Learning Factorizations in Estimation of Distribution Algorithms Using Affinity Propagation

Roberto Santana roberto.santana@upm.es
Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegacedo, 28660, Boadilla del Monte, Madrid, Spain

Pedro Larrañaga pedro.larranaga@fi.upm.es
Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Campus de Montegacedo, 28660, Boadilla del Monte, Madrid, Spain

José A. Lozano ja.lozano@ehu.es
Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 20018, San Sebastian, Spain

Abstract

Estimation of distribution algorithms (EDAs) that use marginal product model factorizations have been widely applied to a broad range of mainly binary optimization problems. In this paper, we introduce the affinity propagation EDA (AffEDA) which learns a marginal product model by clustering a matrix of mutual information learned from the data using a very efficient message-passing algorithm known as affinity propagation. The introduced algorithm is tested on a set of binary and nonbinary decomposable functions and using a hard combinatorial class of problem known as the HP protein model. The results show that the algorithm is a very efficient alternative to other EDAs that use marginal product model factorizations such as the extended compact genetic algorithm (ECGA) and improves the quality of the results achieved by ECGA when the cardinality of the variables is increased.

Keywords

Estimation of distribution algorithms, affinity propagation, factorized distribution algorithms, message-passing algorithms, marginal product models, factorizations, extended compact genetic algorithm.

1 Introduction

Estimation of distribution algorithms (EDAs; Larrañaga and Lozano, 2002; Mühlenbein and Paaß, 1996; Pelikan et al., 2002) are a class of evolutionary algorithms characterized by the use of probability models instead of genetic operators. In EDAs, machine learning methods are used to extract relevant features of the search space. The mined information is represented using a probabilistic model which is later employed to generate new points. In this way, modeling is used to orient the search to promising areas of the search space.

EDAs mainly differ in the class of probabilistic models used and the methods applied to learn and sample these models. In general, simple models are easy to learn and sample, but their capacity to represent complex interactions is limited. On the other hand, more complex models can represent higher-order interactions between the variables but, in most of the cases, a substantial amount of computational time and memory are required to learn these models. Some EDAs use learning algorithms that are, to some extent, able to adapt the complexity of the model to the characteristics of the data. Examples of EDAs that use these types of models are those based on Bayesian networks (Etxeberria and Larrañaga, 1999; Mühlenbein and Mahnig, 2001; Pelikan, 2005).

One of the ways of adapting the structure of the model to the characteristics of the data is by means of scoring metrics (Heckerman et al., 1995) which evaluate the accuracy of the probabilistic model approximations and allow for search in the model space. Nevertheless, the use of metrics has a cost in terms of time. During the search for the models, and in order to compute the metrics, high order probabilistic tables have to be stored. Other EDAs that do not employ Bayesian networks but apply scoring metrics (e.g., the extended compact genetic algorithm, ECGA, see Harik, 1999; Harik et al., 2006; for an EDA that uses the minimum description length metric, MDL, see Rissanen, 1978) suffer from the same drawback.

In order to reduce the complexity of the model learning step, or to improve the accuracy of the approximation of higher-order marginal probabilities (particularly when the population size is not sufficient for accurate estimates), there are EDAs that use lower-order statistics to approximate higher-order interactions. Among them are: PADA (Soto, 2003; Ochoa et al., 2003), DSMGA (Yu et al., 2003), MN-FDA (Santana, 2003), MN-EDA (Santana, 2005), and EDNA (Gámez et al., 2007). These algorithms are different in the class of probabilistic models they learn, the step at which the approximation is applied, and the techniques used to learn the approximations. The approach presented in this paper focuses on learning marginal product models (Harik, 1999) and uses a new learning algorithm based on a type of structural learning procedure completely different to those previously applied in EDAs. In comparison to previous EDAs, our proposal introduces a more robust learning algorithm. It also avoids the use of highly costly sampling methods such as those used by MN-EDA and EDNA.

We present an EDA whose model is constructed using mutual information between pairs of variables and an affinity propagation algorithm (Frey and Dueck, 2006, 2007). Only univariate and bivariate marginals are computed in the structural learning step. The rationale of the learning algorithm is to efficiently cluster the matrix of mutual information in order to obtain groups of mutually interacting variables. From these groups, the factors of the factorization are formed.

