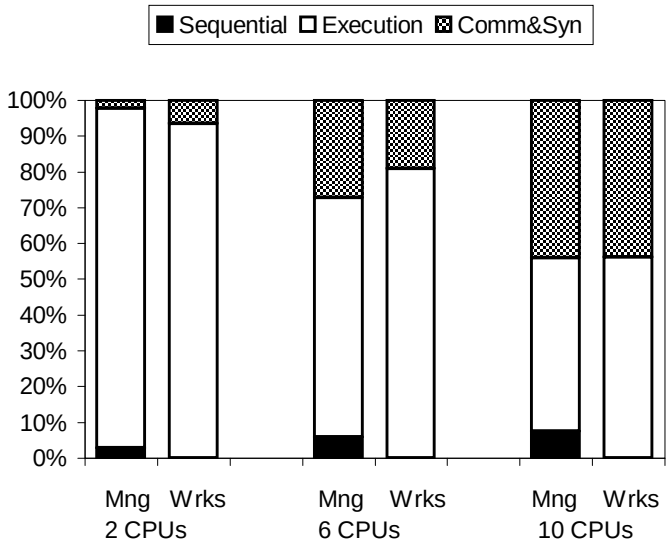


Figure 3.13: Detail of the computation time for the second version of the EBNA$_{PC}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

more) manager is not able to attend workers fast enough, so workers waste time while waiting for more to do.

- In the second version, the manager acts also as a worker, making use of idle times and therefore the communication and synchronization time for the manager reduces drastically. This works fine for small numbers of workers but, as occurs with the first version, when this number is larger blocking situations arise at the worker side.

TABLE 3.5: EFFICIENCY-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE EBNA$_{BIC}$ ALGORITHM. EXECUTION TIMES FOR MANAGER AND WORKERS.

First version

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

Second version

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

TABLE 3.6: EFFICIENCY-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSION OF THE EBNA$_{PC}$ ALGORITHM. EXECUTION TIMES FOR MANAGER AND WORKERS.

First version

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

Second version

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

3.4 Continuous domain

3.4.1 EGNA$_{BIC}$ algorithm

Algorithm description

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

also valid for this algorithm.

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

$$BIC(S,D) = \sum_{r=1}^{N}\sum_{i=1}^{n}[-\frac{1}{2}\ln(2\pi v_i) - \frac{1}{2v_i}(x_{ir} - m_i - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j))^2] \\ -\frac{1}{2}\log(N)(2n + \sum_{i=1}^{n} \mid \boldsymbol{Pa}_i \mid) \qquad (3.5)$$

where

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

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

$$BIC(S,D) = \sum_{i=1}^{n} BIC(i,S,D) \qquad (3.6)$$

where

$$BIC(i,S,D) = \sum_{r=1}^{N}[-\frac{1}{2}\ln(2\pi v_i) - \frac{1}{2v_i}(x_{ir} - mi - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j))^2] \\ -\frac{1}{2}\log(N)(2+ \mid \boldsymbol{Pa}_i \mid) \qquad (3.7)$$

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

Parallel approach

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

Due to the similarity of this algorithm to the discrete one (EBNA$_{BIC}$) we consider unnecessary to explain the entire process again. To obtain a complete view of the characteristics of the algorithm and the parallel solution, see Section 3.3.1.

3.4.2 EGNA$_{EE}$ algorithm

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 complete graph, in which 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 [195]. The statistic to exclude the edge between X_i and X_j 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 [209] 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.

In [195], the authors obtained the density and distribution functions of the likelihood ratio test statistic under the null hypothesis. These expressions are of the form:

$$f_{lik}(t) = g_\chi(t) + \frac{1}{4}(t-1)(2n+1)g_\chi(t)N^{-1} + O(N^{-2}) \tag{3.8}$$

$$F_{lik}(x) = G_\chi(x) - \frac{1}{2}(2n+1)xg_\chi(x)N^{-1} + O(N^{-2}) \tag{3.9}$$

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, its parameters have to be learnt to complete the Gaussian network. This can be done using the following formulas:

$$\hat{m}_i = \overline{X}_i \tag{3.10}$$

$$\hat{b}_{ji} = \frac{\hat{\sigma}_{ji}}{\hat{\sigma}^2_{jj}} \tag{3.11}$$

$$\hat{v}_i = \hat{\sigma}^2_{ii} - \sum_{X_j \in \boldsymbol{Pa}_i} \frac{\hat{\sigma}_{ji}}{\hat{\sigma}^2_{jj}} + 2 \sum_{X_j \in \boldsymbol{Pa}_i} \sum_{X_k \in \boldsymbol{Pa}_{i_{k>j}}} \frac{\hat{\sigma}_{jk}\hat{\sigma}_{ji}\hat{\sigma}_{ki}}{\hat{\sigma}^2_{jj}\hat{\sigma}^2_{kk}} \tag{3.12}$$

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

Six major structures are needed to implement this algorithm:

- The graph structure, 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 covariance 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.
- 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 pseudo-code for the sequential structure learning algorithm is presented in Figure 3.14.

```
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, ..., n
            for j = i + 1, ..., n
                calculate r_{ij|rest}
                if "test fulfilled"
                    update S with (i, j) arc deletion
Step 5. Calculate the parameters m, b and v
```

Figure 3.14: Pseudo-code for the sequential learning phase, EGNA$_{EE}$.

TABLE 3.7: TIME MEASUREMENT (%) OF DIFFERENT PHASES OF THE EGNA$_{EE}$ ALGORITHM.

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

Parallel approach. First version

In Table 3.1 (presented in Section 3.2), the execution time required by the learning phase have been presented for different individual sizes. It can be observed that the learning phase is not as time-consuming as in previous algorithms. Due to this, a deeper analysis has been done to know how the computation time is distributed among the different procedures. Table 3.7 shows the most important procedures that are executed by the present algorithm: learning and "sampling and evaluation" of new individuals.

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

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, this phase completes the following steps:

1. Calculate means and Σ from the selected individuals.

2. Obtain the precision matrix.

TABLE 3.8: TIME MEASUREMENT (%) OF DIFFERENT STEPS OF THE EGNA$_{EE}$ LEARNING PHASE, FOR DIFFERENT SIZES OF THE *Sphere model* PROBLEM.

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

3. Compute the tests.

4. Carry out the parameter learning.

Table 3.8 shows the time consumed by each of these steps. Different measurements have been obtained taking into account different individual sizes and, although the percentages vary depending on the size of the individual, it can be observed that the most computationally expensive steps are the first (where means and Σ are calculated) and the fourth (where the parameters are learnt). In the second step –calculate the inverse of the covariance matrix– the LU decomposition method is used, and execution is very fast. The third step (test) is also rapidly executed because it uses the values computed in the previous phases to obtain the r_{ij} values. Therefore, our parallelizing efforts focus on steps one (59-30%) and four (30-64%).

To calculate Σ it is first necessary to obtain the means for each variable, so the database of cases is sent to each worker and the number of means to be calculated (number of variables) is distributed among all the workers (manager included). Being n the number of variables and *numworkers* the number of workers, each worker will have $n/numworkers$ 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 worker computes $(n(n-1)/2)/numworkers$ values of the matrix. Next, each worker sends its results to the manager and it updates the Σ structure, continuing with the sequential processes: calculate the precision matrix (inverse of Σ) and compute the tests.

Finally, the parameters must be learnt and, again, a correct workload distribution has to be done. Due to the initial structure of the Gaussian network, the variables with lower index can have potentially more dependencies and, thus, need more time to compute their respective $\boldsymbol{b}$ and $\boldsymbol{v}$ values. Taking this characteristic into account, and trying to reduce as much as possible the communication and synchronization tasks, we decided to distribute the variables (work) in the following way:

- Create $s = \lceil n/numworkers \rceil$ sets.
- Each worker selects, based on its identifier i (between 0 and $numworkers - 1$), the i^{th} variable of the first subset, then the $(i-1)^{th}$ variable of the second subset and so on, until there are no subsets. When i is less than zero then it is set to $numworkers - 1$.

In this way, each worker takes a subset of variables, created following a even distribution (as can be checked looking at the obtained results). Other way to do this could be the use of the well-known on-demand scheme: each worker asks for a variable to compute and the manager attends those requests. However, as seen for the EBNA$_{PC}$ algorithm, this approach can require excessive communication times, reducing the efficiency.

Figures 3.15 and 3.16 show the pseudo-code for this first parallel version.

Parallel approach. Second version

In the previous algorithms we observed that the most time-consuming part of the programs was the learning phase, and we focused our efforts on it. However, for EGNA$_{EE}$, the "sampling and evaluation" phase also consumes a significant portion of time (30 to 50% for our simple target problem), and it is compulsory to work on it if we want an efficient and scalable parallel program. In fact, this situation appears in all EDAs that evaluate individuals using a complex function, or those that require a large population. Therefore, this idea can be

```
Step 1.  Initialize the structure of the Gaussian network S
Step 2.  Send D structure to the workers
         set the number of variables (NSet) to work with
Step 3.  Send "calculate means" order to the workers
         Synchronize with all workers and update m
Step 4.  Send "calculate covariance" order to the workers
         Synchronize with all workers and update Σ
Step 5.  Calculate the inverse of Σ
Step 6.  for i = 1, ..., n
             for j = i + 1, ..., n
                 calculate r_{ij|rest}
                 if "test fulfilled"
                     update S with (i, j) arc deletion
Step 7.  Send "calculate parameters" order to the workers
         Send S to the workers
         Receive from the workers all the changes and update b and v
Step 8.  Send "stop" order to the workers
```

Figure 3.15: Pseudo-code for the parallel learning phase. First version of the EGNA_{EE} algorithm for the manager.

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

Figure 3.16: Pseudo-code for the parallel learning phase. First version of the EGNA_{EE} algorithm for the workers.

Step 1. **for** each i in $NewPopSet$
obtain and evaluate a new individual
add the individual to the population

Figure 3.17: Pseudo-code for the sequential "sampling and evaluation" phase, EGNA$_{EE}$.

Step 1. Send S, $Order$, m, b and v structures to the workers
Step 2. Receive the new individuals from the workers
update the population

Figure 3.18: Pseudo-code for the parallel "sampling and evaluation" phase. EGNA$_{EE}$ algorithm for the manager.

adapted to any of the algorithms presented in this chapter (in the next chapter we are going to do so for the EBNA$_{BIC}$ algorithm).

Following the general structure of EDAs, an initial population is created, the best N individuals are selected, a probabilistic model is induced (learning phase) and, finally, a new population is generated based on the induced model. For this last step, an adaptation of the Probabilistic Logic Sampling (PLS) proposed in [82] is used. In this method the instances are generated one variable at a time in a forward way. That is, a variable is sampled after all its parents have already been sampled. To do that an ancestral ordering of the variables is given $(\pi(1), \ldots, \pi(n))$ where parent variables are before children variables. Once the values of $\mathbf{Pa}_{\pi(i)}$ have been assigned, we simulate a value for $X_{\pi(i)}$, using the distribution $f(x_{\pi(i)}|\boldsymbol{pa}_{\boldsymbol{\pi(i)}})$. For the simulation of a univariate normal distribution, a simple method based on the sum of 12 uniform variables is applied [162].

A pseudo-code for this "sampling and evaluation" process can be seen in Figure 3.17.

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

Figures 3.18 (manager) and 3.19 (workers) show the parallel proposal for the "sampling and evaluation" phase.

Step 1. Receive S, $Order$, m, b and v structures
Step 2. **for** each i in $NewPopSet$
obtain and evaluate a new individual
Step 3. Send the new individuals to the manager

Figure 3.19: Pseudo-code for the parallel "sampling and evaluation" phase. EGNA$_{EE}$ algorithm for the workers.

3.4.3 Experiments in the continuous domain

The scenario used for evaluations in the continuous domain is the same used for the discrete one, so detailed information can be obtained in Section 3.3.3.

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

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:

$$Sphere(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \tag{3.13}$$

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

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

- EGNA$_{BIC}$: The size of the individual is 100. 2,000 is the size of the population. 1,999 new individuals are created in each generation and the best 1,000 are selected. Execution is stopped when the 10th generation is reached.

- EGNA$_{EE}$: 1,500 has been selected as individual-size. The population has 6,000 individuals, 5,999 new ones being created in each generation. Then, the best 3,000 are selected. Execution is stopped when the 15th generation is reached.

Time-related dimension

In Tables 3.9[3] and 3.10 the execution times, speed up and efficiency are presented.

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

For the first version of EGNA$_{EE}$, it can be observed that parallelizing only the learning phase is not enough to obtain an efficient parallel algorithm. As the "sampling and evaluation" phase requires around the 30% of the total execution time, when 6 or more CPUs are used, this phase becomes an obvious bottleneck. In the second version, the mentioned "sampling and evaluation" phase is also parallelized, and the efficiency improves noticeably.

[3]Take note that both EGNA$_{BIC}$ sequential versions are different and, therefore, their execution times differ too.

TABLE 3.9: TIME-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $EGNA_{BIC}$ PARALLEL ALGORITHM.

First version

CPUs	*MPI version*			*MPI&Threads version*		
	Time	*SpdUp*	*Effic*	*Time*	*SpdUp*	*Effic*
Seq.	4h 06' 24"	-	-	-	-	-
2	1h 48' 24"	2.27	1.14	1h 52' 03"	2.20	1.10
6	39' 01"	6.32	1.05	40' 21"	6.11	1.02
10	24' 12"	10.18	1.02	25' 09"	9.80	0.98
20				14' 07"	17.45	0.87

Second version

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

TABLE 3.10: TIME-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $EGNA_{EE}$ PARALLEL ALGORITHM.

First version

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

Second version

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

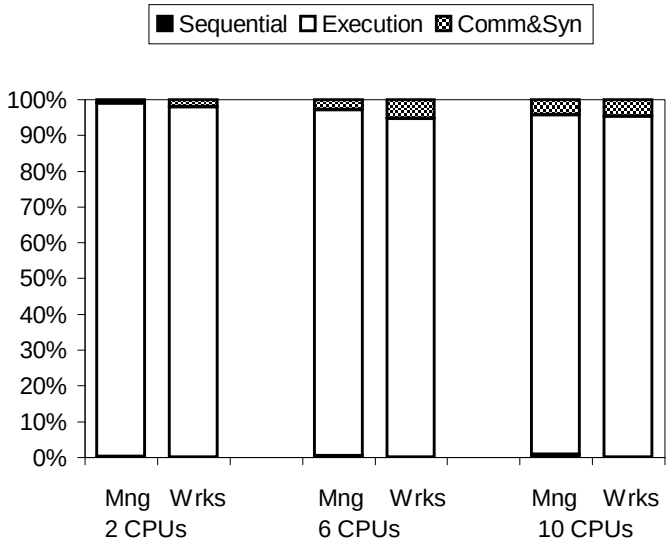


Figure 3.20: Detail of the computation time for the first version of the EGNA$_{BIC}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

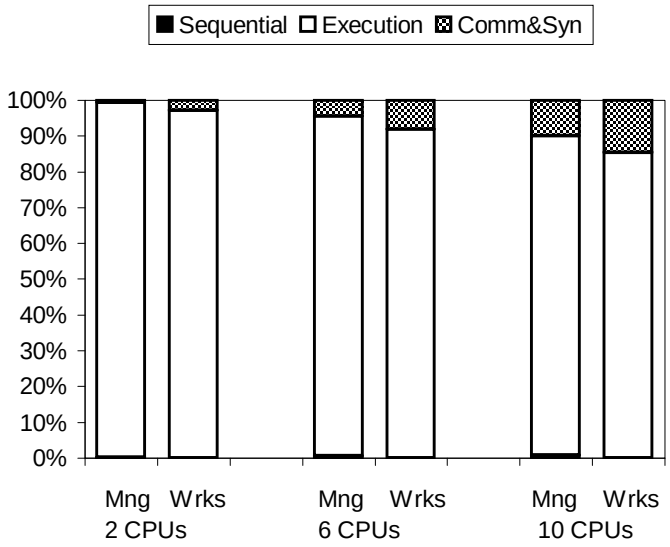


Figure 3.21: Detail of the computation time for the second version of the EGNA$_{BIC}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

Efficiency-related dimension

Performance results are summarized in Figures 3.20, 3.21, 3.22 and 3.23 and Tables 3.11 and 3.12. They show detailed information about execution times, exposing the percentage that each different section requires, as well as absolute timing values and deviations for workers.

