Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 66 (2006) 1002–1013
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Parallel EDAs to create multivariate calibration models for quantitative chemical applications

A. Mendiburu[a,*,1], J. Miguel-Alonso[a,2], J.A. Lozano[b,1], M. Ostra[c,3], C. Ubide[c,3]

[a]*Department of Computer Architecture and Technology, University of the Basque Country, P. Manuel Lardizabal, 1. C.P. 20018, San Sebastian, Gipuzkoa, Spain*
[b]*Department of Computer Science and Artificial Intelligence, The University of the Basque Country, San Sebastián, Spain*
[c]*Department of Applied Chemistry, The University of the Basque Country, San Sebastián, Spain*

Received 3 May 2005; received in revised form 24 February 2006; accepted 3 March 2006
Available online 17 April 2006

Abstract

This paper describes the application of a collection of data mining methods to solve a calibration problem in a quantitative chemistry environment. Experimental data obtained from reactions which involve known concentrations of two or more components are used to calibrate a model that, later, will be used to predict the (unknown) concentrations of those components in a new reaction. This problem can be seen as a selection + prediction one, where the goal is to obtain good values for the variables to predict while minimizing the number of the input variables needed, taking a small subset of really significant ones. Initial approaches to the problem were principal components analysis and filtering combined with two prediction techniques: artificial neural networks and partial least squares regression. Finally, a parallel estimation of distribution algorithm was used to reduce the number of variables to be used for prediction, yielding the best models for all the considered problems.

Keywords: Evolutionary algorithms; Estimation of distribution algorithms; Parallel computing; Artificial neural network; Partial least squares regression; Feature extraction and construction; Chemical calibration models

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 information is easily stored into a personal computer, opening new possibilities for handling these large volumes of data. 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 the UV–Vis spectrum can be made dependent exclusively on the concentration of 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 make 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,

* Corresponding author.
E-mail addresses: amendiburu@si.ehu.es (A. Mendiburu), j.miguel@ehu.es (J. Miguel-Alonso), lozano@si.ehu.es (J.A. Lozano), miren.ostra@ehu.es (M. Ostra), carlos.ubide@ehu.es (C. Ubide).

[1] Work done with the support of the Spain's MEC (TIN 2005-03824) and the Basque Government (ETORTEK-NASH).
[2] Work done with the support of the Spain's MEC (TIN2004-07440-C02-02).
[3] Work done with the support of the Spain's MEC (PPQ2003-07318-C02-02).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.03.001

enough experimental matrices of data are obtained for different and known concentrations of the species of interest. Then, all that 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 the different multivariate calibration algorithms, Partial least squares regression (PLS) and artificial neural networks (ANN) have frequently been used. Nowadays, PLS is, by far, the most widely used algorithm because it was specifically developed to provide optimum prediction ability. However, ANN is claimed to provide better results in cases where non-linear systems are involved; the reason is that ANNs are intrinsically non-linear, whereas PLS is a linear algorithm [2]. 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 their 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. Several approaches can be used for this size reduction: feature construction and feature subset selection (FSS) [15]. 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 FSS, two main approaches must be mentioned: 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 built with that subset of variables.

In the search of a simple solution for the problem, two techniques 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 estimation of distribution algorithms (EDAs) to implement a wrapper-based selection of variables. Among the different solutions (filter and/or wrapper) that have been presented in the last years, we decided to use parallel EDAs because of their promising results in previous works in the FSS area [11,12], although we are conscious that other efficient approaches exist [5]. Additionally, as EDAs require long computation times, we present a parallel implementation that makes their application viable when facing complex problems that would be unaffordable if using a single computer.

The rest of the paper is organized as follows: Section 2 begins with a description of the two prediction techniques (ANN and PLS) used to complete the calibration. Section 3 introduces three example problems used throughout this paper to evaluate our proposals. The different approaches developed for each prediction technique are shown in Sections 4 (ANN) and 5 (PLS). After an introduction to EDAs and parallel EDAs in Section 6, we present in Section 7 a filter + wrapper solution based on these algorithms. Section 8 summarizes the results obtained with all the different techniques and, finally, Section 9 ends with some conclusions as well as proposals for future work.

2. Prediction techniques

2.1. Artificial neural networks

Essentially, an ANN [17] 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) model [23]. 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 [24].

2.2. Partial least squares regression

PLS Regression [28] is a common tool in chemistry, used to analyze two data matrices **X** and **Y** by a linear multivariate model. In PLS, the structure of **X** and also of **Y** is modelled to give richer results than the traditional multiple regression approach. This is achieved indirectly by extracting latent variables **T** and **U** from matrices **X** and **Y**, respectively. The extracted factors **T** (also referred to as **X**-scores) are used to predict the **Y**-scores **U**, and then the predicted **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.

3. Chemical reactions used in this study

The problems used throughout this paper 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, calibration models will be created. The features (variables) that

Table 1
Characteristics of the problems used in the experiments

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

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 consists 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 paper 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 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 [16]. The problem defined as *medium* corresponds to mixtures of three related 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 [8]. The problem defined as large corresponds to mixtures of formaldehyde, glyoxal and glutaraldehyde. These mixtures react with 3-methyl-2-benzothiazolone hydrazone [26]. In Table 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).