Affinity propagation is a recently introduced clustering method which takes as input a set of measures of similarity between pairs of data points and outputs a set of clusters of the points. For each cluster, a typical or representative member, which is called the exemplar, is identified. The method has been praised (Mézard, 2007) because of its ability to efficiently and quickly handle very large problems. We introduce affinity propagation in EDAs as an efficient way to find the problem structure from the mutual information matrix. Our contribution can be set in the trend of a number of proposals (Ochoa et al., 2003; Höns, 2006; Höns et al., 2007; Mendiburu et al., 2007) that combine the classical learning and sampling methods used by EDAs with different classes of message-passing and inference methods (Pearl, 1988; Yedidia et al., 2005). This research trend leads to a new generation of EDAs that integrate different types of machine learning strategies. Our work is also related to recent efforts (Pelikan et al., 2008) to decrease the asymptotic complexity of model building in EDAs. We show that the use of the affinity propagation algorithm can dramatically reduce the time complexity of learning marginal product models. Affinity propagation is also very suitable for parallelization. Therefore, further efficiency enhancements to our algorithm are still possible.

The paper is organized as follows: In the next section, we introduce the notation and give a brief overview of EDAs. Section 3 analyzes the question of learning marginal product models from short order marginal probabilities. In Section 4, message-passing algorithms are briefly reviewed and the general scheme of the affinity propagation algorithm is presented. Section 5 introduces an EDA that learns a marginal product model using affinity propagation. Comparison with other related EDAs, analysis of its computational cost, and details about the algorithm's implementation are presented. Section 6 presents the experiments made for a number of functions and a problem defined on a simplified protein model. The conclusions of our paper and some trends for future research are given in Section 7.

2 Estimation of Distribution Algorithms and Factorizations

Let X be a discrete random variable. A value of X is denoted x. $\mathbf{X} = (X_1, \ldots, X_n)$ will denote a vector of random variables. We will use $\mathbf{x} = (x_1, \ldots, x_n)$ to denote an assignment to the variables. S will denote a set of indices in $\{1, \ldots, n\}$, and $\mathbf{X}_S$ (respectively, $\mathbf{x}_S$) a subset of the variables of $\mathbf{X}$ (respectively, a subset of values of $\mathbf{x}$) determined by the indices in S.

The joint probability mass function of $\mathbf{x}$ is represented as $p(\mathbf{X} = \mathbf{x})$ or $p(\mathbf{x})$. $p(\mathbf{x}_S)$ will denote the marginal probability distribution for $\mathbf{X}_S$. We use $p(X_i = x_i \mid X_j = x_j)$ or, in a simplified form, $p(x_i \mid x_j)$, to denote the conditional probability distribution of X_i given $X_j = x_j$.

2.1 Probabilistic Modeling in EDAs

One of the goals of probabilistic modeling in EDAs is to obtain a condensed, accurate model of the selected points. This representation is usually expressed by means of a factorization which is constructed from a graphical model. In simple terms, a factorization of a distribution $p(\mathbf{x})$ is a generalized product of marginal probability distributions $p(\mathbf{X}_s)$ each of which is called a factor.

Regarding the type of independence relationships between the variables in a factorization, a marginal product factorization only represents marginal independence relationships between sets of variables. In these factorizations, the variables are divided into disjoint factors. In contrast, those that consider conditional factors can represent marginal and conditional independence between the variables. In these factorizations, one variable can belong to several different factors.

In this paper, we will focus on EDAs that use marginal product models (MPMs). Two well known examples are the univariate marginal distribution algorithm (UMDA; Mühlenbein and Paaß, 1996) and the extended compact GA (ECGA; Harik, 1999; Harik et al., 2006). We first devote some time to present ECGA and the learning procedures it employs.

2.2 ECGA

ECGA uses a factorization of the probability where variables are separated in non-overlapping factors. These factors are found by optimizing the minimum description length (MDL) metric of the model representing the selected solutions. In the model, each factor is assumed to be independent of the rest. The structure of ECGA is similar to most of EDAs. The algorithm finds an MPM by minimization of a MDL metric using a greedy search. Tournament selection is generally applied with the algorithm.

Let χ be the alphabet cardinality of the variables,[1] m the number of dependency sets of the model (factors), k_i the number of variables in the definition set S_i, and N_{ij} the number of solutions in the current population that contain the instantiation $j \in \{1, \ldots, |\chi^{k_i}|\}$ for the definition set S_i.