As can be observed, the behavior of the EGNA$_{BIC}$ algorithm is similar to the EBNA$_{BIC}$

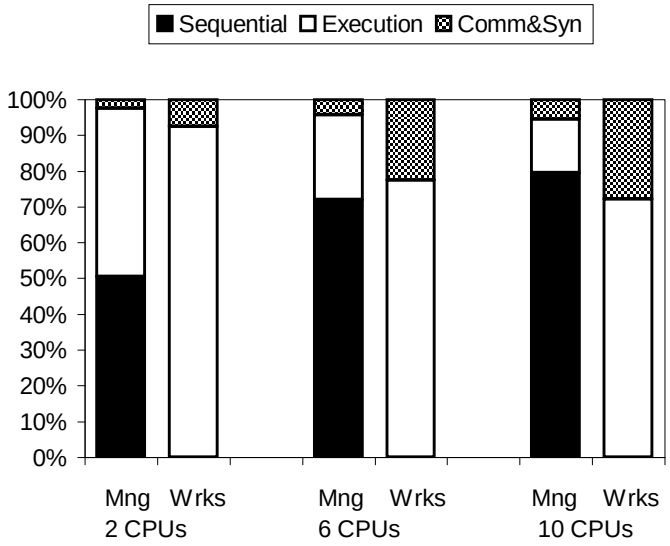


Figure 3.22: Detail of the computation time for the first version of the EGNA$_{EE}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

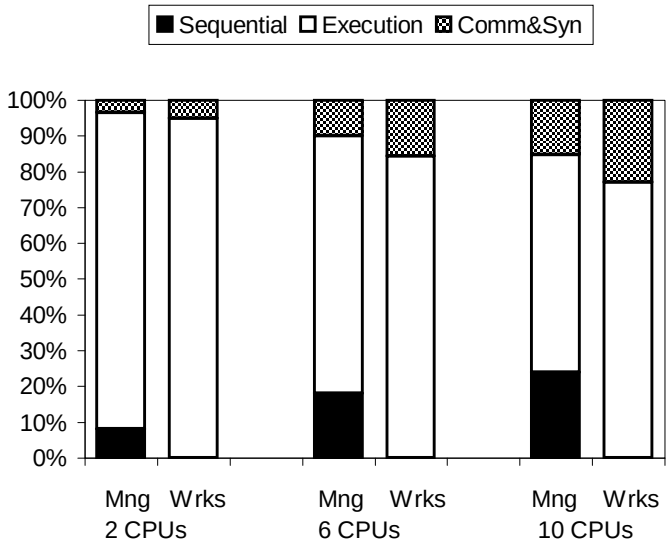


Figure 3.23: Detail of the computation time for the second version of the EGNA$_{EE}$ algorithm, using a pure MPI implementation. 2, 6 and 10 CPUs have been used.

algorithm and therefore, conclusions are the same: the algorithm presents a good scalability because the time spent in the sequential step is quite low and in addition it is not required too much communication. These conclusions can be consulted in Section 3.3.3.

For both versions of EGNA$_{EE}$ algorithm, we reach these conclusions:

- In the first version, where the "sampling and evaluation" phase has not been parallelized, the efficiency decreases notably when more CPUs are used, as it could be expected from the distribution of execution times for the different phases of the se-

TABLE 3.11: EFFICIENCY-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $EGNA_{BIC}$ ALGORITHM. EXECUTION TIMES FOR MANAGER AND WORKERS.

First version

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

Second version

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

TABLE 3.12: EFFICIENCY-RELATED EXPERIMENTAL RESULTS FOR BOTH VERSIONS OF THE $EGNA_{EE}$ ALGORITHM. EXECUTION TIMES FOR MANAGER AND WORKERS.

First version

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

Second version

	Manager			*Workers*	
CPUs	*Seq.*	*Exec*	*Comm*	*Exec*	*Comm*
2	8’ 42”	1h 34’ 47”	3’ 38”	1h 33’ 26”	4’ 59”
6	8’ 42”	34’ 31”	4’ 43”	33’ 08”±17”	6’ 06”
10	8’ 42”	22’ 12”	5’ 35”	21’ 25”±05”	6’ 22”

quential algorithm. This is the reason why the second version (where this phase has been parallelized) performs better.

- In general, the scalability of the algorithm is poor. This is due to the high communication requirements –different calculations that require continuous synchronization– and to the portion of the algorithm that has not been parallelized (the sequential part), which becomes more prominent when the number of processors is increased.

3.5 Summary

In this chapter several parallel solutions have been proposed for four different EDAs. For each one, a previous study of the execution times of the procedures that conform the sequential

algorithm was made. In most of the cases, the step that requires the most substantial part of the total execution time is the learning phase, where a probabilistic graphical model is induced from the database of individuals. In the continuous domain section, we have seen that, depending on the algorithm, there may be other steps that also take up much computational time –in particular, the sampling and creation of new individuals once the structure has been learnt.

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

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

In fact, in the next chapter, the second parallel approach of EBNA$_{BIC}$ will be modified, parallelizing the "sampling and evaluation" phase due to the problem that is used. In addition, the efficiency study done in this chapter will be extended using both programming paradigms (MPI and threads) and testing the behavior of the algorithm in different architectures (clusters of computers and multiprocessors).

Chapter 4

Performance evaluation of a parallel implementation of EBNA$_{BIC}$

In this chapter, an exhaustive evaluation of the parallel EBNA$_{BIC}$ algorithm is presented. This evaluation has been done from the point of view of performance and scalability, using three different architectures: two clusters of computers and one multiprocessor.

A complex problem, Feature Subset Selection (FSS) for classification, has been chosen to assess the behavior of the algorithm. Taking into account that the evaluation of the individuals requires a significant percentage of the total execution time, the parallel version of EBNA$_{BIC}$ presented in the previous chapter has been modified, allowing it to process in a parallel way the "sampling and evaluation" phase (as done in the second parallel version of EGNA$_{EE}$).

Results show that the algorithm has good scalability properties and that, for this particular problem, it is capable of obtaining good levels of performance when running on a cluster of personal computers.

4.1 Introduction

In the previous chapter, parallel approaches have been presented for four EDAs, in both discrete and continuous domains. It has been seen that parallelism is really useful when these algorithms are applied to large problems, reducing notably the execution time. In addition, most of the parallel approaches scale satisfactorily, making good use of small-size, affordable parallel computers.

As different machines are nowadays available for intensive computing tasks, we decided to do some additional experimental work testing a range of machines and architectures in order to study their performance characteristics when running the kind of problems we target.

To carry out these experiments, we focused on one particular algorithm: parallel EBNA$_{BIC}$ (second approach). Regarding the problem, we decided to put the algorithm in a more realistic situation using a more complex fitness function: FSS [109] problem for classification.

The rest of this chapter is organized as follows: a preliminary analysis of the sequential version and the changes introduced in the parallel version are presented in Section 4.2. Section 4.3 discusses the performance of the parallel approach and, finally, conclusions are presented in Section 4.4.

Step 1. **for** $i = 1, \ldots, M$
create a new individual, X_i=(X_i^1,X_i^2,...,X_i^n)
evaluate X_i
Step 2. Select the best N individuals
Step 3. Learn a Bayesian network from the previously selected individuals
Step 4. Generate a new population by sampling the Bayesian network
evaluate each individual
Step 5. **Go to** Step 2 until a stop criterion is met

Figure 4.1: Pseudo-code for the sequential EBNA$_{BIC}$ algorithm.

4.2 Parallel EBNA$_{BIC}$ and the FSS problem

In order to check if the parallelization of an algorithm is viable, its behavior must be studied carefully. As a remainder, in Figure 4.1 the pseudo-code of the EBNA$_{BIC}$ algorithm is shown. It can be observed that, once the initial population is created, steps 2 and 3 are repeated until the algorithm converges.

We have already stated that this kind of algorithms has been designed to solve complex optimization problems. Depending on the characteristics of the particular target problem, some steps of the algorithm may require more execution time than others. For instance, the longer the number of variables that constitute the individual $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$, the harder the creation of the Bayesian network.

This issue has been considered in the previous chapter using a simple function ($OneMax$), observing that the learning phase requires almost all the execution time. However, in other problems, in which the evaluation of the fitness function is more complex, the execution percentage required by each phase changes notably.

We decided, therefore, to complete a more exhaustive study testing our parallel algorithm in real problems. Attending to previous experiments using EDAs, in [90, 92] FSS problems for supervised classification are solved, obtaining good results and convergency rates (particularly for the EBNA$_{BIC}$ algorithm). In addition, the authors observe the excessive execution time required by the EBNA$_{BIC}$ algorithm when solving problems with more than two hundred variables. This seems to be an appropriate scenario to test the behavior and efficiency of the parallel EBNA$_{BIC}$.

The basic problem of supervised classification in data mining is related to the induction of a model that classifies a given object into one of several pre-defined classes. In order to induce the classification model, each object is described by a pattern of n features. Usually, objects have many features and data files are large. Due to this characteristic, the following question is formulated by researchers: "Are all the features useful for learning the classification model?" The FSS approach to this question could be presented as follows: "given a set of candidate features, select the best subset that really matters for constructing the classifier".

Two are the common approaches used in FSS: filter and wrapper. In the filter approach, variables are selected taking into account some intrinsic properties of the dataset. In contrast, the wrapper approach considers the final objective of the selected variables; for instance, in a classification problem, the wrapper approach evaluates each subset of variables by means of the accuracy of the classifier that is built with this subset of variables.

EDAs are used in this dissertation in the context of the wrapper approach, where a

TABLE 4.1: PROFILING OF THE SEQUENTIAL IMPLEMENTATION OF EBNA$_{BIC}$ FOR DIFFERENT PROBLEMS: PROPORTION OF EXECUTION TIME USED BY THE LEARNING AND "SAMPLING AND EVALUATION" PHASES.

Problem	*Indiv. size*	*% Learning*	*% Sampl. and Eval.*
$FSS_{Arrhythmia}$	279	82.6	17.3
$FSS_{InternetAdv.}$	1558	58.6	41.3

classifier is used to measure the quality of each individual (subset of features). In order to use EDAs, the problem must be characterized as an optimization one. As we have said before, the goal is to select, from the set of features that describe an object, the subset that provides the best classifier. Therefore, we define an individual as $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$, being n the number of features. Each variable X_i of the individual can take two possible values: 0 to indicate that the i^{th} feature has not been selected or 1 to indicate that it has been selected. Each individual is evaluated measuring the accuracy of the classifier. This classifier is generated using only the selected features of the individual. Usually, the quality of a classifier is measured in terms of the percentage of rightly classified instances (accuracy percentage).

We have observed the behavior of the sequential version of the EBNA$_{BIC}$ algorithm when solving two well-known datasets: `Arrythmia` and `Internet Advertisement` [18]. Table 4.1 shows the cost of the most important steps of this algorithm. It can be observed that, as the evaluation of the fitness function is complex, execution times are more evenly distributed between the learning of the Bayesian network and the "sampling and evaluation" of the new population. Consequently, a modification of the EBNA$_{BIC}$ algorithm described in the previous chapter is necessary in order to have an efficient parallel algorithm: in addition to the parallelization of the learning phase, it is necessary to parallelize the "sampling and evaluation" phase. To do that, we have adapted to EBNA$_{BIC}$ the parallel implementation described for the second version of the EGNA$_{EE}$ algorithm in Section 3.4.2.

It is necessary to bear in mind that the parallel version of EBNA$_{BIC}$ has been designed following the Manager-Worker scheme designed in the previous chapter. Two parallel computing paradigms have been used, MPI and POSIX threads. Therefore, depending on the particular architectures available, different combinations of these paradigms can be chosen.

4.3 Performance evaluation

In this section we evaluate our proposal of parallel implementation of EBNA$_{BIC}$ applied to a FSS problem. Particularly, we have chosen `Internet Advertisements`, a middle-large size problem from the UCI Machine Learning Repository [18].

It must be highlighted that among the results presented in this chapter we neither show accuracy percentages, nor other parameters that indicate the quality of the obtained classifier. Our aim is to focus only on the quality of our parallel solution in terms of execution time, trying to extract conclusions from the behavior of the developed parallel algorithm. In this way, we prove that this parallel version of EBNA$_{BIC}$ can be applied to FSS problems of medium to large sizes, extending the work in [92].

The `Internet Advertisements` database has 1,558 features, 2 classes, and 3,279 instances. In consequence, the size of the individual is 1,558 and each variable has two possible

values (0 or 1, that is, not selected or selected). For the EBNA$_{BIC}$ algorithm, we have fixed the rest of the parameters as follows: size of the population is 2,000, size of the selected population is 1,000 and 1,999 new individuals are sampled and evaluated in each new generation. These are commonly recommended values [103].

With regard to the fitness function, each individual will be evaluated using Naive Bayes, a simple but good supervised classification algorithm. In order to maintain the same fitness function chosen in the previously mentioned works [90, 92], a 10-fold cross-validation is used to estimate the accuracy of the classifier (fitness function). In this technique (k-fold cross-validation), the dataset is randomly divided in k pieces, using $k-1$ of them to train the model and one to test its goodness. The process is repeated k times, using each time different pieces for training and testing.

Three different machines have been available for experimentation. Two of them are clusters of PC-class computers, and the other is an 8-way multiprocessor. The main characteristics of these machines are presented below:

Athlon Cluster: A cluster of 8 nodes. Each node has two AMD ATHLON MP 2000+ processors (1.67GHz), with 256KB of cache memory each and 1GB of (shared) RAM. Operating system is Linux. C++ compiler is version 3.3.2 of GNU's gcc. MPI implementation is LAM (version 6.5.9). Nodes are interconnected using a switched Gigabit Ethernet network. This is the same cluster used in the experiments described in Chapter 3.

Xeon Cluster: A cluster of 4 nodes. Each node has two Intel Xeon processors (2.4GHz, hyper-threading enabled), with 512KB of cache memory each and 2GB of (shared) RAM. Operating system is Linux. MPI implementation is LAM (version 7.0.2). C++ compiler is version 8 of Intel's icc. Nodes are interconnected using a switched Gigabit Ethernet network. From the operating system point of view, each physical processor is seen as a dual processor (Linux actually presents each node as a 4-way multiprocessor). This is the way the operating system takes advantage of hyper-threading.

Altix: A SGI Altix 3300 system with 8 Intel Itanium-2 processors (1.3GHz), 3MB of cache memory each and 8GB of shared RAM. Operating system is Linux. MPI implementation is MPICH. C++ compiler is version 8 of Intel's icc. Processors use a proprietary SGI interconnection network that provides, at hardware level, a memory space shared among all processors. Message passing is also very efficiently supported.

Results are expressed in terms of the execution times of the different versions of the programs. For the comparisons to be fair, a common termination criterion must be established. We have decided to end programs after a fixed number of generations: 4.

Each experiment has been repeated 10 times. For the chosen termination criterion, variations of execution times are very small: the observed deviation is less than 2%; therefore, we consider that 10 executions are representative enough. In all tables we collect the mean of the 10 obtained values.

Experiments have been made using different combinations of parallel programming paradigms:

Pure MPI: All communications (either intra-node or inter-nodes) are performed via the explicit interchange of messages (MPI).

Table 4.2: Arrangement of processes/threads per node in the machines used in the experiments.

machine	*Pure MPI version*	*MPI&Threads version*
Athlon	2 single-threaded processes	1 process with 2 threads
Xeon	4 single-threaded processes	1 process with 4 threads
Altix	8 single-threaded processes	1 process with 8 threads

MPI&Threads: A single process is launched at each node. Communication and synchronization between processes is done via MPI. Each process creates a collection of threads that use shared variables to communicate and synchronize.

The baseline experiments for performance comparison have been completed using the sequential program in one node of each of the available machines. For the parallel experiments, the arrangements of processes and threads in the machines depend on the characteristics of each scenario. Details are provided in Table 4.2.

As explained before, workload distribution among processes is done by allotting "chunks" of work to each of them. In the case of using threads (which are pre-forked when process starts), an on-demand scheme is used: threads await for a piece of work, execute it, return results to the master thread, and then wait for another piece.

In terms of measurements of the experiments, we have focused on two interrelated values: efficiency of the parallel program, and load distribution among processes. We have also obtained some traces of the program's execution to illustrate the blocking times caused by processes waiting for the reception of messages.

4.3.1 Efficiency

The main goal of a parallel algorithm is to reduce the execution time as much as possible. In addition, scalability is also a highly desirable property, because it allows efficient execution even for a large number of processes.

We have measured the performance of our program in terms of speed up and efficiency (defined in Section 3.3.3), when running on the different platforms available. Table 4.3 summarizes the results obtained. It can be observed that the efficiency range (0.84-1.09) obtained in the different scenarios is remarkably high. This has been achieved because workload distribution is even and communication and synchronization are carefully tuned. Figure 4.2 shows a portion of the learning process where manager (first horizontal line, denoted as 0) and workers (following horizontal lines identified as 1,2 and 3) look for a new arc addition/deletion that improves the BIC score; communication and synchronization periods (those in dark color) are really short compared to those periods of useful work (those in light color).

It could be also surprising to see efficiencies over 1. However, as mentioned in the previous chapter, there are several causes of this super-linear speed up. Some of those are:

- A parallel computer does not only add CPU power: each CPU has its own, fast cache memory. A parallel program can store more of its data structures in the caches than its sequential counterpart, thus improving execution speeds.

Table 4.3: Measured performance for the different machines and implementations. Total execution time as well as speed up and efficiency is shown.