4. MLP approach

With this choice of calibration algorithm, we need to consider the way it is used. The problems have more than 1000 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 to 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 five-fold cross-validation to measure the accuracy of the different models proposed. 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. As fitness function, 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, that is, mean square error of prediction (MSEP). 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.

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. 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 have 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.

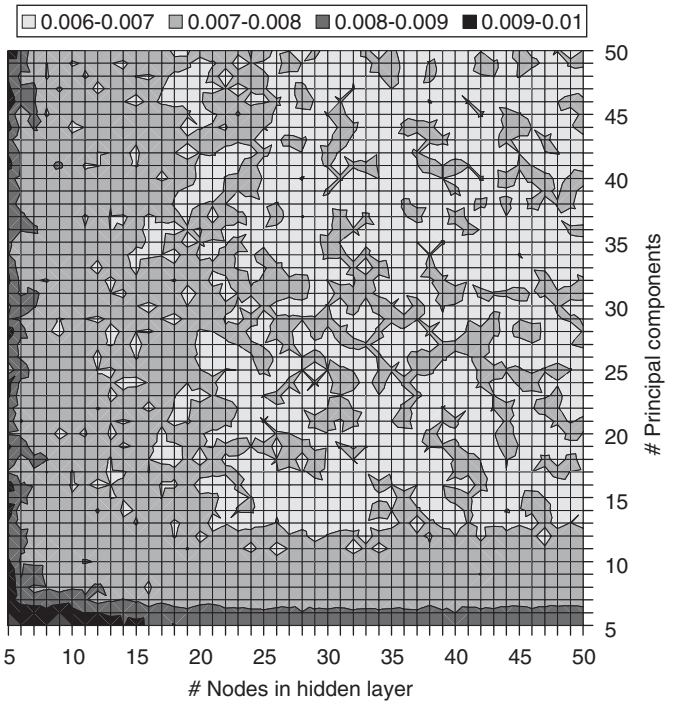

Fig. 1. *Small* problem. MSEP for different PCA + MLP configurations.

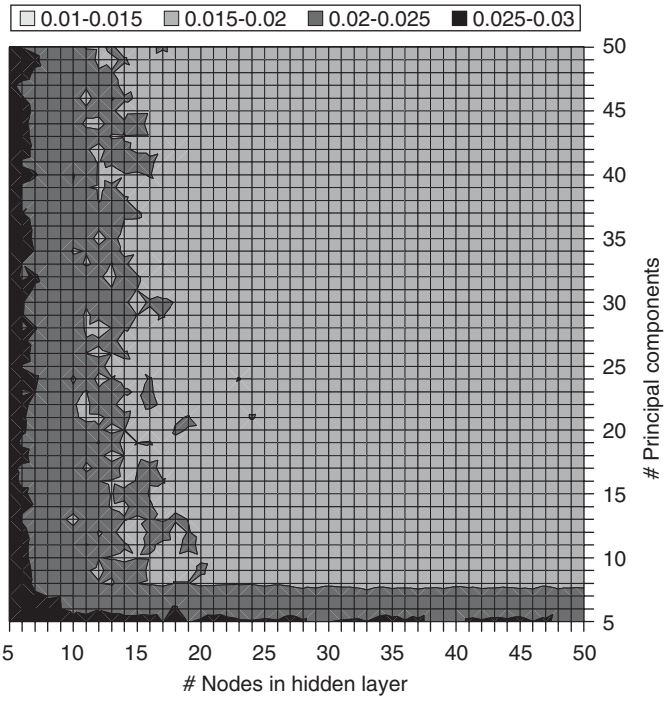

Fig. 2. *Medium* problem. MSEP for different PCA + MLP configurations.

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 5–50 principal components, and 5–50 intermediate nodes. Obtained results (for the three problems) are plotted in error maps (Figs. 1–3) where each map point (x, y) represents the MSEP 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 seven 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 Fig. 3), although this does not happen with the other two problems (see Figs. 1 and 2). Taking into account the propensity to overfitting 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 2.

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 paper is the correlation-based feature selection (CFS), introduced in [9]. 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