In ECGA (Harik, 1999; Harik et al., 2006), learning the MPM in every generation is approached as a constrained optimization problem:

$$\text{Minimize} \quad C_m + C_p \tag{1}$$

$$\text{Subject to} \quad |\chi^{k_i}| \leq N \ \forall i \in \{1, \ldots, m\} \tag{2}$$

where C_m represents the model complexity and is given by

$$C_m = \log_2(N+1) \sum_{i=1}^{m} (|\chi^{k_i}| - 1) \tag{3}$$

and C_p is the compressed population complexity which represents the cost of using a simple model as opposed to a complex one and is evaluated as

$$C_p = \sum_{i=1}^{m} \sum_{j=1}^{|\chi^{k_i}|} N_{ij} \log_2 \left(\frac{N}{N_{ij}} \right). \tag{4}$$

C_m is obtained by considering that each of the frequency counts is of size $\log_2 N$ and each subset of k_i variables requires $|\chi^{k_i}| - 1$ frequency counts to completely determine its marginal distribution. C_p is the entropy of the distribution, that is, the average number of bits it takes to represent the data. In Harik (1997) and Harik et al. (2006) a combined complexity number is obtained by equally weighting C_m and C_p.

The greedy search heuristic used by ECGA is shown in Algorithm 1. It starts with a model where all the variables are assumed to be independent and sequentially merges subsets until the MDL metric no longer improves. At every step, all possible merges are inspected. The computational cost of the merging depends on the size of the marginal tables needed to compute the factors and the number of individuals in the selected population.

Note that by minimizing the combined measure of complexity, an implicit clustering of the variables is performed, C_m manifestly controls the number of clusters while C_p controls the distortion (the sum of mutual information inside the clusters).

Once the factors have been learned, the parameters are learned using the maximum likelihood estimation and new solutions are generated by independently sampling each factor.

[1] Although to simplify the analysis we assume in this section the same cardinality for all the variables, the analysis can be extended to variables with different cardinality.

Algorithm 1: ECGA structural learning algorithm

```
1  Define each factor as composed of a single variable
2  do {
3      For each pair of factors
4          Merge the two factors
5          Evaluate the MDL metric (Equation (1)) of the current model
6          Undo merging
7      Select the merging action that improved the MDL the most
8  } until No further improvement in the metric is achieved
```

3 Learning MPMs from Short Order Marginal Probabilities

There are two main approaches for learning probabilistic models from data (Heckerman et al., 1995; Neapolitan, 2003), learning based on detecting conditional independencies by means of independence tests, and score+search algorithms. Algorithms that combine these approaches have also been proposed (Soto et al., 1999).

In general, the use of score+search methods implies the intensive computation of high order marginal distributions needed to assess the accuracy of each model inspected during the search. The computational cost of this step can be reduced by imposing a constraint to the size of the tables during the construction of the model, but as a result, the model obtained will be constrained in terms of the size of its marginal tables. One open question is whether higher-order models can be learned by limiting the information used during the learning step to only the low order probabilistic tables.

Our final objective is to introduce a structural learning algorithm more efficient than the methods currently used to learn MPMs in EDAs. An approach to this question is to obtain higher-order models by grouping or clustering small order dependence sets in factors that comprise highly interacting sets of variables. This bottom-up approach will begin by the identification of pairwise interactions between variables and will combine them but without the need to compute higher-order tables to evaluate the quality of the model.

3.1 Clustering of the Mutual Information as an Optimization Problem

One of the foundations of the model building method introduced in this paper lies in the fact that the problem of finding an accurate partition of the mutual information matrix is in fact a clustering problem. The kernel of our proposal is the application of a very efficient clustering method which by simultaneously considering all data points as candidate centers of the clusters is able to avoid many of the poor solutions caused by unlucky initializations of commonly used clustering algorithms (e.g., k-means) based on random sampling (Frey and Dueck, 2007).

In order to introduce the method, we first show how the clustering problem can be posed as an optimization problem.

3.1.1 Clustering of a Similarity Matrix

In the general clustering problem, $\mathbf{y}^1, \ldots, \mathbf{y}^Q$ will represent the Q ordered data points. $\mathbf{c} = (c^1, \ldots, c^Q)$ will represent a vector of Q hidden labels corresponding to the Q data points. Each value c^i $(1 \leq c^i \leq Q)$ indicates the cluster each data point $\mathbf{y}^i$ belongs to.

There is an additional constraint, if $c^i = j$ then $\mathbf{y}^i$ belongs to the cluster where $\mathbf{y}^j$ is. Therefore, $c^j = j$ should be fulfilled, implying that $\mathbf{y}^j$ belongs to the cluster it serves to define.