Athlon Cluster

CPUs	*Pure MPI version*			*MPI&Threads version*		
	Exec. time	*Speed up*	*Efficiency*	*Exec. time*	*Speed up*	*Efficiency*
Seq.	52h 34' 25"	-	-	-	-	-
2	24h 18' 32"	2.16	1.08	25h 07' 48"	2.09	1.05
4	12h 19' 31"	4.27	1.07	12h 50' 17"	4.10	1.02
8	6h 06' 30"	8.61	1.08	6h 26' 48"	8.16	1.02
16	3h 18' 57"	15.86	0.99	3h 25' 05"	15.38	0.96

Xeon Cluster

CPUs	*Pure MPI version*			*MPI&Threads version*		
	Exec. time	*Speed up*	*Efficiency*	*Exec. time*	*Speed up*	*Efficiency*
Seq.	21h 36' 17"	-	-	-	-	-
2	13h 09' 47"	1.64	0.82	9h 52' 33"	2.19	1.09
4	6h 44' 10"	3.21	0.80	5h 02' 33"	4.28	1.07
8	3h 32' 22"	6.10	0.76	2h 34' 48"	8.37	1.05

Altix Multiprocessor

CPUs	*Pure MPI version*			*MPI&Threads version*		
	Exec. time	*Speed up*	*Efficiency*	*Exec. time*	*Speed up*	*Efficiency*
Seq.	41h 16' 02"	-	-	-	-	-
8	5h 29' 53"	7.51	0.94	6h 07' 40"	6.73	0.84

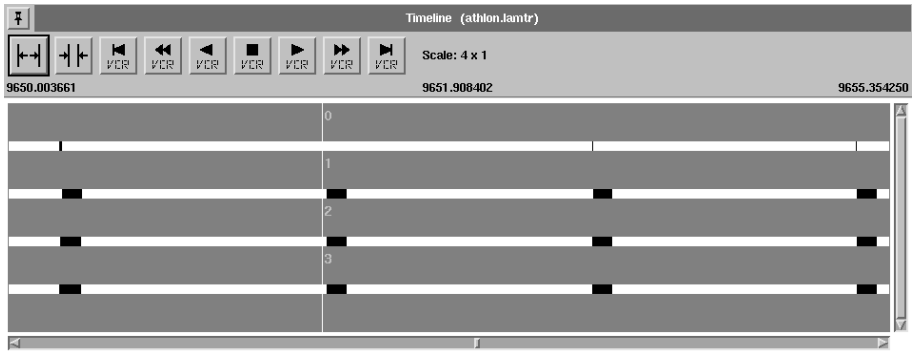


Figure 4.2: Trace of the learning process, for the experiment with 4 CPUs in the Athlon cluster. Communication and synchronization periods are shown in dark color. Computation periods are shown in light color.

- Parallel and sequential programs are not exactly the same. Different program structures can allow the compiler to perform different optimizations that have impact in execution times.

These sources of acceleration may compensate the overheads imposed by communication and synchronization.

Related to the different architectures:

- Experiments on both clusters obtain good results compared to those run on the multiprocessor; in particular, the Xeon cluster is notably fast. Therefore, it seems that for the parallel approaches we have developed, a modest architecture using commodity computers can provide very satisfactory performance levels.

  It is not our aim to extend this conclusion to general programs. The Itanium-2 processor used in the Altix machine has been specifically designed to work efficiently with floating-point intensive computations. Thus, not all workloads are able to take full advantage of this CPU. Newer versions of this processor run at higher clock frequencies (up to 1.66GHz) and incorporate larger cache memories (up to 9MB) in order to improve performance. Unfortunately, we have not had access to a machine with these processors to carry out additional experiments.

- It is also interesting to observe the differences in achieved performance when using MPI or threads. Except for the Xeon cluster, the MPI implementation gets good efficiency, showing that communication and synchronization are done efficiently. Related to the Xeon cluster, the difference between both implementations is due to the hyper-threading technology. Intel has optimized their processors (and compilers) to efficiently support the concurrent execution of several threads, and the MPI&Threads version of our program takes advantage of this.

As conclusion, a rule-of-thumb would be that, in the absence of mechanisms to accelerate thread execution, a pure MPI version of the program is capable of obtaining good levels of performance in most parallel machines.

4.3.2 Load distribution

A good load distribution is essential to achieve good levels of efficiency and scalability. As all parallel processes work together to execute a particular program, the work that each one completes must be equally distributed, avoiding great differences between them, because the slowest process determines the overall execution time. Sometimes –like in this situation–, the work cannot be evenly distributed between the processes (perfect division) due to the characteristics of the problem to be solved. However, given the large number of tasks to be distributed, this difference is almost unnoticeable.

Table 4.4 shows the execution times (actual CPU usage) for the different scenarios and implementations of our program. As expected, the master process (which is in charge of all the sequential part, the distribution of the workload and the collection of partial results) uses a slightly larger portion of CPU time. However, distribution is very satisfactory, because (1) the extra CPU-time required by the manager is not very significant, and (2) distribution of workload among workers is also fairly even. This means that our choice of phases to parallelize, and the approach used to implement them, has been successful in fulfilling our design objectives.

TABLE 4.4: LOAD DISTRIBUTION OF THE DIFFERENT EXPERIMENTS. EXECUTION TIME FOR THE MANAGER AND THE MEAN EXECUTION TIME AND DEVIATION FOR THE WORKERS.

Athlon Cluster

CPUs	*MPI version*		*MPI&Threads version*	
	t. manager	*t. worker±deviation*	*t. manager*	*t. worker±deviation*
2	87,512	86,194 ± 372	90,468	89,637 ± 120
4	44,371	42,950 ± 639	46,217	45,038 ± 360
8	21,999	20,841 ± 098	23,209	22,127 ± 222
16	11,937	10,598 ± 124	12,305	11,101 ± 154

Xeon Cluster

CPUs	*MPI version*		*MPI&Threads version*	
	t. manager	*t. worker±deviation*	*t. manager*	*t. worker±deviation*
2	47,387	46,458 ± 233	35,553	35,070 ± 096
4	24,250	22,991 ± 316	18,153	17,515 ± 206
8	12,742	11,597 ± 172	9,288	8,644 ± 121

Altix Multiprocessor

CPUs	*MPI version*		*MPI&Threads version*	
	t. manager	*t. worker±deviation*	*t. manager*	*t. worker±deviation*
8	19,792	18,392 ± 312	22,060	21,104 ± 049

4.4 Summary

In this chapter we have extended the parallel version of the EBNA$_{BIC}$ algorithm: it performs in parallel not only the learning phase, but also another, expensive one (in computational terms): the "sampling and evaluation" phase.

In order to observe the behavior of the algorithm in three different scenarios a FSS problem has been used: `Internet~Advertisement`. However, it must be pointed out that conclusions can be generalized to other problems with complex fitness functions. Our main goal has been to assess the scalability of this algorithm when solving simple (previous chapter) and complex problems (present chapter). In fact, the algorithm has been evaluated in different scenarios, obtaining excellent levels of performance (with efficiency generally close to 1).

Some interesting conclusions have been obtained. First, clusters of computers can be considered a good option in terms of cost and performance. Second, a MPI-based implementation is competitive enough in a representative range of computing platforms; therefore it does not seem necessary to put too much effort in implementing mixed-paradigm versions (MPI and threads).

Chapter 5

Parallel EDAs to create calibration models for quantitative chemical applications

This chapter is devoted to the utilization of (parallel) EDA-based approaches to solve a problem of interest in the field of quantitative chemistry. The goal is to obtain a calibration model, able to predict the initial concentrations of some species in a chemical reaction.

In the search for sound solutions, different approaches have been considered, from very simple to quite sophisticated, always trying to obtain simple (and easy to interpret) models that, at the same time, provide high levels of accuracy. The combination of parallel EDAs together with Partial Least Squares Regression gives excellent results, generating models that fulfill these properties.

5.1 Introduction

In modern laboratories the development of chemical instrumentation has allowed the existence of equipment that can acquire large amounts of data in a short period of time. For instance, whole Ultraviolet-Visible (UV-Vis) spectra can be obtained at a rate of several samples per second by diode-arrays or charge-coupled devices, and the same happens with mass spectra or nuclear magnetic resonance spectra. Typically, the number of data points in each spectrum ranges between 100 and 1000, and the number of spectra acquired in a run ranges between 100 and 200. All this data is easily stored into a personal computer, opening new possibilities for exploring the information buried inside. All kinds of data mining techniques can then be applied in order to extract knowledge from the raw data.

Many chemical reactions can be followed through the change of their UV-Vis spectrum. When the chemical and physical reaction conditions are controlled, the rate of changes in this spectrum can be made dependent exclusively on the concentration of the species taking part in the reaction. Very similar species give rise, frequently, to different reaction rates with a common reagent; this provides a way to determine the concentration of species in the original mixture. This is, usually, the essential information that is looked for. The raw data of every run makes up the experimental signal that can be used to resolve mixtures of 2-3 highly related components. The use of multivariate calibration algorithms applied to reaction rate data helps to improve the selectivity of analytical methods because of the discriminant power of the reaction kinetics. To do this, a procedure in two steps is accomplished: in the first one,

enough experimental matrices of data are obtained for different and known concentrations of the species of interest. All this information is used to establish a model that, in a second step, can be used to predict the concentration of the same species in unknown sample mixtures.

Among multivariate algorithms used for calibration, Partial Least Squares Regression (PLS) [210] and Artificial Neural Networks (ANN) [119] have frequently been used. Nowadays, PLS is, by far, the most used algorithm because it was specifically developed to provide optimum prediction ability. However, ANNs are claimed to provide better results in cases where non-linear systems are involved; the reason is that ANNs are intrinsically non-linear algorithms, whereas PLS is a linear algorithm [19]. The algorithm of our choice is trained with the experimental data (or with data elaborated from it) during the model-construction step, also known as the calibration step. Once trained, it can be used to predict values. Obviously, the goal is to obtain a model able to provide the initial concentration of the species of interest with an error as low as possible. This model can be very useful in different practical areas: for example, an independent laboratory could use the model to verify that a given drug contains the concentration of components described in the prospectus.

When dealing with this kind of problems, that is, datasets with thousands of variables, an important stage must usually be completed: a reduction of the number of variables, looking for those that have the most relevant information. As introduced in the previous chapter, several approaches can be used for this size reduction: Feature Construction and Feature Subset Selection (FSS). Feature Construction techniques look for the relation among the variables and return a set of new variables, combining the original ones. In contrast, FSS looks for the most significant variables, returning a subset of the original group.

In the search of a simple solution for the problem, two techniques for variable reduction were studied initially: Principal Component Analysis (PCA) and Filtering. As the obtained solutions were not satisfactory, a more complex solution was applied, combining filter and wrapper techniques in two consecutive steps: after a preliminary filtering, we used parallel Estimation of Distribution Algorithms (EDAs) to implement a wrapper-based selection of variables. Although we are conscious that other efficient approaches exist [38], we decided to use parallel EDAs because of two reasons: (1) sequential versions of EDAs have been proven to obtain good results in problems like this [90, 92], and (2) the speed up and scalability of the parallel versions (see Section 4.3) makes them suitable for medium-big problems (as is the case).

The rest of the chapter is organized as follows: Section 5.2 begins with a description of the two prediction techniques (ANN and PLS) used to complete the calibration. Section 5.3 introduces three example problems used throughout this chapter to evaluate our proposals. The different approaches developed for each prediction technique are shown in Sections 5.4 (ANN) and 5.5 (PLS). In Section 5.6 a initial filter+wrapper solution based on parallel EDAs is presented. This method is extended in the next section using different approaches (Section 5.7) and different algorithms (Section 5.8). Finally, Section 5.9 ends with some conclusions.

5.2 Prediction techniques

5.2.1 Artificial Neural Networks

Essentially, an ANN [119] can be defined as a pool of simple processing units (nodes) that work together sending information among them. Each unit stores the information that receives and produces an output that depends on an internal function. From the several

variants of ANNs described in the literature, we used the so-called multilayer perceptron (MLP) [167]. In this model, nodes are organized in layers: an input layer, an output layer and several intermediate (hidden) layers, where the nodes of a layer can only be connected to nodes of adjacent layers. Once the structure has been defined (number of layers and nodes per layer), it is necessary to adjust the weights of the network, so that it produces the desired output when confronted with a particular input. This process is known as training. As occurs with the structure, different proposals have been presented to complete this training. Among them, we selected a classic approach called Backpropagation [170].

5.2.2 Partial Least Squares Regression

PLS Regression [210] is a common tool in chemistry, used to analyze two data matrices $\mathbf{X}$ and $\mathbf{Y}$ by a linear multivariate model. In PLS, the structure of $\mathbf{X}$ and also of $\mathbf{Y}$ is modelled to give richer results than the traditional multiple regression approach. This is achieved indirectly by extracting latent variables $\mathbf{T}$ and $\mathbf{U}$ from matrices $\mathbf{X}$ and $\mathbf{Y}$, respectively. The extracted factors $\mathbf{T}$ (also referred to as $\mathbf{X}$-scores) are used to predict the $\mathbf{Y}$-scores $\mathbf{U}$, and then the predicted $\mathbf{Y}$-scores are used to construct predictions for the responses. One of the main characteristics of PLS, which is particularly useful in our case, is its ability to deal with datasets of many features and a reduced number of samples.

5.3 Chemical reactions used in this study

The problems used throughout this chapter correspond to one binary and two ternary mixtures of chemical species (target variables) whose original concentration we want to predict. To achieve this, another chemical species (the reagent) is added to the mixture. The reaction of the species and the reagent is followed through the change, along time, of the UV-Vis spectrum. From the experimental raw data obtained for different initial mixtures (samples) with known initial concentrations, a calibration model can be created. The features (variables) that represent a sample consist of light absorbance values at several wavelengths at successive intervals of time.

The aim of the calibration step is to generate a model that relates the experimental time-spectral data of calibration mixtures to the concentration data of the species of interest of each mixture.

It must be remarked that the features of a sample consist of b spectra, each spectrum for one time step, whereas the target consists of only one concentration for each species: the initial one, instead of b concentrations (one for each time step). In the former case, there is a quantitative analytical problem of determination (the aim of the present work); in the latter case the problem would consist of following the concentration of species along time which is a completely different problem.

The concentration of the target variables in the training datasets is normalized to values in the range [0,1]. In this chapter we have used datasets from three chemical systems identified as *small*, *medium* and *large*, depending on the number of features available in each case (see Table 5.1). The number of total features can be calculated by multiplying the number of discrete wavelengths by the number of time steps. The problem defined as *small* corresponds to mixtures of acetylsalicylic acid (aspirin) and acetaminophen (paracetamol) that are frequently found in analgesic preparations. They were made to react with bromine as a reagent [111]. The problem defined as *medium* corresponds to mixtures of three related

Table 5.1: Characteristics of the problems used in the experiments

	Small	*Medium*	*Large*
Time interval (step)	3 to 903 (30 s)	3 to 501 (6 s)	2 to 602 (10 s)
Wavelength interval	300 to 390 (2 nm)	250 to 496 (6 nm)	290 to 470 (2 nm)
Features	1426	3528	5551
Samples	76	58	181
Targets	2	3	3

aminoacids, present in the human organism: homocysteine is formed as an intermediate in the metabolism of the methionine to cysteine. In this case, mixtures of the three aminoacids were made to react with dichromate, which is a powerful oxidant [65]. The problem defined as *large* corresponds to mixtures of formaldehyde, glyoxal and glutaraldehyde. These mixtures react with 3-methyl-2-benzothiazolone hydrazone [206]. In Table 5.1 we have collected the particular characteristics of each problem. The time interval is the time (in seconds) along which measurements are acquired (the frequency of acquisition is also given); the table also provides the wavelength interval and step (units are in nanometers).

5.4 MLP approach

With this choice of calibration algorithm, we need to consider the way it is used. The problems have more than one thousand variables and feeding directly the neural network with all these values could be an initial approximation. However, results obtained this way are far from good: using so many variables as input makes difficult for the MLP to differentiate the really relevant ones. Therefore, the model we present for this technique has two modules. The first one, selection, takes as input all the dataset and reduces it considering only the most relevant variables or principal components (depending on the approach). The second one uses an (already) trained MLP which takes as input the variables selected by the first step and returns the values for the variables to be predicted.

Due to the small number of cases in the datasets, we have chosen to complete a 5-fold cross-validation to measure the accuracy of the different models proposed. As fitness value, a global error value is given, defined as the average of the square difference between the predicted value and the real value for each variable. Furthermore, as training a MLP is a non-deterministic process, we need to repeat all the process several times; we fixed this value to 10.

In the following sections we explain how PCA and a filtering technique are used to reduce the number of variables, obtaining a subset with the most significant ones.

5.4.1 PCA approach

The initial approach to the problem was to use PCA to extract the main characteristics of the dataset and, afterwards, train the MLP to build the calibration module. PCA involves a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components.

Once a dataset with the principal components is available, we can test different MLP structural configurations in order to select the best one. We do that using a brute-force approach: testing a large range of possibilities for the input and hidden layers of the MLP.