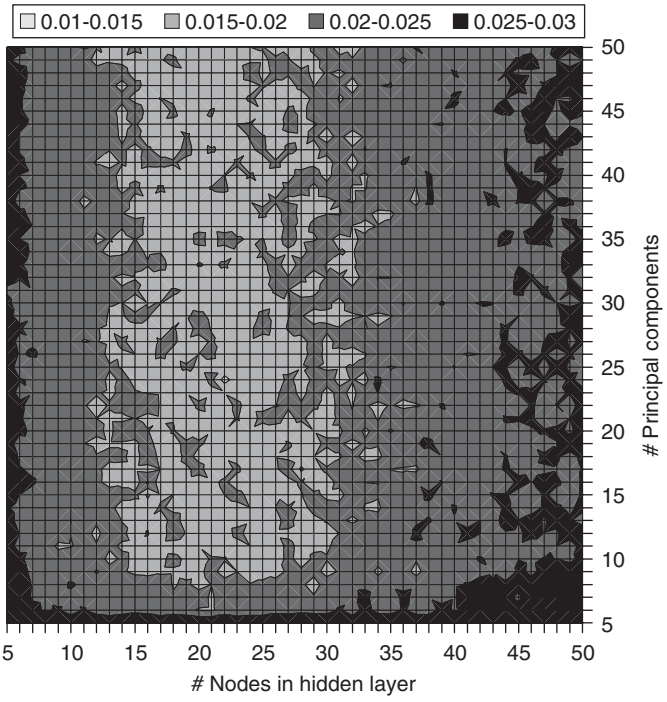

Fig. 3. *Large* problem. MSEP for different PCA + MLP configurations.

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 [6].

The filtering process, when applied to our problems, obtained a subset of a few relevant variables, 33, 55 and 31 for the

Table 2
Input variables, # neurons in the hidden layer and MSEP for the 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	$1.057e^{-2}$

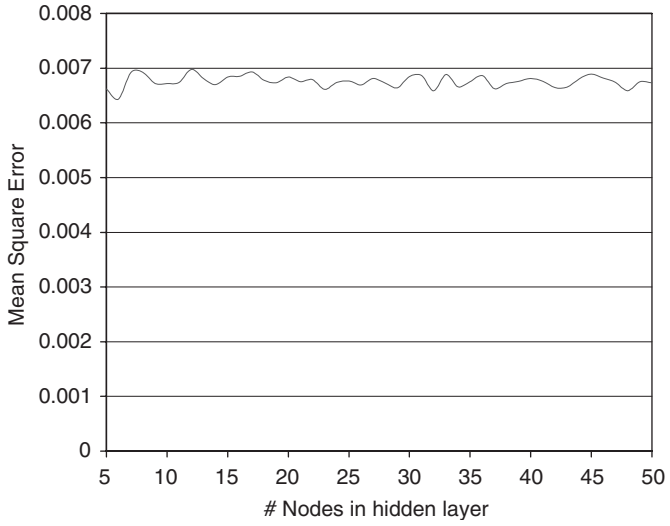

Fig. 4. *Small* problem. MSEP for different Filter + MLP configurations.

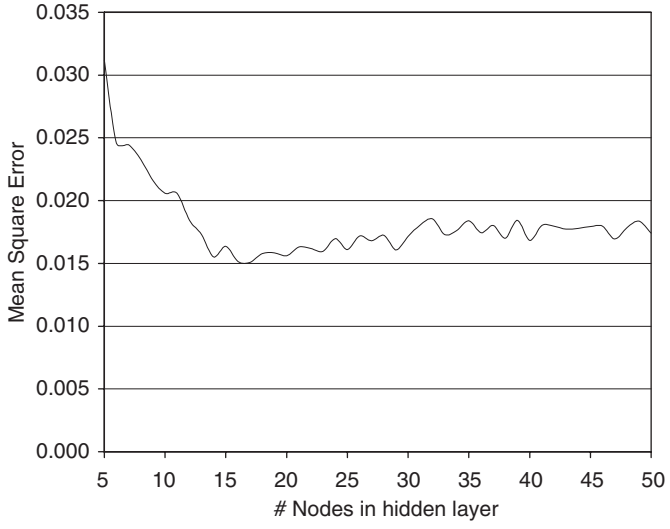

Fig. 5. *Medium* problem. MSEP for different Filter + MLP configurations.

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 Figs. 4–6. 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

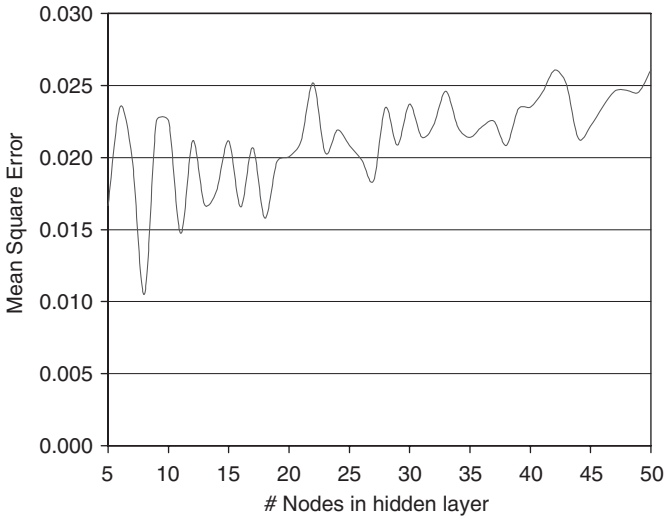

Fig. 6. *Large* problem. MSEP for different Filter + MLP configurations.

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 2.