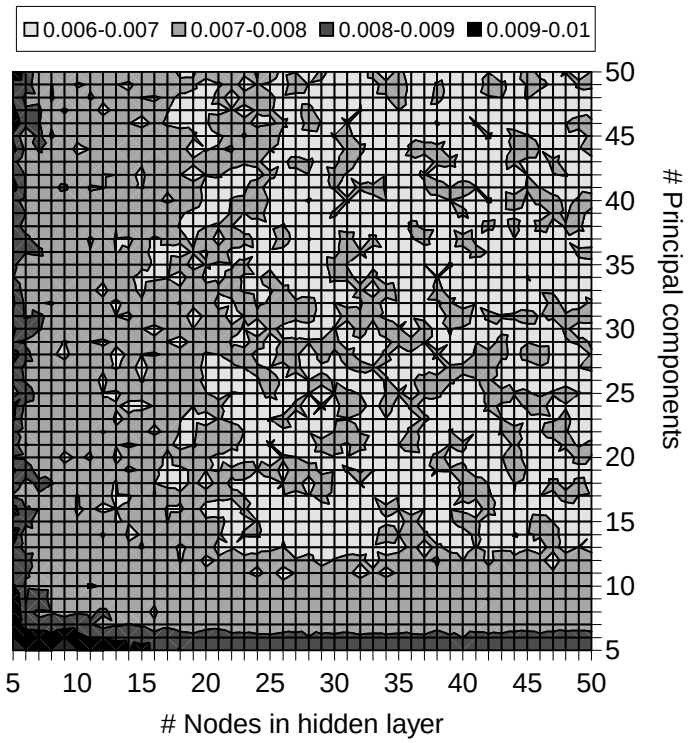


Figure 5.1: *Small* problem. Average square error for different PCA+MLP configurations.

Obviously, we need to put a limit to this trial-and-error process, due to the huge number of possible configurations for the MLP. In particular, we decided to fix the number of hidden layers to one. The number of nodes of the output layer is fixed by each problem, that is, the number of species to be predicted. So, we need to determine the configurations for the input and hidden layers.

The number of nodes of the input layer depends on how many principal components we want to incorporate in our model. A priori, we do not know how many of them are really useful. Also, we do not know the optimum configuration of the hidden layer. For this reason, we tested configurations with 5 to 50 principal components, and 5 to 50 intermediate nodes. Obtained results (for the three problems) are plotted in error maps (Figures 5.1, 5.3, 5.5) where each map point (x, y) represents the average square error for a configuration with x hidden nodes and y input nodes (please note the different scales used for each problem).

As can be seen in the maps, configurations with too few intermediate nodes yield large error values. Regarding the number of components (input nodes), we need more than 7 to achieve a good MLP configuration. Increasing too much the number of nodes in the hidden layer decreases the accuracy for the *large* problem (see Figure 5.5), although this does not happen with the other two problems (see Figures 5.1 and 5.3). Taking into account the propensity to over-fitting of neural networks, the simplest model should be chosen; in other words, the number of nodes in the hidden layer should be as low as possible. The best results for this PCA+MLP approach can be consulted in Table 5.2.

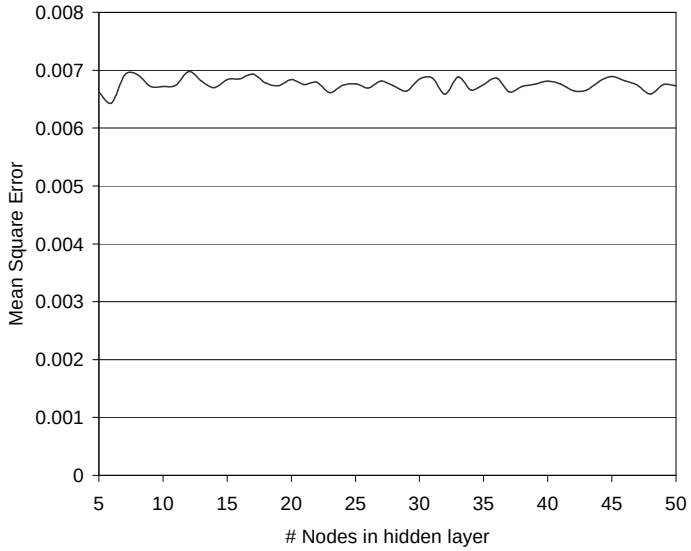


Figure 5.2: ***Small*** **problem. Average square error for different Filter+MLP configurations.**

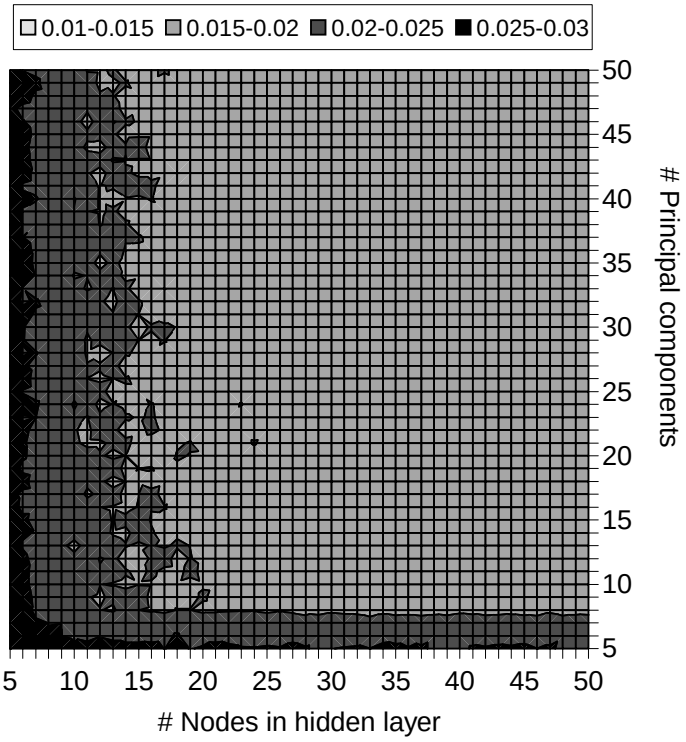


Figure 5.3: ***Medium*** **problem. Average square error for different PCA+MLP configurations.**

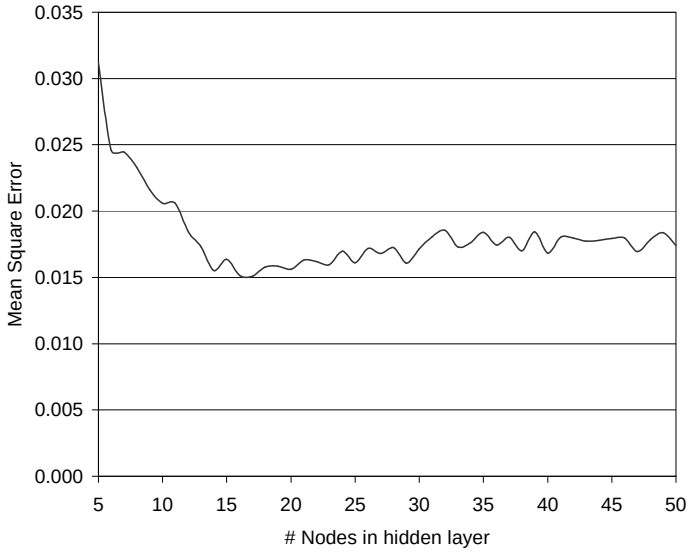


Figure 5.4: ***Medium*** **problem. Average square error for different Filter+MLP configurations.**

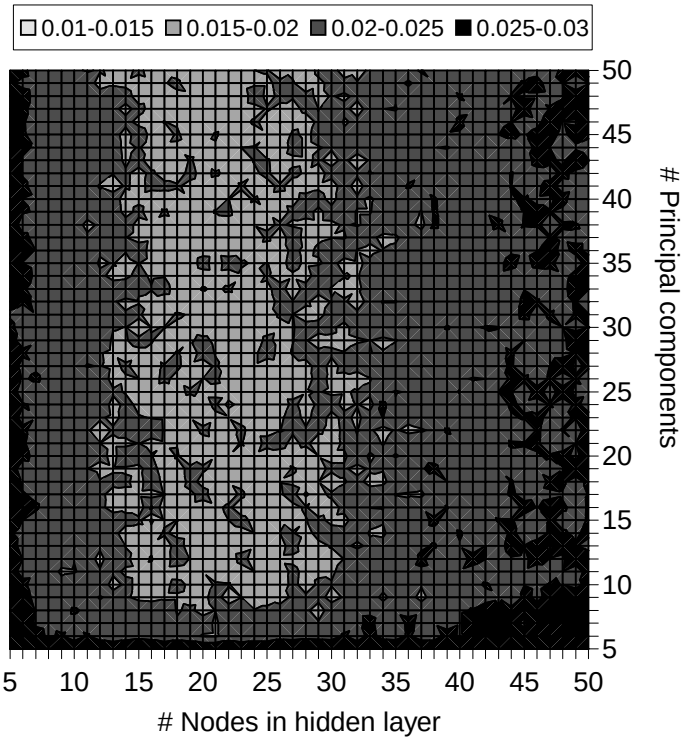


Figure 5.5: ***Large*** **problem. Average square error for different PCA+MLP configurations.**

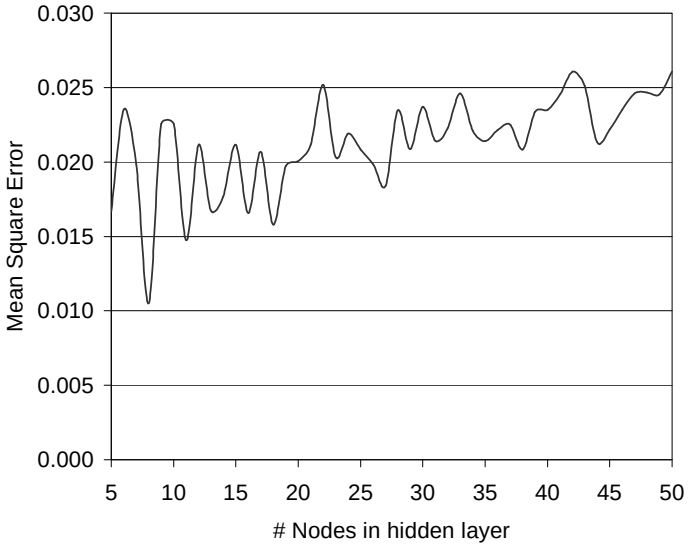


Figure 5.6: *Large* problem. Average square error for different Filter+MLP configurations.

5.4.2 Filter approach

In the literature, several proposals have been described to complete filter approaches. Usually, these techniques perform an univariate evaluation for each variable, assigning it a value. Once all the variables have been evaluated, we have a sorted list (ranking) based on the relevance of each variable.

The approximation that we have used in this chapter is the Correlation-based Feature Selection (CFS) [75]. CFS is a multivariate approach to filter, i.e., it is able to evaluate the goodness of subsets of variables, returning as a result the set of the most relevant ones. This method requires all the data to be discrete, so a previous discretization process was completed employing one of the most used algorithms, the so-called Entropy or Fayyad-Irani method [57].

The filtering process, when applied to our problems, obtained a subset of a few relevant variables, 33, 55 and 31 for the *small*, *medium* and *large* problems respectively, which fix the configurations of the input layers of the MLPs. As we did for the PCA method, we tested 46 different configurations for the intermediate layers, varying the number of nodes from 5 to 50. Obtained error values are plotted in Figures 5.2, 5.4, and 5.6. In this approach we have a behavior similar to PCA+MLP, where increasing the number of hidden-layer nodes is initially helpful but hurts when too many nodes are used. In addition, we can note that this approach uses a simpler model (less nodes in the hidden-layer) and improves for all the problems the results obtained by the previous version (pca+mlp). The best results for this approach can be consulted in Table 5.2.

5.5 PLS approach

5.5.1 PLS with the whole set of features

As done for the previous technique (MLP), we tested two initial approaches using PLS. In the first one, PLS is applied directly over the whole feature set. In the second one, a previous

TABLE 5.2: # **i**NPUT VARIABLES, # **n**EURONS IN THE HIDDEN LAYER AND MEAN SQUARE ERROR FOR THE BEST PCA+MLP AND FILTER+MLP APPROACHES

	Small			*Medium*			*Large*		
	#i	*#n*	*Error*	*#i*	*#n*	*Error*	*#i*	*#n*	*Error*
PCA+MLP	23	33	$6.942e^{-3}$	25	49	$1.521e^{-2}$	24	22	$1.473e^{-2}$
Filter+MLP	33	6	$6.434e^{-3}$	55	17	$1.510e^{-2}$	31	8	$1.057e^{-2}$

TABLE 5.3: # **i**NPUT VARIABLES, # **p**RINCIPAL COMPONENTS AND MEAN SQUARE ERROR FOR THE BEST PLS AND FILTER+PLS APPROACHES

	Small			*Medium*			*Large*		
	#i	*#p*	*Error*	*#i*	*#p*	*Error*	*#i*	*#p*	*Error*
PLS	1426	12	$4.723e^{-3}$	3528	12	$8.982e^{-3}$	5551	16	$4.058e^{-3}$
Filter+PLS	33	11	$5.231e^{-3}$	55	7	$1.890e^{-2}$	31	8	$6.644e^{-3}$

filter process is performed before applying PLS. Experiments were carried out using the PLS package [207] from the R statistical computing environment [159]. As done for MLP, a 5-fold cross-validation was completed.

Each run of PLS returns not one, but a collection of models, each one corresponding to the number of principal components used. We have fixed a maximum of 50 components, and selected the model with the best average error. In all the runs, the best models comprised less than 20 components. The results obtained with this approximation improves notably those obtained with any of the previous MLP configurations (see Table 5.3).

5.5.2 Filter and PLS

PLS has been applied over the same filtered set used for MLP. Table 5.3 summarizes the obtained average error. Note that the filtering yields worse results for all the problems. However, previous works [107, 80] demonstrate that reducing the number of features can be helpful for PLS (although experiments were carried out only in datasets with at most two hundred variables). Therefore, we decided to use EDAs to complete the selection of the variables in a different way.

5.6 Parallel EDAs and PLS

Among the different EDAs used to solve FSS problems, we decided to choose the parallel EBNA$_{BIC}$ algorithm developed in the previous chapter. On the one hand, as explained in that chapter, EBNA$_{BIC}$ has been proven to obtain good results (in terms of accuracy and convergence); on the other hand, the parallel version allows us to face larger problems.

Therefore, we will use the parallel version of EBNA$_{BIC}$ developed in Chapter 3 and extended in Chapter 4. As a reminder, two are the phases executed in a parallel way: learning (creation of the Bayesian network) and "sampling and evaluation" (creation of new individuals and evaluation of their fitness function).

Related to the problem, each variable X_i of the individual can take one of two values: 1 or 0 (1 implies that the i^{th} feature of the dataset is selected and 0 that is not selected).

In order to check the goodness of an individual, a PLS training-evaluation process (5-fold cross-validation) has to be completed using as input the set of features selected from the dataset. As explained before, the mean square error is the fitness value for each individual. The smaller the error, the better the individual.

It is well known in the literature [59] that a large number of cases (individuals) is needed to properly learn a Bayesian network. So, for problems in which individuals have several thousand variables (as is the case for these problems), it would mean that each time a new population is created, thousands of individuals should be evaluated (executing PLS with a 5-fold cross-validation). Unfortunately, this would require excessive time, even for the parallel implementation. Therefore, a previous filtering technique is applied in order to reduce the number of variables over which the smart search will be performed.

Instead of using just one of the different filtering techniques available, we decided to use 2x6 different rankings, combining two discretizations (equal frequency and equal width) with six metrics (Mutual Information, Euclidean distance, Matusita, Kullback-Leibler mode 1, Kullback-Leibler mode 2, and Bhattacharyya (described in [13])) for each target variable. From these partial rankings, a unique ranking for each target variable is created based on the mean of the positions that each variable occupies in each partial ranking. Finally, the different lists (one for each target variable) are merged selecting in an ordered way one variable from each list until there are no variables left. Once the ranking is established, the first 500 (most relevant) features are selected, setting this way the size of the individual.

Given that the aim is to obtain a good accuracy with a minimum number of features, we fixed an initial probability of 0.05 (for each variable) of being selected. The maximum number of generations was set to 300, and 1,000 was the size of the population. 999 new individuals are created in each generation and the next population is obtained selecting the best 1,000 individuals among the present population and the recently created individuals.

The results obtained using this EDA+PLS approach can be consulted in Table 5.4. Moreover, this table also extends the results obtained by previous versions, showing mean square errors and deviations. These values have been obtained as follows:

- For MLP and PLS, five executions of the best configurations (in terms of number of input and hidden nodes for MLP, and number of components for PLS) have been completed, obtaining 5 mean square errors in each execution (due to the 5-fold cross-validation).

- For EDA, ten executions have been completed obtaining ten individuals (solutions). Then, these individuals have been used to execute PLS (with the selected features), obtaining 5 mean square errors for each individual (due again to the 5-fold cross-validation).