5. PLS approach

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 filter process is performed before applying PLS. Experiments were carried out using the PLS package [27] from the R statistical computing environment [22]. As done for MLP, a five-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 smallest MSEP. In all the runs, the best models comprised less than 20 components. Results obtained with this approximation improves notably those obtained with any of the previous MLP-based configurations (see Table 3).

5.2. Filter and PLS

PLS has been applied over the same filtered set used for MLP. Table 3 summarizes the obtained average error. Note that

Table 3
Input variables, # principal components and MSEP for the 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	$6.644e^{-3}$

the filtering yields worse results for all the problems. However, previous works [14,10] 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 and, due to the large number of features in our datasets, we developed a parallel version for a particular EDA, trying to obtain a tool able to manage problems with more than a few hundred features.

6. EDAs and parallel approaches

EDAs were introduced in [20]. Their structure is similar to genetic algorithms (GAs) but, instead of using crossover or mutation operators to obtain the new population, they sample this population from a probability distribution. This probability distribution is estimated from the database that contains a selection of the individuals of the previous population. 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. A common scheme for all EDAs can be given as: (1) generate an initial population of M individuals and evaluate each of them, (2) select N individuals from the set of M, following a given selection method, (3) learn a probability model from the set of selected individuals, and (4) finally, generate (and evaluate) a new population of M individuals, based on the sampling of the probability distribution learnt in the previous step.

Steps 2–4 are repeated until some stop criterion is met (e.g., a maximum number of generations or no improvement after a certain number of generations).

The most important step is the third one: the way the dependencies between variables are learnt. Looking at this step, EDAs can be classified in three families: (1) those where all the variables are considered independent, (2) those that only consider dependencies between pairs of variables, and (3) those that consider multiple dependencies. Detailed information about the different approaches that constitute the family of EDAs and their characteristics can be obtained in [13,21].

Our interest in EDAs for FSS is motivated by results obtained in previous works [11,12]. However, these works reported the difficulty of applying multivariate EDAs (for example, Estimation of Bayesian Networks Algorithms (EBNAs)) on problems with more than 100 variables due to the high computational cost.

Therefore, we want to accelerate this kind of algorithms, designing a parallel approach for one of them (the EBNA_{BIC} algorithm). In the following sections, a description of the algorithm as well as the parallelization scheme are presented.

6.1. Description of the sequential EBNA_{BIC}

The learning phase of EBNA_{BIC} involves the learning of a Bayesian network from the selected individuals at each generation, that is, learning the structure (the graph S) and the parameters ($\boldsymbol{\theta}$, that codify the local probability distributions).

In [19] some parallel approaches for different EDAs were presented, focusing on the learning phase. In those experiments very simple fitness functions (evaluation of the individuals) were used. However, in problems like the one presented in this paper, the evaluation of the fitness function can take a sizeable portion of the overall execution time.

Before designing a parallel approach for a sequential algorithm it is necessary to complete a previous analysis of the most time-consuming phases, in order to select those whose parallelization would be more profitable. We run the sequential version of EBNA_{BIC} to perform an FSS over the *medium* dataset (details will be given in Section 7), and found that the learning phase needs the 32% of the total CPU time, while the "sampling and evaluation" phase requires the 67%. These proportions were similar for the other two problems. Therefore, an efficient parallel algorithm must deal with both phases. In the following subsections we define these two phases and examine the approaches taken to accelerate their execution in a parallel computer.

6.1.1. The learning phase

Once the population is selected, a Bayesian network is learnt from it. In EBNA_{BIC}, a greedy approach is used to learn the structure of the Bayesian network. Each possible network structure will be assigned a score that represents its goodness for the current population. The search will be done adding or deleting edges to the existing Bayesian network when this addition or deletion implies a better score.

Obviously, the score used during this process plays an important role in the algorithm, as it conditions the obtained Bayesian network. EBNA_{BIC} uses the penalized maximum likelihood score denoted by BIC (Bayesian information criterion) [25]. Given a structure S and a dataset D (set of selected individuals), the BIC score can be defined as

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

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

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

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

At each step, an exhaustive search is made through the whole set of possible arc modifications. An arc modification consists of adding or deleting an arc from the current structure S. The modification that maximizes the gain of the BIC score is used to update S, as long as it results in a DAG structure. This cycle continues until there is no arc modification that improves the score. It is important to bear in mind that if we update S with the arc modification (j, i), then only BIC(i, S, D) needs to be recalculated.

The previous 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 dataset D, 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 (it is assumed that in the previous step, S was updated with the arc modification (j, i)) need to be considered. Other arc modifications have not changed its value because of the decomposable property of the score. In this case, the number of terms to be calculated is $n-2$.

The following data structures are used to implement the algorithm: (i) 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, (ii) a structure $S[i]$, $i = 1, 2, \ldots, n$, with the DAG represented as adjacency lists, (iii) a $n \times n$ matrix $G[j, i]$, $j, i = 1, \ldots, n$, where each (j, i) entry represents the gain or loss in score associated with the arc modification (j, i), and (iv) 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 data structure is used to check if an arc modification produces a DAG structure. The pseudo-code for the sequential structure learning algorithm is shown in Fig. 7.