It can be seen that, in all but one case, PLS-based approaches outperform those based on MLP. A prior filtering of the database, looking for the most relevant variables, seems to be useful for the MLP approach (better results for all the problems) but not for PLS. The use of EBNA$_{BIC}$ to carry out a selection of variables shows itself really helpful in terms of improvement of the modelling ability of PLS: compare the resulting errors and deviations with that obtained using the whole set of features. In addition, the number of features selected by EBNA$_{BIC}$ (50, 47 and 66 for the *small*, *medium* and *large* problems respectively), provides an easily interpretable model.

In the quantitative chemistry field, a model is considered valid when the calibration error is equal or less than a 10%. We have summarized in Table 5.5 the calibration errors for the

two best approaches (PLS and EDA+PLS), expressed in percentage, for the three datasets. These percentages have been calculated applying the following formula generally used in this field:

$$Percentage\ Error = 100\sqrt{\frac{\sum_{i=1}^{c}(\hat{C}_i - C_i)^2}{\sum_{i=1}^{c} C_i^2}} \tag{5.1}$$

where c is the number of cases, and $\hat{C}_i$ and C_i are the predicted and actual concentration values respectively. Note that these values ($\hat{C}_i$ and C_i) should be the real, non-normalized ones. Error percentages have been calculated using the values obtained in the best execution of each algorithm.

According to these results, PLS by itself (using all the variables) is able to obtain a valid model only for the *small* and *large* datasets. However, the EDA+PLS method provides valid models for all the problems, even if for the *medium* dataset the results are quite tight (due to the deviation). In addition, this approach presents models that can be more interpretable as the number of features is reduced from thousands to four-six tens.

It must also be pointed out that the EBNA$_{BIC}$ approach is modular in the sense that the calibration module does not have to be necessarily PLS. To show this, we also performed some initial experiments using MLP as the calibration module for the parallel EDA [121], obtaining models with better modelling abilities than those obtained by PCA+MLP or Filter+MLP (for example, for the *large* problem the mean square error was $8.367e^{-3}$). However, as the results obtained using only the PLS approximation where much better ($4.058e^{-3}$ for the *large* problem) with a significantly shorter execution time (a few seconds compared to about 18 hours), we decided to put aside this EDA+MLP approach.

The conclusion of this initial experiments is that, in terms of accuracy of calibration and validity of the models, the combination of parallel EBNA$_{BIC}$ with PLS is the most promising of all the techniques considered, for the three databases used in this study.

5.7 Improving the parallel EDAs and PLS approach

As our first attempt to use parallel EDAs combined with PLS in a calibration problem has been very satisfactory, we decided to dive deeper into the problem, trying to improve the initial proposals and reduce the prediction error even more. In the following sections, two new approaches will be evaluated, which are expected to be useful for the chemistry experts (chemists) in order to draw conclusions and choose the best ways to create accurate calibration models in the future.

5.7.1 One model versus m models

Looking at the results obtained using this first EDA+PLS approach, we realized that the results were not as good as we could expect. Therefore, we decided to re-examine the entire process looking for steps that could be improved.

According to the information received about the problem, it was supposed that the target variables were highly correlated and, therefore, we decided to work with all the target variables together and get a single prediction model for each problem. However, looking at the ranking list obtained for each target variable, we noticed some differences between the lists of preferred features for each target variable.

This means that, when mixing the m rankings (one per each target variable) into a unique list, important information could be lost. In consequence, a new approach is presented in this

TABLE 5.4: MEAN SQUARE ERRORS AND DEVIATIONS FOR EACH APPROACH AND PROBLEM (CONSIDERING ALL EXECUTIONS $_m$, AND ONLY THE BEST EXECUTION $_b$).

Small

	$Error_m$	Dev_m	$Error_b$	Dev_b
PCA+MLP	$6.942e^{-3}$	$3.287e^{-3}$	$6.832e^{-3}$	$4.141e^{-3}$
Filter+MLP	$6.434e^{-3}$	$2.931e^{-3}$	$6.226e^{-3}$	$2.929e^{-3}$
PLS	$4.723e^{-3}$	$4.133e^{-3}$	$2.582e^{-3}$	$7.733e^{-4}$
Filter+PLS	$5.231e^{-3}$	$1.411e^{-3}$	$4.692e^{-3}$	$9.504e^{-4}$
EDA+PLS	$1.716e^{-3}$	$6.241e^{-4}$	$1.458e^{-3}$	$5.516e^{-4}$

Medium

	$Error_m$	Dev_m	$Error_b$	Dev_b
PCA+MLP	$1.521e^{-2}$	$4.105e^{-3}$	$1.435e^{-2}$	$3.955e^{-3}$
Filter+MLP	$1.510e^{-2}$	$5.573e^{-3}$	$1.431e^{-2}$	$5.053e^{-3}$
PLS	$8.982e^{-3}$	$3.260e^{-3}$	$6.567e^{-3}$	$1.671e^{-3}$
Filter+PLS	$1.890e^{-2}$	$6.178e^{-3}$	$1.640e^{-2}$	$4.754e^{-3}$
EDA+PLS	$4.678e^{-3}$	$1.806e^{-3}$	$4.006e^{-3}$	$1.548e^{-3}$

Large

	$Error_m$	Dev_m	$Error_b$	Dev_b
PCA+MLP	$1.473e^{-2}$	$2.747e^{-3}$	$1.347e^{-2}$	$3.749e^{-3}$
Filter+MLP	$1.057e^{-2}$	$1.539e^{-3}$	$9.866e^{-3}$	$1.863e^{-3}$
PLS	$4.058e^{-3}$	$7.514e^{-4}$	$3.907e^{-3}$	$6.209e^{-4}$
Filter+PLS	$6.644e^{-3}$	$1.316e^{-3}$	$5.851e^{-3}$	$1.151e^{-3}$
EDA+PLS	$3.154e^{-3}$	$6.016e^{-4}$	$2.855e^{-3}$	$7.730e^{-4}$

TABLE 5.5: MEAN ERROR PERCENTAGES AND DEVIATIONS (ONLY FOR THE BEST EXECUTION) FOR PLS AND EBNA$_{BIC}$+PLS APPROACHES.

	Small	*Medium*	*Large*
	% Error	*% Error*	*% Error*
PLS	9.16±3.83	14.72±3.82	5.38±0.79
EDA+PLS	5.37±1.41	9.71±2.09	4.62±0.61

section, splitting each problem in m different and independent problems, where a separate model will be created for each of the m target variables. It is supposed that, in this way, each target variable will use its 500 most relevant features according to the previous ranking process.

The results obtained after completing the new experiments (detailed for each target variable) can be consulted in Tables 5.6 (mean square errors and deviations) and 5.7 (percentages). As expected, this approach improves the results obtained by using a unique model (particularly for the *small* problem). In addition, a statistical test was completed, particularly the Wilcoxon rank sum, with a confidence level of 0.05. The results show that there exists a statistical difference between both approaches for all the target variables and

TABLE 5.6: DETAIL OF MEAN SQUARE ERRORS AND DEVIATIONS FOR THE EBNA$_{BIC}$+PLS APPROACH WITH m MODELS USING *ranking*.

Small

EDA+PLS (2 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.533e^{-4}$	$8.515e^{-5}$	$7.333e^{-5}$	$3.210e^{-5}$
Target 2	$1.017e^{-3}$	$3.983e^{-4}$	$6.186e^{-4}$	$2.055e^{-4}$

Medium

EDA+PLS (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$3.108e^{-3}$	$1.542e^{-3}$	$1.937e^{-3}$	$8.051e^{-4}$
Target 2	$1.665e^{-3}$	$8.305e^{-4}$	$1.350e^{-3}$	$5.778e^{-4}$
Target 3	$3.919e^{-3}$	$1.888e^{-3}$	$3.118e^{-3}$	$1.540e^{-3}$

Large

EDA+PLS (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$5.405e^{-4}$	$1.499e^{-4}$	$4.065e^{-4}$	$6.998e^{-5}$
Target 2	$4.152e^{-4}$	$1.048e^{-4}$	$3.205e^{-4}$	$1.224e^{-4}$
Target 3	$4.992e^{-3}$	$1.052e^{-3}$	$4.525e^{-3}$	$9.893e^{-4}$

TABLE 5.7: DETAIL OF ERROR PERCENTAGES AND DEVIATIONS FOR THE EBNA$_{BIC}$+PLS APPROACH WITH m MODELS USING *ranking*.

Small

	% Error		
	Target 1	*Target 2*	*Mean*
EDA+PLS (2 models)	1.33±0.27	3.81±0.76	2.57

Medium

	% Error			
	Target 1	*Target 2*	*Target 3*	*Mean*
EDA+PLS (3 models)	6.76±1.40	5.76±2.68	8.56±1.36	7.03

Large

	%Error			
	Target 1	*Target 2*	*Target 3*	*Mean*
EDA+PLS (3 models)	1.92±0.32	1.71±0.13	6.43±0.54	3.36

problems. However, it can be observed that the error percentage for *target 3* in the *medium* problem is still quite high (8.56%) and, therefore, due to the deviation, other approaches could be considered in order to improve the model.

5.7.2 Other codification schemes

Now that encouraging results (in term of prediction accuracy) have been obtained creating m models for each problem, it is time for experts to discuss the results focusing also on the selected set of variables (features). As a reminder, each feature of the dataset contains a light absorbance value for a particular wavelength in a particular moment (time) of the reaction.

These new approaches, instead of completing an initial ranking process (ordering the features and selecting the best 500), consider all the features, using two different codifications for the individual:

Time intervals: The individual is codified in such a way that each variable X_i represents the i^{th} time interval. As in the previous representation, each X_i can take two values: 0 or 1. If the i^{th} variable takes the value 1, the light absorbance values for all the wavelengths in this i^{th} time interval will be used to construct the model. In this case, individual sizes are 31, 84, and 61 for the *small*, *medium*, and *large* problems respectively.

Wavelengths: In this case, the variables of the individual represent the different wavelengths. If the variable X_i is selected, all the light absorbance values of the i^{th} wavelength (through the reaction process) will be taken into account. With this codification, individual sizes are 46, 42, and 91 for the *small*, *medium*, and *large* problems respectively.

For these experiments, we used the approximation that creates a separate model for each target variable (as it has been seen to be the best option). Related to EDA, the following parameters have been selected: a initial probability of 0.05 for each variable of being selected, a maximum of 300 generations, and a population size of 200.

In Tables 5.8 and 5.9, the results of these two new approaches are presented. It must be noted that the *time intervals* approach is better than the *wavelengths* approach, showing smaller prediction errors. Compared to the *ranking* solution, these approaches show higher prediction errors with some interesting exceptions: for example, the error percentage for *target 3* in the *medium* problem is lower in these approaches (particularly for *time intervals*, with a 6.07%). Therefore, it seems that valuable conclusions can also be obtained from these new proposals.

Finally, statistical tests (Wilcoxon, $\alpha = 0.05$) have been completed, comparing the EDA+PLS approach (from now on *ranking* approach) with each of these two proposals (*time intervals* and *wavelengths*) showing significant differences.

Looking at the results (error percentages), and considering each target independently, both approaches are able to provide good models for the *small* and *large* problems, but results for the *medium* problem exceed the fixed limit (10%) for *target 1*.

Related to these new proposals, it must be noticed that when a variable takes the value 1, this means that a time interval or a particular wavelength are considered important. From our point of view, this information seems to be more meaningful than a set of isolated light absorbance values for particular wavelengths and time intervals. However, as said before, the experts should study these models in order to obtain useful conclusions.

TABLE 5.8: DETAIL OF MEAN SQUARE ERRORS AND DEVIATIONS FOR THE EBNA$_{BIC}$+PLS APPROACH WITH m MODELS USING *wavelengths* AND *time intervals*.

Small

Time intervals (2 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$8.613e^{-5}$	$4.501e^{-5}$	$6.092e^{-5}$	$1.595e^{-5}$
Target 2	$3.772e^{-3}$	$1.712e^{-3}$	$3.227e^{-3}$	$1.894e^{-3}$
Wavelengths (2 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$4.209e^{-4}$	$4.039e^{-4}$	$2.284e^{-4}$	$7.774e^{-5}$
Target 2	$4.204e^{-3}$	$1.714e^{-3}$	$3.541e^{-3}$	$2.227e^{-3}$

Medium

Time intervals (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$5.819e^{-3}$	$2.646e^{-3}$	$4.453e^{-3}$	$2.001e^{-3}$
Target 2	$3.370e^{-3}$	$1.665e^{-3}$	$2.825e^{-3}$	$1.550e^{-3}$
Target 3	$2.581e^{-3}$	$1.464e^{-3}$	$1.569e^{-3}$	$5.510e^{-4}$
Wavelengths (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$8.624e^{-3}$	$3.999e^{-3}$	$7.274e^{-3}$	$3.310e^{-3}$
Target 2	$4.932e^{-3}$	$2.006e^{-3}$	$3.691e^{-3}$	$1.368e^{-3}$
Target 3	$3.785e^{-3}$	$1.999e^{-3}$	$2.632e^{-3}$	$1.223e^{-3}$

Large

Time intervals (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.140e^{-3}$	$2.997e^{-4}$	$9.386e^{-4}$	$2.285e^{-4}$
Target 2	$5.085e^{-4}$	$1.243e^{-4}$	$4.396e^{-4}$	$1.321e^{-4}$
Target 3	$7.193e^{-3}$	$1.563e^{-3}$	$6.491e^{-3}$	$1.609e^{-3}$
Wavelengths (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.504e^{-3}$	$3.140e^{-4}$	$1.399e^{-3}$	$2.157e^{-4}$
Target 2	$7.454e^{-4}$	$1.760e^{-4}$	$6.869e^{-4}$	$1.950e^{-4}$
Target 3	$8.309e^{-3}$	$1.587e^{-3}$	$7.885e^{-3}$	$1.425e^{-3}$

5.8 EBNA$_{BIC}$ and UMDA

In order to properly finish this study about the different approaches to the problem, we decided to repeat the most promising experiments using another algorithm of the family of EDAs: UMDA.

UMDA is a very simple algorithm that considers all the variables to be independent and, therefore, the factorization is given as a product of independent univariate marginal distributions (see Chapter 2 for additional information).

Throughout this problem, we have focused on the EBNA$_{BIC}$ algorithm because it was applied to the FSS problem in [92], providing good results and a faster convergence rate than other algorithms. However, in a later study by [36], the author did not find any improvement when using complex EDAs (based on Bayesian networks) instead of simpler univariate approaches (like UMDA).

Therefore, trying to clarify these different conclusions we repeated some experiments using *ranking*, *wavelengths* and *time interval* approaches, creating a model for each target

TABLE 5.9: DETAIL OF ERROR PERCENTAGES AND DEVIATIONS FOR THE EBNA$_{BIC}$+PLS APPROACH WITH m MODELS USING *wavelengths* AND *time intervals*.

Small

	% Error		
	Target 1	*Target 2*	*Mean*
Time intervals (2 models)	1.22±0.23	8.55±3.32	4.88
Wavelengths (2 models)	2.44±0.41	8.86±2.51	5.65

Medium

	% Error			
	Target 1	*Target 2*	*Target 3*	*Mean*
Time intervals (3 models)	10.35±3.96	8.35±3.97	6.07±2.30	8.26
Wavelengths (3 models)	13.60±2.37	9.13±2.56	7.98±2.28	10.24

Large

	% Error			
	Target 1	*Target 2*	*Target 3*	*Mean*
Time intervals (3 models)	2.87±0.46	2.00±0.35	7.74±1.31	4.20
Wavelengths (3 models)	3.54±0.34	2.52±0.25	8.43±1.40	4.80

variable. We implemented a parallel version of the UMDA algorithm, in which the "sampling and evaluation" phase is executed in a parallel way following the same scheme used for EBNA$_{BIC}$.

The results obtained from the experiments have been analyzed from two points of view: quality (accuracy) of the obtained models, and convergence and execution time of the algorithms.

5.8.1 Quality of the models

Related to the quality of the models, we can observe in Tables 5.10, and 5.11 that both algorithms, EBNA$_{BIC}$ and UMDA get similar results. Attending to the error percentages, EBNA$_{BIC}$ obtains generally better models (lower errors) for all the approaches. However, statistical tests demonstrate that there is no significant difference between both approaches[1]. Therefore, we should focus on the characteristics and selected features of the obtained models in order to make a decision.

5.8.2 Convergence and execution times

In Table 5.12 the number of generations needed by each approach as well as the execution times (using 22 processors) are presented.

Related to the convergence, it can be seen that the EBNA$_{BIC}$ algorithm needs generally less generations than the UMDA in almost all the approaches and problems (this fact confirms conclusions obtained in previous works). However, as EBNA$_{BIC}$ has to learn (create) a Bayesian network at each generation, depending on the size of the individual, a smaller

[1]Except for *target 2* in the *small* problem

TABLE 5.10: DETAIL OF MEAN SQUARE ERRORS AND DEVIATIONS FOR THE UMDA+PLS APPROACH WITH m MODELS USING *ranking*, *wavelengths*, AND *time intervals*.

Small

Ranking (2 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.134e^{-4}$	$5.637e^{-5}$	$4.871e^{-5}$	$2.008e^{-5}$
Target 2	$8.659e^{-4}$	$3.584e^{-4}$	$7.271e^{-4}$	$3.136e^{-4}$
Time intervals (2 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$3.501e^{-3}$	$1.167e^{-2}$	$6.196e^{-5}$	$3.141e^{-5}$
Target 2	$3.927e^{-3}$	$1.676e^{-3}$	$3.561e^{-3}$	$5.131e^{-4}$
Wavelengths (2 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$4.902e^{-4}$	$1.221e^{-3}$	$2.232e^{-4}$	$6.217e^{-5}$
Target 2	$4.277e^{-3}$	$2.087e^{-3}$	$3.726e^{-3}$	$1.637e^{-3}$

Medium

Ranking (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$3.074e^{-3}$	$1.731e^{-3}$	$2.452e^{-3}$	$1.484e^{-3}$
Target 2	$1.411e^{-3}$	$6.233e^{-4}$	$1.188e^{-3}$	$4.315e^{-4}$
Target 3	$3.727e^{-3}$	$1.750e^{-3}$	$3.109e^{-3}$	$1.223e^{-3}$
Time intervals (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$6.829e^{-3}$	$3.086e^{-3}$	$5.298e^{-3}$	$2.452e^{-3}$
Target 2	$3.799e^{-3}$	$1.988e^{-3}$	$2.644e^{-3}$	$1.321e^{-3}$
Target 3	$2.295e^{-3}$	$1.343e^{-3}$	$1.584e^{-3}$	$5.863e^{-4}$
Wavelengths (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.002e^{-2}$	$5.547e^{-3}$	$7.669e^{-3}$	$3.573e^{-3}$
Target 2	$5.069e^{-3}$	$1.731e^{-3}$	$3.882e^{-3}$	$9.511e^{-4}$
Target 3	$3.743e^{-3}$	$1.874e^{-3}$	$2.786e^{-3}$	$1.010e^{-3}$

Large

Ranking (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$5.229e^{-4}$	$1.172e^{-4}$	$4.064e^{-4}$	$1.296e^{-4}$
Target 2	$4.067e^{-4}$	$8.220e^{-5}$	$3.319e^{-4}$	$8.019e^{-5}$
Target 3	$4.896e^{-3}$	$9.252e^{-4}$	$4.586e^{-3}$	$9.890e^{-4}$
Time intervals (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.060e^{-3}$	$2.660e^{-4}$	$9.683e^{-4}$	$6.565e^{-5}$
Target 2	$5.180e^{-4}$	$1.193e^{-4}$	$4.694e^{-4}$	$1.132e^{-4}$
Target 3	$7.279e^{-3}$	$1.631e^{-3}$	$5.300e^{-3}$	$1.582e^{-3}$
Wavelengths (3 models)	$Error_m$	Dev_m	$Error_b$	Dev_b
Target 1	$1.556e^{-3}$	$3.904e^{-4}$	$1.446e^{-3}$	$5.080e^{-4}$
Target 2	$7.472e^{-4}$	$1.932e^{-4}$	$6.812e^{-4}$	$1.951e^{-4}$
Target 3	$8.141e^{-3}$	$1.328e^{-3}$	$7.651e^{-3}$	$1.170e^{-3}$

TABLE 5.11: DETAIL OF ERROR PERCENTAGES AND DEVIATIONS FOR THE UMDA+PLS APPROACH WITH m MODELS USING *ranking*, *wavelengths* AND *time intervals*.

Small

	% Error		
	Target 1	*Target 2*	*Mean*
Ranking (2 models)	1.08±0.14	4.16±1.63	2.62
Time intervals (2 models)	1.21±0.32	9.22±1.27	5.21
Wavelengths (2 models)	2.31±0.36	9.20±1.86	5.75

Medium

	% Error			
	Target 1	*Target 2*	*Target 3*	*Mean*
Ranking (3 models)	7.69±1.68	5.20±0.78	8.18±2.67	7.02
Time intervals (3 models)	11.51±3.23	7.65±2.54	6.30±1.52	8.48
Wavelengths (3 models)	13.45±5.09	9.63±4.10	8.01±1.47	10.36

Large

	% Error			
	Target 1	*Target 2*	*Target 3*	*Mean*
Ranking (3 models)	1.90±0.21	1.73±0.17	6.50±1.23	3.38
Time intervals (3 models)	2.90±0.47	2.07±0.31	6.91±0.59	3.96
Wavelengths (3 models)	3.57±0.40	2.49±0.41	8.33±0.43	4.80

number of generations does not imply a smaller execution time (see for example the *ranking* approach). Therefore, it seems difficult to obtain a rule-of-thumb to decide between both proposals, as the number of generations and the execution time is highly related to the fitness function and individual size used in the problems.

5.9 Summary

In this chapter we have used EDAs in a quantitative chemistry problem: the prediction of unknown concentrations of species in a chemical reaction, using a model generated from well-controlled experiments. The challenging part in this problem is the calibration step, that is, the creation and evaluation of the model. This is done by a combination of variable reduction and multivariate calibration techniques.

As initial approaches, PCA and filtering were used together with MLP, as well as PLS with and without filtering, with the aim of reducing the calibration error of the obtained models.

In a second phase, a filter technique was combined with a wrapper approach based on parallel EDAs. Due to the high computational cost of this technique, a fast, parallel program was used for a particular EDA (EBNA$_{BIC}$). Encouraged by the good results obtained using EDA+PLS, additional experiments were completed splitting the model into m models (one for each target variable) and using different codification approaches (considering wavelengths and time intervals). In addition, other algorithm of the family of EDAs (UMDA) was parallelized and applied to the problem, providing similar results with shorter execution times. It

TABLE 5.12: NUMBER OF GENERATIONS AND EXECUTION TIMES FOR THE EBNA$_{BIC}$+PLS AND UMDA+PLS APPROACHES WITH m MODELS USING *ranking*, *wavelengths*, AND *time intervals*.

small

	Target 1		*Target 2*	
Approach	*Gen.*	*Exec.*	*Gen.*	*Exec.*
EBNA$_{BIC}$ (ranking)	300	2h 17' 27"	286	2h 07' 01"
UMDA (ranking)	300	1h 12' 48"	295	1h 08' 18"
EBNA$_{BIC}$ (time)	31	3' 06"	35	3' 50"
UMDA (time)	38	3' 27"	49	5' 32"
EBNA$_{BIC}$ (wave.)	36	4' 34"	37	4' 09"
UMDA (wave.)	42	5' 09"	39	3' 52"

medium

	Target 1		*Target 2*		*Target 3*	
Approach	*Gen.*	*Exec.*	*Gen.*	*Exec.*	*Gen.*	*Exec.*
EBNA$_{BIC}$ (ranking)	280	2h 03' 22"	289	2h 12' 18"	300	2h 14' 49"
UMDA (ranking)	290	1h 07' 12"	292	1h 08' 51"	300	1h 10' 15"
EBNA$_{BIC}$ (time)	38	6' 07"	38	7' 28"	44	6' 55"
UMDA (time)	63	9' 47"	50	9' 01"	65	9' 38"
EBNA$_{BIC}$ (wave.)	36	6' 40"	49	7' 53"	54	12' 40"
UMDA (wave.)	41	6' 35"	53	7' 44"	37	6' 36"

large

	Target 1		*Target 2*		*Target 3*	
Approach	*Gen.*	*Exec.*	*Gen.*	*Exec.*	*Gen.*	*Exec.*
EBNA$_{BIC}$ (ranking)	278	2h 29' 37"	300	3h 12' 06"	260	2h 36' 27"
UMDA (ranking)	297	1h 34' 15"	299	1h 53' 34"	213	1h 18' 54"
EBNA$_{BIC}$ (time)	51	21' 54"	47	19' 18"	69	33' 43"
UMDA (time)	77	37' 05"	62	27' 21"	70	33' 09"
EBNA$_{BIC}$ (wave.)	33	15' 56"	45	27' 38"	43	25' 59"
UMDA (wave.)	36	17' 53"	53	32' 19"	36	21' 09"

must be noted that the use of parallel algorithms has been really helpful, allowing the test of many alternatives in a reasonable amount of time.

Related to the calibration problem, promising results have been obtained using parallel EDAs, being this method the one that generates models with the smallest errors for all the datasets included in this study. In addition, it is also interesting the fact that parallel EDAs can be used not only in combination with PLS or MLP: any other method could be used, due to its modular construction; it is only required to adapt the evaluation function of the individuals to the particular calibration technique.

From the point of view of the different approaches, it is clear that the best results are obtained creating one model for each target variable. However, a deeper analysis of the

obtained results, performed by the experts, is necessary in order to choose the best solution in terms of accuracy and interpretability of the calibration models.

Chapter 6

Conclusions

In this chapter we summarize the main contributions of this work, suggesting also some plans to extend and improve the ideas presented throughout this dissertation.

6.1 Conclusions

Our work has been focused on the design of parallel implementations for existing sequential EDAs, and particularly for those based on probabilistic graphical models. The parallel approaches have been designed mixing two programming paradigms: MPI and threads. Different experiments have been carried out showing that the efficiency and the scalability of the approaches are, in general, satisfactory. In addition, three different parallel computer architectures have been tested, looking for the most promising scenario in terms of performance and price. Finally, thanks to the availability of parallel computers and programs, different experiments have been completed trying to solve a real problem in the field of quantitative chemistry. In fact, different valid solutions (models) have been obtained, and presented to the experts (chemists) so that they can choose the best one taking into account other aspects (besides accuracy).

The main contributions of this dissertation can be summarized as follows:

- A review of EDAs has been presented. Even if our work has been done using a few particular EDAs, we have considered interesting to explain the main characteristics of this family of algorithms, and also to do an exhaustive search throughout the literature looking for the most meaningful approaches and results, trying to offer to the reader up-to-date information about these algorithms.

- We have paid special attention to four different EDAs that use probabilistic graphical models. Parallel approaches for these algorithms have been presented in order to make them suitable to be used in complex problems (reducing the execution time). Particularly:

  - We have completed different parallel approaches for two algorithms belonging to the discrete domain: EBNA$_{BIC}$ and EBNA$_{PC}$, and for other two belonging to the continuous domain: EGNA$_{BIC}$ and EGNA$_{EE}$.
  - Two phases have been parallelized: the construction of the probability model (learning phase) and the creation and evaluation of new individuals based on the

learnt model ("sampling and evaluation" phase). It has been seen that for problems with complex fitness functions, the parallelization of the learning phase is not enough. In addition, for particular algorithms like EGNA$_{EE}$ the parallelization of the "sampling and evaluation" phase is necessary even if simple fitness functions are used.

- A double-level parallel scheme has been used, mixing MPI and threads. The idea is to use MPI to distribute tasks along nodes in a cluster (or multicomputer) and to use threads inside the same (multiprocessor) machine. Experiments showed that the efficiency and scalability of the algorithms is satisfactory. For example, using a cluster of 5 computers with two CPUs per computer, efficiency values are: 1.05, 0.67, 0.86, and 0.56 for the second versions of the EBNA$_{BIC}$, EBNA$_{PC}$, EGNA$_{BIC}$ and EGNA$_{EE}$ algorithms respectively.
- The preliminary experiments using "toy" functions conducted in Chapter 3 have been extended in Chapter 4 using two cluster of computers (AMD Athlon and Intel Xeon processors) and a multiprocessor (SGI Altix). Executions were completed using a parallel implementation of the EBNA$_{BIC}$ algorithm applied to a real, complex problem: FSS. Efficiency and scalability results are similar to those obtained in the previous experiments, and it can be concluded that, in general, a pure MPI implementation is a good choice (from the point of view of speed ups and homogeneity). However, the utilization of threads is advantageous when using the newest Intel processors, which include the Hyper-Threading technology. For example, using 4 computers with two processors per computer, the pure MPI version gets an efficiency of 0.76 while the mixed MPI and threads version obtains a notably higher efficiency: 1.05.
- The conclusions obtained from the experiments show that the use of parallel techniques is in general really helpful, making the algorithms affordable for a wider range of problems and providing also faster executions.

- Finally, in Chapter 5 two parallel implementations (particularly EBNA$_{BIC}$ and UMDA) have been applied to a real chemical problem. The aim was to obtain a calibration model able to predict the initial concentration values of some species taking part in a chemical reaction. Our goal has been twofold: firstly, obtain a valid model (minimizing the prediction error) and secondly, reduce the number of features (variables) needed to construct the model. Thanks to the reduction of execution times provided by the parallelism, the problem has been studied from different points of view, obtaining valid models (prediction error lower than 10%) and reducing the number of features from thousands to around sixty. Now, a second step must be completed by the experts (chemists), studying the obtained models and their characteristics.

6.2 Future work

In terms of future work, we plan to extend the research presented in this dissertation in different ways, some of them related to the parallel implementation of EDAs, and others related to the chemical problem.

From the point of view of parallelism, our future lines consider:

- Improvements of the presented parallel approaches, taking advantage of the knowledge acquired during this work. For example, for the EBNA$_{PC}$ algorithm, a workload

distribution mechanism using independent sets of pairs could be applied, avoiding sending/receiving operations for each pair. Additionally, for the second version of the EGNA$_{EE}$ algorithm, it would be interesting to look for methods to accelerate the sequential part, for example, using parallel libraries to solve some large matrix-related computations.

- Since this work started, many new EDAs have been proposed. Therefore, the ideas and conclusions obtained in this work could be used to parallelize other algorithms. In addition, important research has been done in some particular areas, like Bayesian network learning. These new approaches could be integrated into algorithms like EBNA$_{BIC}$ (replacing for example the B algorithm by other newer techniques).

- Our approach to parallelism has been very conservative, trying always to maintain the behavior of the sequential algorithms. We plan to explore other parallel schemes. For example, the island-based models presented in Chapter 2 use simple probabilistic models; they could be extended, using probabilistic graphical models like Bayesian or Gaussian networks. In addition, a double-level parallel scheme could be implemented, where the first level represents the different islands, and the second level would be the use of parallel implementations in each island. In this way, we could obtain double advantage: improvement of the accuracy and reduction of the execution time.

- Extend the experiments completed in Chapter 4, where EBNA$_{BIC}$ was tested on two clusters of computers and a multiprocessor. A new Intel Itanium-2 processors has been commercialized since we carried out our experiments. This new processor fixes some faults present in its predecessor (for example, and insufficient cache size), and it would be interesting to test the efficiency and performance using parallel implementations for algorithm in both discrete and continuous domains. Obtained conclusions could be interesting for future acquisitions.

On the other hand, regarding the chemical problem:

- We plan to extend the *wavelength* and *time interval* approaches. These approaches obtain a model that considers a set of wavelengths or a set of time intervals. However, the first approach, *ranking*, considers a set of particular features (light absorbance values). Therefore, a second execution could be completed, in which, the individual represents all the values for the selected wavelengths (through all the intervals) or all the values for the selected time intervals (for all the wavelengths). This approach would be comparable to the *ranking* approach, as the final result is a set of light absorbance values for particular wavelengths at particular time moments. In summary, this is another way to solve the problem using all the possible features in a double-step scheme.

- We also intend to select a larger number of variables in the ranking process. Instead of using just the best 500 variables, the availability of fast and scalable computing platforms would allow us to duplicate this number and observe the quality of the results obtained.

- Taking into account the good results obtained by PLS, we decided to put aside the MLP approach. However, in order to complete a fair comparison between the two techniques, we should search for up-to-date information about more efficient neural networks and do some additional research using them.

- Finally, in addition to the accuracy (error prediction) obtained by each model, it is also interesting to consider the features selected from the initial set. Therefore, a deeper study must be completed, together with the chemists, in order to understand why some characteristics are selected in a particular reaction, and their relation with the species.

Bibliography

[1] C. W. Ahn, D. E. Goldberg, and R.S. Ramakrishna. Multiple-deme parallel Estimation of Distribution Algorithms: Basic framework and application. In *Lecture Notes in Computer Science 3019: Parallel Processing and Applied Mathematics, PPAM 2003*, pages 544–551. Springer, 2004.

[2] C. W. Ahn and R. S. Ramakrishna. Elitism-based compact Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):367–385, 2003.

[3] C. W. Ahn, R. S. Ramakrishna, and D. E. Goldberg. Real-coded Bayesian optimization algorithm. In J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, volume 192 of *Studies in Fuzziness and Soft Computing*, chapter 3, pages 51–73. Springer, 2005.

[4] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons Inc, 2005.

[5] E. Alba and M. Tomassini. Paralelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

[6] E. Alba-Cabrera, R. Santana, A. Ochoa-Rodriguez, and M. Lazo-Cortés. Finding typical testors by using an evolutionary strategy. In F. Muge, M. Piedade, and R. Caldas Pinto, editors, *Proceedings of the 5th Ibero American Symposium on Pattern Recognition*, pages 267–278, Lisbon, Portugal, 2000.