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

Fig. 7. Pseudo-code for the sequential structural learning algorithm.

6.1.2. Parallelization of the learning phase

We have chosen a well-known design pattern for parallel programming: the master-worker paradigm. The master runs the sequential parts and when it reaches the costly phase(s), it distributes parts of its workload among a collection of workers. Then, it collects and summarizes the partial results, and continues normal operation. Sometimes, depending on the characteristics of the work that must be completed, it is interesting to design the master in such a way that it can make use of its idle time: while waiting for the workers to finish, the master completes part of the work just as another worker. This approach is particularly useful in small-scale parallel computers.

The paradigm was implemented using message passing interface (MPI). The selected programming language was C++.

Related to the algorithm, we have explained before that the learning phase (creation of the Bayesian network) is a greedy process where each network will be measured using the decomposable BIC criterion. So, each worker (master included) will compute a subset of variables returning the results to the master, which then updates the structure of the Bayesian network applying the arc addition/deletion that improves the score the most.

With regard to the execution scheme, once initialization has been completed, workers wait for a job request. Two different types of job requests can be received: (1) to compute the initial BIC scores for their respective set of variables—each worker receives a chunk of variables to work with—and (2) to update the BIC scores and maintain the integrity of the local structures—each node addition or deletion means that the master must notify the workers that the (i, j) edge has changed and they must update their own copy of the network as well as BIC scores. Figs. 8 and 9 show the pseudo-code for master and workers.

6.1.3. Parallelization of the "sampling and evaluation" phase

In [3] the author introduces a summary of ideas for the design of efficient parallel evolutionary algorithms, which include the parallelization of fitness evaluation. In our work, as we want to maintain exactly the same behavior of the sequential algorithm, we will use a single-population master-worker approach, in which each worker evaluates a subset of individuals. Additionally, instead of receiving from the manager the subset to evaluate, we decided (based on previous experiments) that the worker will also be responsible of sampling (creating) this subset. In this way, once the workers have finished,

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

Fig. 8. Parallel structural learning phase. Pseudo-code for the master.

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

Fig. 9. Parallel structural learning phase: pseudo-code for the workers.

the manager receives the new individuals together with their fitness value.

The parallel approach is as follows: being *NewPopSet* the amount of new individuals to be created and *comp* the number of computational nodes, the master sends to each worker the order to create and evaluate $NewPopSet/comp$ new individuals (note that the master plays also the worker role, creating and evaluating a portion of the new individuals). Once all these new individuals have been created and evaluated, they are gathered by the master, which adds them to the population and continues with the algorithm.

In our experiments we have always operated with sets of identical computers; this is the reason we assign the same (or similar) number of computations to each worker. In other heterogeneous situations, it would be better to use an on-demand

Input: *Order*, *Probs*
Step 1. send *Order* and *Probs* structures to the workers
Step 2. set the number of individuals ($NewPopSet$) to create
Step 3. send $NewPopSet$ to the workers
Step 4. receive the new individuals from the workers
update the population

Fig. 10. Pseudo-code for the parallel "sampling and evaluation" phase for the master.

Step 1. receive *Order* and *Probs* structures
Step 2. receive the number of individuals ($NewPopSet$) to create
Step 3. **for** $i = 1, \ldots, NewPopSet$
create a new individual and evaluate it
Step 4. send the new individuals to the master

Fig. 11. Pseudo-code for the parallel "sampling and evaluation" phase for the workers.

scheme, where only one piece of work is sent to each worker and, when it is completed, another one is sent, until all the computations have been terminated. This alternative provides better load balancing, but needs more communication than the one we use. Yet another possibility could be to previously measure the computation capacity of each computer, assigning afterwards to each of them a workload tailored to its capacity. The pseudo-codes for the parallel "sampling and evaluation" phase are shown in Figs. 10 (master) and 11 (workers).

7. Experiments using parallel EDAs and PLS

In order to adapt the EBNA_{BIC} algorithm to the FSS problem we are dealing with, the following choices were introduced: (i) each variable X_i of the individual can take one of two values: 0 or 1 (1 implies that the ith feature is selected and 0 that is not selected), (ii) in order to check the goodness of an individual, a PLS training-evaluation process (with a five-fold cross-validation) must be completed using as input the set of selected variables from the dataset. As explained before, MSEP is the fitness value for each individual. The smaller the error, the better the individual.

It is well known [7] 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 our example problems), this would mean that each time a new population is created, thousands of individuals should be evaluated (executing PLS with a five-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 2×6 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 [1])) for each target variable. From these partial rankings, a unique ranking for each target

Table 4
MSEP and deviations (considering all executions $_m$, and only the best execution $_b$)

	$Error_m$	Dev_m	$Error_b$	Dev_b
Small problem				
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 problem				
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 problem				
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	$3.154e^{-3}$	$6.016e^{-4}$	$2.855e^{-3}$	$7.730e^{-4}$

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 1000 was the size of the population. In each generation, 999 new individuals are created and the next population is obtained selecting the best 1000 individuals among the present population and the recently created individuals.