[7] W. Ali and A. P. Topchy. Memetic optimization of video chain designs. In Deb et al. [52], pages 869–882.

[8] Anonymous. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):159–416, 1994.

[9] S. Baluja. Population-Based Incremental Learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon Report, CMU-CS-94-163, 1994.

[10] S. Baluja and R. Caruana. Removing the genetics from the standard Genetic Algorithms. Technical report, Carnegie Mellon Report, CMU-CS-95-141, 1995.

[11] S. Baluja and S. Davies. Fast probabilistic modeling for combinatorial optimization. In *Proceedings of the 15th National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 1998, IAAI 1998, Wisconsin, USA, July 26-30, 1998*, pages 469–476. AAAI Press–The MIT Press, 1998.

[12] W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999, Orlando, Florida, USA, July 13-17, 1999*. Morgan Kaufmann, 1999.

[13] M. Ben-Bassat. Use of distance measures, information measures and error bounds in feature evaluation. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics*, volume 2, pages 773–791. North-Holland Publishing Company, 1982.

[14] E. Bengoetxea, P. Larrañaga, I. Bloch, and A. Perchant. Solving graph matching with EDAs using a permutation–based representation. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 243–265. Kluwer Academic Publishers, 2002.

[15] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres. Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. In *Workshop Notes of CaNew2000: Workshop on Bayesian and Causal Networks: From Inference to Data Mining*, 2000. Fourteenth European Conference on Artificial Intelligence, ECAI2000. Berlin.

[16] E. Bengoetxea, T. Mikelez, P. Larrañaga, and J. A. Lozano. Results in Function Optimization with EDAs in continuous domain. In Larrañaga and Lozano [103], pages 181–194.

[17] H. G. Beyer, U. M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llorà, S. Mancoridis, M. Pelikan, G. R. Raidl, T. Soule, A. M. Tyrrell, J. P. Watson, and E. Zitzler, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005, Washington DC, USA, June 25-29, 2005*. ACM, 2005.

[18] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[19] M. Blanco, J. Coello, H. Iturriaga, S. Maspoch, and M. Porcel. Use of circular dichroism and artificial neural networks for the kinetic-spectrophotometric resolution of enantiomers. *Analytica Chimica Acta*, 431:115–123, 2001.

[20] R. Blanco, I. Inza, and P. Larrañaga. Learning Bayesian networks in the space of structures by Estimation of Distribution Algorithms. *International Journal of Intelligent Systems*, 18(2):205–220, 2003.

[21] R. Blanco, P. Larrañaga, I. Inza, and B. Sierra. Selection of highly accurate genes for cancer classification by Estimation of Distribution Algorithms. In S. Quaglini, P. Barahona, and S. Andreassen, editors, *AIME*, volume 2101 of *Lecture Notes in Computer Science*, pages 29–34. Springer, 2001.

[22] P. A. N. Bosman. *Design and Application of Iterated Density-Estimation Evolutionary Algorithms*. PhD thesis, Utrech University, 2003.

[23] P. A. N. Bosman and D. Thierens. An algorithmic framework for density estimation based evolutionary algorithms. Technical Report UU-CS-1999-46, Utrech University, 1999.

[24] P. A. N. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In Banzhaf et al. [12], pages 60–67.

[25] P. A. N. Bosman and D. Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pages 197–200, 2000.

[26] P. A. N. Bosman and D. Thierens. Expanding from discrete to continuous Estimation of Distribution Algorithms: The IDEA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI. Lecture Notes in Computer Science 1917*, pages 767–776, 2000.

[27] P. A. N. Bosman and D. Thierens. IDEAs bases on the normal kernels probability density function. Technical Report UU-CS-2000-11, Utrech University, 2000.

[28] P. A. N. Bosman and D. Thierens. Mixed ideas. Technical Report UU-CS-2000-45, Utrech University, 2000.

[29] P. A. N. Bosman and D. Thierens. Negative log-likelihood and statistical hypothesis testing as the basis of model selection in IDEAs. In *Genetic and Evolutionary Computation Conference GECCO-00. Late Breaking Papers*, pages 51–58, 2000.

[30] P. A. N. Bosman and D. Thierens. Permutation optimization by iterated estimation of random keys marginal product factorizations. In Merelo-Guervós et al. [122], pages 331–340.

[31] W. Bossert. Mathematical optimization: Are there abstract limits on natural selection? In P. S. Moorehead and M. M. Kaplan, editors, *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution*, pages 35–46. The Wistar Institute Press, Philadelphia, PA, 1967.

[32] D. F. Brown, A. B. Garmendia-Doval, and John A. W. McCall. Markov random field modelling of royal road Genetic Algorithms. In P. Collet, C. Fonlupt, J. K. Hao, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution*, volume 2310 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2001.

[33] W. Buntine. Theory refinement in Bayesian networks. In Bruce D'Ambrosio and Philippe Smets, editors, *UAI*, pages 52–60. Morgan Kaufmann, 1991.

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

[35] E. Cantú-Paz. *Efficient and accurate parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[36] E. Cantú-Paz. Feature subset selection by Estimation of Distribution Algorithms. In Langdon et al. [99], pages 303–310.

[37] E. Cantú-Paz. Pruning Neural Networks with Distribution Estimation Algorithms. In Cantú-Paz et al. [39], pages 790–800.

[38] E. Cantú-Paz. Feature Subset Selection, class separability, and Genetic Algorithms. In Deb et al. [52], pages 959–970.

[39] E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U. M. O'Reilly, H. G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003, Part I, Chicago, IL, USA, July 12-16, 2003*, volume 2723 of *Lecture Notes in Computer Science*. Springer, 2003.

[40] E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U. M O'Reilly, H. G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003, Part II, Chicago, IL, USA, July 12-16, 2003*, volume 2724 of *Lecture Notes in Computer Science*. Springer, 2003.

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

[42] D. Cho and B. Zhang. Evolutionary continuous optimization by distribution estimation with variational Bayesian independent component analyzers mixture model. In Yao et al. [218], pages 212–221.

[43] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[44] G. F. Cooper and E. A. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[45] D. Corne, Z. Michalewicz, B. McKay, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, G. Raidl, K. C. Tan, and A. Zalzala, editors. *Proceedings of the 2005 Congress on Evolutionary Computation, CEC-2005, Edinburgh, U.K., September 2-5, 2005*. IEEE Press, 2005.

[46] C. Cotta, E. Alba, R. Sagarna, and P. Larrañaga. Adjusting weights in artificial neural networks using evolutionary algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

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

[48] L. M. De Campos, J. A. Gámez, P. Larrañaga, S. Moral, and T. Romero. Partial abductive inference in Bayesian networks: an empirical comparison between GAs and EDAs. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 323–341. Kluwer Academic Publishers, 2002.

[49] L. De la Ossa, J. A. Gámez, and J. M. Puerta. Migration of probability models instead of individuals: An alternative when applying the island model to EDAs. In Yao et al. [218], pages 242–252.

[50] L. De la Ossa, J. A. Gamez, and J. M. Puerta. Improving model combination through local search in parallel univariate EDAs. In Corne et al. [45], pages 1426–1433.

[51] L. De la Ossa, J. A. Gámez, and J. M. Puerta. Initial approaches to the application of islands-based parallel EDAs in continuous domains. In Skie and Yang [194], pages 580–587.

[52] K. Deb, R. Poli, W. Banzhaf, H. G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2004, Part II, Seattle, WA, USA, June 26-30, 2004*, volume 3103 of *Lecture Notes in Computer Science*. Springer, 2004.

[53] K. Deb, R. Poli, W. Banzhaf, H. G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. Luca Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2004, Part I, Seattle, WA, USA, June 26-30, 2004*, volume 3102 of *Lecture Notes in Computer Science*. Springer, 2004.

[54] K. Deb, A. Prapat, S. Agarwal, and T. Meyarivan. A fast and multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[55] S. Droste. Not all linear functions are equally difficult for the compact Genetic Algorithm. In Beyer et al. [17], pages 679–686.

[56] R. Etxeberria and P. Larrañaga. Global optimization with Bayesian networks. In Ochoa et al. [137], pages 332–339.

[57] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann, 1993.

[58] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[59] N. Friedman and Z. Yakhini. On the sample complexity of learning Bayesian networks. In E. Horvitz and F. V. Jensen, editors, *UAI XII*, pages 274–282. Morgan Kaufmann, 1996.

[60] J. C. Gallagher and S. Vigraham. A Modified Compact Genetic Algorithm For The Intrinsic Evolution of continuous time recurrent neural networks. In Langdon et al. [99], pages 163–170.

[61] J. C. Gallagher, S. Vigraham, and G. R. Kramer. A family of compact Genetic Algorithms for intrinsic evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 8(2):111–126, 2004.

[62] M. R. Gallagher. *Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling*. PhD thesis, University of Queensland, 2000.

[63] M. R. Gallagher, M. Frean, and T. Downs. Real-valued evolutionary optimization using a flexible probability density estimator. In Banzhaf et al. [12], pages 840–846.

[64] Y. Gao and J. Culberson. Space Complexity of Estimation of Distribution Algorithms. *Evolutionary Computation*, 13(1):125–143, 2005.

[65] R. García, G. López-Cueto, M. Ostra, and C. Ubide. Multicomponent determinations using addition-generated reagent profiles and partial least squares regression. *Analytica Chimica Acta*, 535:287–295, 2005.

[66] D. Geiger and D. Heckerman. Learning Gaussian networks. In R. López de Mántaras and D. Poole, editors, *UAI*, pages 235–243. Morgan Kaufmann, 1994.

[67] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, Reading MA, 1989.

[68] C. González, J. A. Lozano, and P. Larrañaga. Analyzing the PBIL algorithm by means of discrete dynamical systems. *Complex Systems*, 12(4), 2001.

[69] C. González, J. A. Lozano, and P. Larrañaga. Mathematical modelling of umda_{c} algorithm with tournament selection. Behaviour on linear and quadratic functions. *International Journal of Approximate Reasoning*, 31(3):313–340, 2002.

[70] C. González, J. D. Rodríguez, J. A. Lozano, and P. Larrañaga. Analysis of the Univariate Marginal Distribution Algorithm modeled by Markov chains. In J. Mira and J. R. Álvarez, editors, *IWANN 2003*, volume 2686 of *Lecture Notes in Computer Science*, pages 510–517. Springer, 2003.

[71] T. Gosling. The simple supply chain model and evolutionary computation. In Sarker et al. [178], pages 2322–2329.

[72] J. Grahl, S. Minner, and F. Rothlauf. Behaviour of UMDAc with truncation selection on monotonous functions. In Corne et al. [45], pages 2553–2559.

[73] J. Grahl and F. Rothlauf. PolyEDA: Combining Estimation of Distribution Algorithms and linear inequality constraints. In Deb et al. [53], pages 1174–1185.

[74] J. J. Grefenstette. Optimization of control parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[75] M. A. Hall and L. A. Smith. Feature subset selection: A correlation based filter approach. In N. Kasabov et al, editor, *Proceedings of the Fourth International Conference on Neural Information Processing and Intelligent Information Systems*, pages 855–858, Dunedin, 1997.

[76] H. Handa. Hybridization of Estimation of Distribution Algorithms with a repair method for solving constraint satisfaction problems. In Cantú-Paz et al. [39], pages 991–1002.

[77] H. Handa. Estimation of Distribution Algorithms with mutation. In G. R. Raidl and J. Gottlieb, editors, *EvoCOP*, volume 3448 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2005.

[78] G. Harik. Linkage learning in via probabilistic modeling in the EcGA. Technical Report 99010, IlliGAL, 1999.

[79] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact Genetic Algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, 1999.

[80] K. Hasegawa, T. Kimura, and K. Funatsu. GA Strategy for Variable Selection in QSAR Studies: Application of GA-Based Region Selection to a 3D-QSAR Study of Acetylcholinesterase Inhibitors. *Journal of Chemical Information and Computer Sciences*, 39 (1):112–120, 1999.

[81] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[82] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Ross D. Shachter, Tod S. Levitt, Laveen N. Kanal, and John F. Lemmer, editors, *UAI*, pages 149–163. North-Holland, 1988.

[83] J. I. Hidalgo, M. Prieto, J. Lanchares, R. Baraglia, F. Tirado, and O. Garnica. Hybrid Parallelization of a Compact Genetic Algorithm. In *Euro PDP*, pages 449–455. IEEE Computer Society, 2003.

[84] T. Hiroyasu, M. Miki, M. Sano, H. Shimosaka, S. Tsutsui, and J. Dongarra. Distributed Probabilistic Model-Building Genetic Algorithm. In Cantú-Paz et al. [39], pages 1015–1028.

[85] M. Höhfeld and G. Rudolph. *Towards a Theory of Population-Based Incremental Learning.* International Conference on Evolutionary Computation , Indianapolis, USA, April 13-16, 1997.

[86] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1992.

[87] Y. Hong, Q. Ren, and J. Zeng. Adaptive population size for Univariate Marginal Distribution Algorithm. In Corne et al. [45], pages 1396–1402.

[88] Y. Hong, Q. Ren, and J. Zeng. Genetic drift in Univariate Marginal Distribution Algorithm. In Beyer et al. [17], pages 745–746.

[89] Y. Hong, Q. Ren, and J. Zeng. Optimization of noisy fitness functions with Univariate Marginal Distribution Algorithm. In Corne et al. [45], pages 1410–1417.

[90] I. Inza, P. Larrañaga, R. Etxeberria, and B. Sierra. Feature Subset Selection by Bayesian networks based optimization. *Artificial Intelligence*, 123(1-2):157–184, 2000.

[91] I. Inza, P. Larrañaga, and B. Sierra. *Data Mining: A Heuristic Approach*, chapter Estimation of Distribution Algorithms for Feature Subset Selection in large dimensionality domains, pages 97–116. H. Abbass, R. Sarker, C. Newton eds., IDEA Group Publishing, 2002.

[92] I. Inza, P. Larrañaga, and B. Sierra. Feature Subset Selection by Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 269–294. Kluwer Academic Publishers, 2002.

[93] I. Inza, P. Larrañaga, and B. Sierra. Feature weighting in K-NN by means of Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 295–311. Kluwer Academic Publishers, 2002.

[94] I. Inza, M. Merino, P. Larrañaga, J. Quiroga, B. Sierra, and M. Girala. Feature Subset Selection by Genetic Algorithms and Estimation of Distribution Algorithms - a case study in the survival of cirrhotic patients treated with tips. *Artificial Intelligence in Medicine*, 23(2):187–205, 2001.

[95] M. H. Jin, C. Y. Kao, Y. C. Huang, D. F. Hsu, R. G. Lee, and C. K. Lee. Compact Genetic Algorithm for active interval scheduling in hierarchical sensor networks. In Beyer et al. [17], pages 2205–2206.

[96] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3(1):77–112, 2004.

[97] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3(3):355–356, 2004.

[98] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[99] W. B. Langdon, E. Cantú-Paz, K. E. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. K. Burke, and N. Jonoska, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, New York, USA, July 9-13, 2002*. Morgan Kaufmann, 2002.

[100] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. In C. Boutilier and M. Goldszmidt, editors, *UAI*, pages 343–352. Morgan Kaufmann, 2000.

[101] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.

[102] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. In Whitley et al. [208], pages 201–204.

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

[104] P. Larrañaga, J. A. Lozano, and E. Bengoetxea. Estimation of Distribution Algorithms based on multivariate normal and Gaussian networks. Technical Report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 2001.

[105] M. Laumanns and J. Ocenasek. Bayesian optimization algorithms for multi-objective optimization. In Merelo-Guervós et al. [122], pages 298–307.

[106] S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[107] R. Leardi and A. Lupiañez. Genetic Algorithms applied to feature selection in PLS regression: how and when to use them. *Chemometrics and Intelligent Laboratory Systems*, 41:195–207, 1998.

[108] C. F. Lima, K. Sastry, D. E. Goldberg, and F. G. Lobo. Combining competent crossover and mutation operators: a probabilistic model building approach. In Beyer et al. [17], pages 735–742.

[109] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[110] X. Llorà and D. E. Goldberg. Wise breeding GA via machine learning techniques for function optimization. In Cantú-Paz et al. [39], pages 1172–1183.

[111] G. López-Cueto, M. Ostra, and C. Ubide. New way of application of the bromate-bromide mixture in kinetic analysis. *Analytica Chimica Acta*, 445:117–126, 2001.

[112] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, volume 192 of *Studies in Fuzzines and Soft Computing*. Springer, 2005.

[113] J. A. Lozano and A. Mendiburu. Solving job schedulling with Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 231–242. Kluwer Academic Publishers, 2002.

[114] J.A. Lozano, R. Sagarna, and P. Larrañaga. Parallel Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, pages 129–145. Kluwer Academic Publishers, 2002.

[115] Q. Lu and X. Yao. Clustering and learning Gaussian distribution for continuous optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 35(2): 195–204, 2005.

[116] J. Madera, E. Alba, and A. Ochoa. A Parallel Island Model for Estimation of Distribution Algorithms. In J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 159–186. Springer, 2005.