The results obtained using this EDA + PLS approach can be consulted in Table 4. This table includes and extends the results obtained by previous versions, showing MSEPs and deviations. These values have been obtained as follows:

- For MLP and PLS, five executions of the best configurations (in terms of input and hidden nodes for MLP, and number of components for PLS) have been completed, obtaining five MSEP in each execution (due to the five-fold cross-validation).
- For EDA, 10 executions have been completed, yielding 10 individuals (solutions). Then, these individuals have been used to execute PLS (with the selected features), obtaining five MSEP for each individual (due again to the five-fold cross-validation).

8. Discussion of results

Once the experiments have been carried out, we can analyze the obtained results from two points of view: quality of the calibration models (in terms of MSEP) and execution time required by each approach.

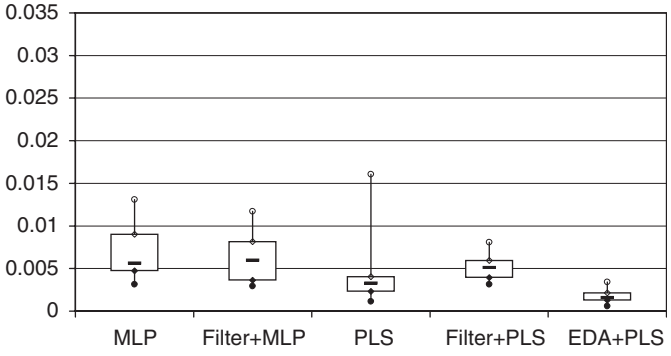

Fig. 12. *Small* problem. Boxplot of MSEPs.

8.1. Calibration accuracy

Results are provided in Table 4 and Figs. 12–14 (boxplot diagrams). 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), provide easily interpretable models.

In the quantitative chemistry field, a model is considered valid when the calibration error is equal or less than a 10%. We

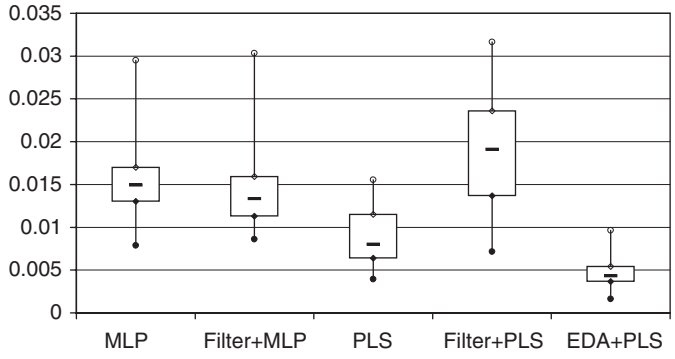

Fig. 13. *Medium* problem. Boxplot of MSEPs.

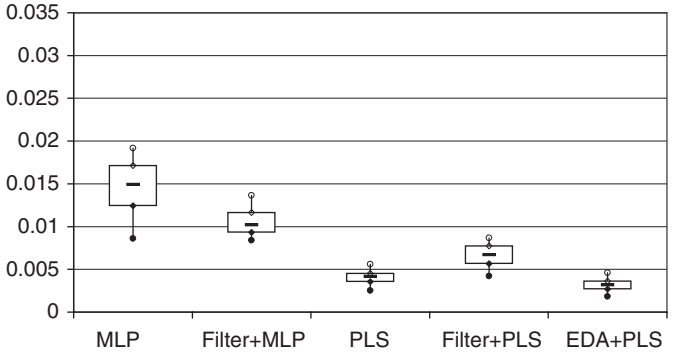

Fig. 14. *Large* problem. Boxplot of MSEPs.

have summarized in Table 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:

$$\text{Percentage error} = 100\sqrt{\frac{\sum_{i=1}^{c}(\hat{C}_i - C_i)^2}{\sum_{i=1}^{c} C_i^2}}, \qquad (3)$$

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 (we attribute this to the reduced number of cases in the dataset). In addition, this approach presents models that can be more interpretable as the number of features is reduced from thousands to four-six tens.

The conclusion of this section 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.

8.2. Execution time

All the experiments were completed in a 8-node Linux cluster. Each node has two processors (AMD Athlon MP 2000+, 1.6 GHz, 256 KB of cache memory) and 1 GB of (shared) RAM. Table 6 shows the execution times required by each approximation. It can be observed that PLS is a very fast method, requiring less than 1 min of execution time in a single processor. In contrast, as the utilization of a MLP requires the selection of the number of nodes in the input and hidden layers, the execution time required to explore a large range of these parameters is quite notable, even using the parallel cluster.

Finally, the last approximation (EDA + PLS) requires between 7 and 9 h to complete a run (using 8 processors). This is not, obviously, the best approach in terms of computational demands, but when resources are available, the accuracy of the resulting predictions may make the effort worthwhile.