[117] T. Mahnig and H. Mühlenbein. Mathematical analysis of optimization methods using search distributions. In A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pages 205–208, 2000.

[118] B. Maxwell and S. Anderson. Training hidden markov models using population-based learning. In Banzhaf et al. [12], page 944.

[119] James L. McClelland and David E. Rumelhart. *Parallel Distributed Processing, Volume 2:Psychological and Biological Models, (ed. with PDP Research grp)*. MIT Press Cambridge, Massachusetts, 1986, 1986.

[120] A. Mendiburu, J. A. Lozano, and J. Miguel-Alonso. Parallel implementation of EDAs based on probabilistic graphical models. *IEEE Transactions on Evolutionary Computation*, 9(4):406–423, 2005.

[121] A. Mendiburu, J. Miguel-Alonso, J. A. Lozano, M. Ostra, and C. Ubide. Parallel and multi-objective EDAs to create multivariate calibration models for quantitative chemical applications. In Skie and Yang [194], pages 596–603.

[122] J. J. Merelo-Guervós, P. Adamidis, H. G. Beyer, J. L. Fernández-Villacañas Martín, and H. P. Schwefel, editors. *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, PPSN VII, Granada, Spain, September 7-11, 2002*, volume 2439 of *Lecture Notes in Computer Science*. Springer, 2002.

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

[124] H. Mühlenbein and T. Mahnig. Convergence theory and applications of the factorized distribution algorithm. *Journal of Computing and Information Technology*, 7:19–32, 1999.

[125] H. Mühlenbein and T. Mahnig. The factorized distribution algorithm for additively decomposed functions. In Ochoa et al. [137], pages 301–313.

[126] H. Mühlenbein and T. Mahnig. Evolutionary algorithms: From recombination to search distributions. In L. Kallel, B. Naudts, and A. Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, pages 135–173. Springer, Berlin, 2001.

[127] H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5:215–247, 1999.

[128] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In H. M. Voigt, W. Ebeling, I. Rechenberger, and H. P. Schwefel, editors, *PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer, 1996.

[129] J. Ocenasek, S. Kern, N. Hansen, and P. Koumoutsakos. A mixed Bayesian optimization algorithm with variance adaptation. In Yao et al. [218], pages 352–361.

[130] J. Ocenasek and J. Schwarz. The parallel Bayesian optimization algorithm. In *Proceedings of the European Symposium on Computational Intelligence*, pages 61–67, 2000.

[131] J. Ocenasek and J. Schwarz. The distributed Bayesian optimization algorithm for combinatorial optimization. In *EUROGEN - Evolutionary Methods for Design, Optimisation and Control, CIMNE*, pages 115–120, 2001.

[132] J. Ocenasek and J. Schwarz. Estimation of distribution algorithm for mixed continuous-discrete optimization problems. In *2nd Euro-International Symposium on Computational Intelligence*, pages 227–232. IOS Press, Kosice, Slowakia, 2002.

[133] J. Ocenasek, J. Schwarz, and M. Pelikan. Design of multithreaded Estimation of Distribution Algorithms. In Cantú-Paz et al. [40], pages 1247–1258.

[134] A. Ochoa, H. Mühlenbein, and M. Soto. Factorized Distribution Algorithm using Bayesian networks. In A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pages 212–215, 2000.

[135] A. Ochoa, H. Mühlenbein, and M. Soto. A Factorized Distribution Algorithm using single connected Bayesian networks. In Schoenauer et al. [182], pages 787–796.

[136] A. Ochoa, M. Soto, R. Santana, J. Madera, and N. Jorge. The factorized distribution algorithm and the junction tree: A learning perspective. In Ochoa et al. [137], pages 368–377.

[137] A. Ochoa, M. R. Soto, and R. Santana, editors. *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, Habana, Cuba, 1999.

[138] T. Okabe, Y. Jin, B. Sendhoff, and M. Olhofer. Voronoi-based estimation of distribution algorithm for multi-objective optimization. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1594–1601, Portland, Oregon, 20-23 June 2004. IEEE Press.

[139] T. K. Paul and H. Iba. Optimization in continuous domain by real-coded estimation of distribution algorithm. In A. Abraham, M. Köppen, and K. Franke, editors, *HIS*, volume 105 of *Frontiers in Artificial Intelligence and Applications*, pages 262–271. IOS Press, 2003.

[140] T. K. Paul and H. Iba. Reinforcement Learning Estimation of Distribution Algorithm. In Cantú-Paz et al. [40], pages 1259–1270.

[141] T. K. Paul and H. Iba. Identification of informative genes for molecular classification using Probabilistic Model Building Genetic Algorithm. In Deb et al. [53], pages 414–425.

[142] T.K. Paul and H. Iba. Linear and combinatorial optimizations by Estimation of Distribution Algorithms. In *Proceedings of the 9th MPS Symposium on Evolutionary Computation*, pages 99–106, 2003.

[143] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms*, volume 170 of *Studies in Fuzzines and Soft Computing*. Springer, 2005.

[144] M. Pelikan and D. E. Goldberg. Genetic Algorithms, clustering, and the breaking of symmetry. In Schoenauer et al. [182], pages 385–394.

[145] M. Pelikan and D. E. Goldberg. Hierarchical problem solving and the Bayesian optimization algorithm. In D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H.G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 267–274, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[146] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent Genetic Algorithms. In Spector et al. [197], pages 511–518.

[147] M. Pelikan and D. E. Goldberg. Hierarchical BOA solves Ising spin glasses and MAXSAT. In Cantú-Paz et al. [40], pages 1271–1282.

[148] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In Banzhaf et al. [12], pages 525–532.

[149] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation and Bayesian networks. *Evolutionary Computation*, 8(3):311–340, 2000.

[150] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.

[151] M. Pelikan, David E. Goldberg, and K. Sastry. Bayesian Optimization Algorithm, Decision Graphs, and Occam's Razor. In Spector et al. [197], pages 519–526.

[152] M. Pelikan and T. K. Lin. Parameter-less hierarchical BOA. In Deb et al. [52], pages 24–35.

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

[154] M. Pelikan and K. Sastry. Fitness inheritance in the Bayesian optimization algorithm. In Deb et al. [52], pages 48–59.

[155] M. Pelikan, K. Sastry, and D. E. Goldberg. Multiobjective hBOA, clustering, and scalability. In Beyer et al. [17], pages 663–670.

[156] J. M. Peña, J. A. Lozano, and P. Larrañaga. Benefits of data clustering in multimodal function optimization via EDAs. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, pages 101–127, 2002.

[157] J. M. Peña, J. A. Lozano, and P. Larrañaga. Globally Multimodal Problem Optimization Via an Estimation of Distribution Algorithm Based on Unsupervised Learning of Bayesian Networks. *Evolutionary Computation*, 13(1):43–66, 2005.

[158] P. Posik. Distribution tree-building real-valued evolutionary algorithm. In Yao et al. [218], pages 372–381.

[159] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004.

[160] R. Rastegar and M. R. Meybodi. A new estimation of distribution algorithm based on learning automata. In Corne et al. [45], pages 1982–1987.

[161] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann–Holzboog, Stuttgart, 1973.

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

[163] J. Rissanen. Modeling by shortest data description. Automatica , 465–471, 1978.

[164] J. Rivera. Using Estimation of Distribution Algorithms as an evolutive component of the XCS classifier system. Technical report, University of La Habana (In spanish), 1999.

[165] V. Robles, P. de Miguel, and P. Larrañaga. Solving the Traveling Salesman Problem with Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, pages 211–229. Kluwer Academic Publishers, 2002.

[166] T. Romero, P. Larrañaga, and B. Sierra. Learning Bayesian networks in the space of orderings with Estimation of Distribution Algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(4):607–625, 2004.

[167] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1962.

[168] J. Roure, P. Larrañaga, and R. Sangüesa. An empirical comparison between k-means, GAs and EDAs in partitional clustering. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 343–360. Kluwer Academic Publishers, 2002.

[169] S. Rudlof and M. Köppen. *Stochastic hill climbing by vectors of normal distributions*. Proceedings of the First Online Workshop on Soft Computing (WSC1), Nagoya, Japan, 1996.

[170] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by backpropagating errors. *Nature*, 323:533–536, 1986.

[171] D. Ruta and B. Gabrys. Application of the evolutionary algorithms for classifier selection in multiple classifier systems with majority voting. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, pages 399–408. Springer, 2001.

[172] R. Sagarna and P. Larrañaga. Solving the knapsack problem with Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 195–209. Kluwer Academis Publishers, 2002.

[173] R. Sagarna and J. A. Lozano. On the performance of Estimation of Distribution Algorithms applied to software testing. *Applied Artificial Intelligence*, 19(5):457–489, 2005.

[174] R. Sagarna and J. A. Lozano. Scatter search in software testing, comparison and collaboration with Estimation of Distribution Algorithms. *European Journal of Operational Research*, 169(2):392–412, 2006.

[175] R. Santana. Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97, 2005.

[176] R. Santana and A. Ochoa. Dealing with constraints with Estimation of Distribution Algorithms: The univariate case. In Ochoa et al. [137], pages 378–384.

[177] R. Santana, F. B. Pereira, E. Costa, A. Ochoa-Rodriguez, P. Machado, A. Cardoso, and M. R. Soto. Probabilistic evolution and the Busy Beaver problem. In Whitley et al. [208], page 380.

[178] R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors. *Proceedings of the 2003 Congress on Evolutionary Computation, CEC2003, Canberra, Australia, 8-12 December, 2003*. IEEE Press, 2003.

[179] K. Sastry, H. A. Abbass, D. E. Goldberg, and D. D. Johnson. Sub-structural niching in Estimation of Distribution Algorithms. In Beyer et al. [17], pages 671–678.

[180] K. Sastry and D. E. Goldberg. On Extended Compact Genetic Algorithm. In Whitley et al. [208], pages 352–359.

[181] K. Sastry and D. E. Goldberg. Designing competent mutation operators via probabilistic model building of neighborhoods. In Deb et al. [52], pages 114–125.

[182] M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo-Guervós, and H. P. Schwefel, editors. *Proceeding of the 6th International Conference on Parallel Problem Solving from Nature, PPSN VI, Paris, France, September 18-20, 2000*, volume 1917 of *Lecture Notes in Computer Science*. Springer, 2000.

[183] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 7(2):461–464, 1978.

[184] M. Sebag and A. Ducoulombier. Extending Population-Based Incremental Learning to continuous search spaces. In A. E. Eiben, T. Bäck, M. Schoenauer, and H. P. Schwefel, editors, *PPSN V*, volume 1498 of *Lecture Notes in Computer Science*. Springer, 1998.

[185] I. Servet, L. Trave-Massuyes, and D. Stern. Telephone network traffic overloading diagnosis and evolutionary techniques. In J. K. Hao, E. Lutton, E. M. A. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 137–144. Springer, 1998.

[186] R. Shachter and C. Kenley. Gaussian influence diagrams. *Management Science*, 35: 527–550, 1989.

[187] S. Shakya, J. McCall, and D. Brown. Updating the probability vector using MRF technique for a univariate EDA. In E. Onaindia and S. Staab, editors, *STAIRS 2004, Proceedings of the Second Starting AI Researchers' Symposium*, volume 109 of *Frontiers in Artificial Intelligence and Applications*, Valencia, Spain, August 2004. IOS Press.

[188] S. Shakya, J. McCall, and D. Brown. Incorporating a metropolis method in a distribution estimation using Markov random field algorithm. In Corne et al. [45], pages 2576–2583.

[189] S. Shakya, J. McCall, and D. Brown. Using a Markov network model in a univariate EDA: an empirical cost-benefit analysis. In Beyer et al. [17], pages 727–734.

[190] S. Shakya, J. McCall, and D. F. Brown. Estimating the distribution in an EDA. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, editors, *Adaptive and Natural Computing Algorithms*, Springer Computer Series, pages 202–205, Coimbra, Portugal, 21-23 March 2005. Springer.

[191] J. L. Shapiro. Drift and Scaling in Estimation of Distribution Algorithms. *Evolutionary Computation*, 13(1):99–123, 2005.

[192] H. Shimosaka, T. Hiroyasu, and M. Miki. Comparison of pulling back and penalty methods for constraints in BGA. In Sarker et al. [178], pages 1941–1948.

[193] B. Sierra, E. Jiménez, I. Inza, P. Larrañaga, and J. Muruzábal. Rule induction using Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 313–322. Kluwer Academic Publishers, 2002.

[194] T. Skie and C. S. Yang, editors. *34th International Conference on Parallel Processing Workshops (ICPP 2005 Workshops), 14-17 June 2005, Oslo, Norway*. IEEE Computer Society, 2005.

[195] P. W. F. Smith and J. Whittaker. Edge exclusion tests for graphical Gaussian models. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 555–574. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[196] M. Soto, A. Ochoa, S. Acid, and L. M. de Campos. Introducing the polytree aproximation of distribution algorithm. In Ochoa et al. [137], pages 360–367.

[197] L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2001, San Francisco, California, USA, July 7-11, 2001*. Morgan Kaufmann, 2001.

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

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

[200] R. Sukthankar, S. Baluja, and J. Hancock. Multiple adaptive agents for tactical driving. *Applied Intelligence*, 9(1):7–23, 1998.

[201] G. Syswerda. Simulated crossover in Genetic Algorithms. In *Foundations of Genetic Algorithms*, volume 2, pages 239–255, San Mateo, CA, 1993. Morgan Kaufmann.

[202] M. Tsuji, M. Munetomo, and K. Akama. Modeling dependencies of loci with string classification according to fitness differences. In Deb et al. [52], pages 246–257.

[203] S. Tsutsui. Probabilistic Model-Building Genetic Algorithms in permutation representation domain using edge histogram. In Merelo-Guervós et al. [122], pages 224–233.

[204] S. Tsutsui. Solving flow shop scheduling problems with probabilistic model-building Genetic Algorithms using edge histograms. In L. Wang, K. C. Tan, T. Furuhashi, J. Kim, and X. Yao, editors, *SEAL*, volume 2, pages 465–471, 2002.

[205] S. Tsutsui, T. Hiroyasu, and M. Miki. The effect of introducing a tag node in solving scheduling problems using edge histogram based sampling algorithms. In Sarker et al. [178], pages 22–29.

[206] N. Villegas. *Desenvolupament de procediments cinètics per l'anàlisi de multicomponents*. PhD thesis, Universidad Autónoma de Barcelona, 2003.

[207] R. Wehrens and B. H. Mevik. *PLS: Partial Least Squares Regression (PLSR) and Principal Component Regression (PCR)*, 2005. R package version 1.0-2.

[208] L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H. G. Beyer, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2000, Las Vegas, Nevada, USA, July 8-12, 2000*. Morgan Kaufmann, 2000.

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

[210] S. Wold, M. Sjöström, and L. Eriksson. PLS-regression: a basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, 58:109–130, 2001.

[211] A. H. Wright, R. Poli, C. R. Stephens, W. B. Langdon, and S. Pulavarty. An estimation of distribution algorithm based on maximum entropy. In Deb et al. [52], pages 343–354.

[212] L. Xu, A. Krzyzak, and E. Oja. Rival penalized competitive learning for clustering analysis RBF net and curve detection. *IEEE Transactions on Neural Networks*, 4(4): 636–649, 1993.

[213] K. Yanai and H. Iba. Estimation of distribution programming based on Bayesian network. In Sarker et al. [178], pages 1618–1625.

[214] K. Yanai and H. Iba. Program evolution by integrating EDP and GP. In Deb et al. [53], pages 774–785.

[215] S. Yang. Memory-enhanced Univariate Marginal Distribution Algorithms for Dynamic Optimization Problems. In Corne et al. [45], pages 2560–2567.

[216] S. Yang. Population-Based Incremental Learning with memory scheme for changing environments. In Beyer et al. [17], pages 711–718.

[217] S. Yang and X. Yao. Experimental study on polulation-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11):815–834, 2005.

[218] X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiño, A. Kabán, and H. P. Schwefel, editors. *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, PPSN VIII, Birmingham, UK, September 18-22, 2004*, volume 3242 of *Lecture Notes in Computer Science*. Springer, 2004.

[219] B. Yuan and M. Gallagher. Playing in continuous spaces: Some analysis and extension of Population-Based Incremental Learning. In Sarker et al. [178], pages 443–450.

[220] B. Yuan and M. Gallagher. Experimental results for the special session on real-parameter optimization at cec 2005: A simple, continuous EDA. In Corne et al. [45], pages 1792–1799.

[221] B. Yuan and M. Gallagher. On the importance of diversity maintenance in Estimation of Distribution Algorithms. In Beyer et al. [17], pages 719–726.

[222] B. Yuan, M. Gallagher, and S. Crozier. MRI magnet design: search space analysis, EDAs and a real-world problem with significant dependencies. In Beyer et al. [17], pages 2141–2148.

[223] Q. Zhang and H. Mühlenbein. On global convergence of FDA with proportionate selection. In Ochoa et al. [137], pages 340–343.

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