When proposing a parallel approach to a problem, an important issue is its scalability: would the program run faster

Table 5
Mean error percentages (only for the best execution) for PLS and EDA+PLS approaches

	Small	*Medium*	*Large*
PLS	9.16	14.72	5.38
EDA + PLS	5.37	9.71	4.62

Table 6
Summary of execution times for each approach and problem using 8 CPUs

	Small	*Medium*	*Large*
PCA + MLP	2 h 06′	1 h 34′	4 h 43′
Filter + MLP	3′	2′	6′
PLS (1 CPU)	1′	1′	1′
Filter + PLS (1 CPU)	1′	1′	1′
EDA + PLS	6 h 54′	7 h 02′	9 h 13′

Table 7
Speed up and efficiency results for the EDA and PLS approach with the medium dataset

#CPUs	Time	Speed up	Efficiency
Sequential	51 h 53′25″	—	—
2	25 h 57′11″	2.00	1.00
4	13 h 18′24″	3.90	0.97
8	7 h 01′49″	7.38	0.92
16	3 h 46′01″	13.78	0.86

if I use more computing nodes? How faster? In order to see the scalability of the (parallel) EDA + PLS alternative, we executed it using different computing cluster sizes, for the medium dataset. Table 7 shows the speed up and efficiency for 2, 4, 8 and 16 processors. It must be pointed out that the speed up of the algorithm is almost linear, getting a good efficiency (86%) even when 16 CPUs are used.

9. Conclusions and future work

This paper presents some promising approaches to model the behavior of chemical reaction mixtures, and to relate it with the initial concentration of the chemical species taking part in it. This is done by a combination of variable reduction and multivariate calibration techniques.

As initial approaches, PCA and filtering were used with MLP, as well as PLS with and without filtering, with the aim of reducing the calibration error of the obtained models. Experiments with different problems show that any approach using PLS outperforms notably those based on MLP.

In a second phase, a filter technique was combined with a wrapper approach based on EDAs. Due to the high computational cost of this technique, a parallel approach has been developed for a particular EDA ($EBNA_{BIC}$), that allows the use of this algorithm to face problems that would be unaffordable with a sequential solution.

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 our study. In addition, it must be noted that the parallel EDA can be used not only in combination with PLS or MLP: any other algorithm 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.

The results achieved, together with the reasonable execution times, encourages us to further explore the utilization of EDAs in calibration processes. A line of work that shows potential is the use of multi-objective approaches (preliminary results are presented in [18]). Our plans also include the exploration of other algorithms inside the family of EDAs. Some preliminary work based on approaches proposed in [4] shows promising results using more simple algorithms (suggesting that algorithms like UMDA could get results similar to $EBNA_{BIC}$). However, it must be pointed out that, also for UMDA, the most time-consuming step is the "sampling and evaluation" phase and therefore, a parallel approach would still be necessary in order to obtain acceptable execution times.

Acknowledgment

The authors acknowledge Dr. S. Maspoch (Universidad Autónoma de Barcelona, Spain) for providing the large experimental data used in this work.

References

[1] M. Ben-Bassat, Use of distance measures, information measures and error bounds in feature evaluation, in: P.R. Krishnaiah, L.N. Kanal (Eds.), Handbook of Statistics, vol. 2, North-Holland, Amsterdam, 1982, pp. 773–791.
[2] M. Blanco, J. Coello, H. Iturriaga, S. Maspoch, M. Porcel, Use of circular dichroism and artificial neural networks for the kinetic-spectrophotometric resolution of enantiomers, Anal. Chim. Acta 431 (2001) 115–123.
[3] E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, Norwell, MA, USA, 2000.
[4] E. Cantú-Paz, Feature subset selection by estimation of distribution algorithms, in: 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, N. Jonoska (Eds.), GECCO, Morgan Kaufmann, Los Altos, CA, 2002, pp. 303–310.
[5] E. Cantú-Paz, Feature subset selection, class separability, and genetic algorithms, in: K. Deb et al. (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference GECCO-04, vol. 2, Springer, Berlin, 2004, pp. 959–970.
[6] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proceedings of the 13th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 1993, pp. 1022–1027.
[7] N. Friedman, Z. Yakhini, On the sample complexity of learning Bayesian networks, in: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence, 1996, pp. 274–282.
[8] R. García, G. López-Cueto, M. Ostra, C. Ubide, Multicomponent determinations using addition-generated reagent profiles and partial least squares regression, Anal. Chim. Acta 535 (2005) 287–295.
[9] M.A. Hall, L.A. Smith, Feature subset selection: a correlation based filter approach, in: N. Kasabov, et al. (Eds.), Proceedings of the Fourth International Conference on Neural Information Processing and Intelligent Information Systems, Dunedin, New York, 1997, pp. 855–858.
[10] K. Hasegawa, T. Kimura, K. Funatsu, GA Strategy for variable selection in QSAR studies: application of GA-based region selection to a 3D-QSAR study of acetylcholinesterase inhibitors, J. Chem. Inform. Comput. Sci. 39 (1) (1999) 112–120.
[11] I. Inza, P. Larrañaga, R. Etxeberria, B. Sierra, Feature subset selection by Bayesian networks based optimization, Artificial Intelligence 123 (1–2) (2000) 157–184.
[12] I. Inza, P. Larrañaga, B. Sierra, Feature subset selection by estimation of distribution algorithms, in: P. Larrañaga, J.A. Lozano (Eds.), Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, Kluwer Academic Publishers, Dordrecht, 2002, pp. 269–294.

[13] P. Larrañaga, J.A. Lozano (Eds.), Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer Academic Publishers, Dordrecht, 2002.
[14] R. Leardi, A. Lupiañez, Genetic algorithms applied to feature selection in PLS regression: how and when to use them, Chemometrics Intelligent Laboratory Syst. 41 (1998) 195–207.
[15] H. Liu, H. Motoda, Feature Selection for Knowledge Discovery and Data Mining, Kluwer Academic Publishers, Dordrecht, 1998.
[16] G. López-Cueto, M. Ostra, C. Ubide, New way of application of the bromate-bromide mixture in kinetic analysis, Anal. Chim. Acta 445 (2001) 117–126.
[17] J.L. McClelland, D.E. Rumelhart, Parallel Distributed Processing, vol. 2: Psychological and Biological Models, MIT Press, Cambridge, MA, 1986 (ed. with PDP Research grp).
[18] A. Mendiburu, J. Miguel-Alonso, J.A. Lozano, M. Ostra, C. Ubide, Parallel and multi-objective EDAs to create multivariate calibration models for quantitative chemical applications, in: ICPP Workshops, IEEE Computer Society, 2005, pp. 596–603.
[19] A. Mendiburu, J.A. Lozano, J. Miguel-Alonso, Parallel implementation of EDAs based on probabilistic graphical models, IEEE Trans. Evolutionary Comput. 9 (4) (2005) 406–423.
[20] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions i. binary parameters, in: Lecture Notes in Computer Science, vol. 1411, Parallel Problem Solving from Nature—PPSN IV, 1996, pp. 178–187.
[21] M. Pelikan, D.E. Goldberg, F. Lobo, A survey of optimization by building and using probabilistic models, Comput. Optim. Appl. 21 (1) (2002) 5–20.
[22] R Development Core Team, R: a language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0 (2004).
[23] F. Rosenblatt, Principles of Neurodynamics, Spartan Books, New York, 1962.
[24] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by backpropagating errors, Nature 323 (1986) 533–536.
[25] G. Schwarz, Estimating the dimension of a model, Ann. Statist. 7 (2) (1978) 461–464.
[26] N. Villegas, Desenvolupament de procediments cinètics per l'anàlisi de multicomponents, Ph.D. Thesis, Universidad Autónoma de Barcelona, 2003.
[27] R. Wehrens, B.-H. Mevik, PLS: Partial Least Squares Regression (PLSR) and Principal Component Regression (PCR), R package version 1.0-2 (2005).
[28] S. Wold, M. Sjöström, L. Eriksson, PLS-regression: a basic tool of chemometrics, Chemometrics Intelligent Laboratory Syst. 58 (2001) 109–130.

A. Mendiburu received the B.S. degree in computer science from the University of the Basque Country, Gipuzkoa, Spain, in 1995. Since 1999 he has been a Lecturer at the Department of Computer Architecture and Technology, University of the Basque Country. His main research areas are evolutionary computation, probabilistic graphical models, and parallel computing.

Jose Miguel-Alonso received the Ph.D. degree in computer science from the University of the Basque Country, Gipuzkoa, Spain, in 1996. He is a Full Professor at the Department of Computer Architecture and Technology, University of the Basque Country. His research interests include parallel computing, networks for parallel systems, and network (cluster, grid) computing.

Jose A. Lozano received the B.S. degrees in mathematics and computer science and the Ph.D. degree from the University of the Basque Country, Gipuzkoa, Spain, in 1991, 1992, and 1998, respectively. Since 1999, he has been an Associate Professor of Computer Science at the University of the Basque Country. He has edited three books and has published over 20 refereed journal papers. His main research interests are: evolutionary computation, machine learning, probabilistic graphical models, and bioinformatics.
Prof. Lozano is an Associate Editor of the IEEE Transactions on Evolutionary Computation and was a Co-Chair of the 8th International Conference on Parallel Problem Solving from Nature (PPSN'04).

Miren Ostra received the Ph.D. degree in chemistry from the University of the Basque Country, Gipuzkoa, Spain, in 2002. Since 2002 she has been an Associate Proffesor of chemistry sciences at the University of the Basque Country. Her research interests include kinetic methods of analysis, multivariate calibration applied to multicomponent determinations, and process analysis for the control of electroplating baths.

Carlos Ubide received the Ph.D. degree in chemistry from the University of the Basque Country, Gipuzkoa, Spain, in 1985. Since 1979 he has been an Associate Professor of analytical chemistry at the University of the Basque Country. His research interests encompass the kinetic aspects of analytical chemistry, the application of chemometrics to the determination of chemical species (individual or multicomponent), the application of chemometrics to the control of metal electrodeposition baths and keeping in the shade.