The similarity between two data points is represented by $s(\mathbf{y}^i, \mathbf{y}^j)$. In general, the similarity measure does not have to be symmetric, but for our analysis we will assume that symmetry holds. We want to maximize the similarity between points that belong to the same cluster. The function to be maximized is

$$\mathcal{F}(\mathbf{c}) = \frac{2}{Q'} \sum_{l=1}^{Q'} \frac{1}{(|D_l|(|D_l|-1))} \sum_{i<j, c^i=l, c^j=l} s(\mathbf{y}^i, \mathbf{y}^j) \tag{5}$$

where $|D_l|$ is the number of points in the cluster D_l, $|D_l| > 0$ and $\sum_{l=1}^{Q'} |D_l| = Q$. The number of clusters Q' is a variable of the problem. Since the clusters do not overlap, its maximal number is Q. The value of Q' is also related to the average size of the clusters. A higher value of Q' implies that clusters are smaller on average.

Notice that Equation (5) computes the average sum of similarities between pairs of points that belong to the same cluster (i.e., $c^i = l, c^j = l$). This average is obtained by dividing by $\frac{(|D_l|(|D_l|-1))}{2}$ which is the number of pairs in each cluster D_l. By dividing by the number of clusters Q', the average of the clusters similarity is obtained. Obviously, $\mathcal{F}(\mathbf{c})$ will be higher as pairs belonging to the same cluster are more similar.

In our particular instantiation of the clustering problem, we are interested in clustering variables (i.e., $Q = n$ and $\mathbf{y}^i = X_i$), and we want to identify clusters of variables with a high value of pairwise mutual information. Therefore, $s(\mathbf{y}^i, \mathbf{y}^j) = I(X_i, X_j)$ where $i, j \in \{1, \ldots, n\}$. The mutual information is a symmetric similarity measure. This means that the clustering problem will involve a smaller number of parameters than in the general case where $s(\mathbf{y}^i, \mathbf{y}^j)$ is not necessarily equal to $s(\mathbf{y}^j, \mathbf{y}^i)$.

Equation (5) measures the sum of the average distances between pairs of points in each cluster. The problem representation is flexible, allowing for a representation of clusterings with a different number of clusters. However, it has two main drawbacks. Not every assignment of c^i is valid. It has to be satisfied that if $c^i = j$, implying that $\mathbf{y}^i$ belongs to the cluster where $\mathbf{y}^j$ is, then $c^j = j$. The other drawback is that the representation is highly redundant.

Accomplishing the maximization of Equation (5) is difficult because we have to determine at the same time the number of clusters Q' and the cluster membership of each point. Therefore, the problem is more complex than the k-partitioning problem (Babel et al., 1998) for which the number of clusters is known in advance. We deal with the clustering problem using a relaxation approach, that is, the maximization of the function is addressed in two steps. Firstly, a (possibly suboptimal) solution of the clustering problem is found by minimizing a related function. This step provides a bound to the initial number of clusters whose size is constrained for efficiency considerations. Secondly, a local optimization method is applied to the suboptimal solution to find a better (possibly optimal) clustering solution. We expect this approach to be faster than greedy factorization learning, which is perhaps the search method most frequently used in EDAs.

3.1.2 Finding Initial Clusters and Clusters Exemplars

Instead of approaching the clustering of the mutual information in a straightforward way, we will first find a solution that maximizes the similarity of each point of the cluster to a distinguished point exemplar of the cluster. The rationale is that we can start from finding groups of points that have a strong similarity with another single point (the exemplar) and then verify whether these points have a high average similarity between them.

In this case, the clustering problem can be defined in terms of finding the maxima of a function $E(\mathbf{c}) = \sum_{i=1}^{n} s(\mathbf{y}^i, \mathbf{y}^{c^i})$ that depends on a set of n hidden labels $c^1, \ldots, c^n$, corresponding to the n data points. Each label indicates the exemplar to which the point belongs and $s(\mathbf{y}^i, \mathbf{y}^{c^i})$ is the similarity of data point $\mathbf{y}^i$ to its exemplar. An exemplar should satisfy that $c^j = j$. In order to avoid invalid configurations, constraints have to be imposed on the solutions. The problem is then posed as the problem of maximizing the net similarity $\mathcal{S}$ (Frey and Dueck, 2007)

$$\mathcal{S}(\mathbf{c}) = E(\mathbf{c}) + \sum_{l=1}^{n} \delta_l(\mathbf{c}) = \sum_{i=1}^{n} s(\mathbf{y}^i, \mathbf{y}^{c^i}) + \sum_{l=1}^{n} \delta_l(\mathbf{c}) \tag{6}$$

where

$$\delta_l(\mathbf{c}) = \begin{cases} -\infty & \text{if } c^l \neq l \text{ and } \exists i : c^i = l \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

In comparison with Equation (5), the function represented by Equation (6) automatically specifies the membership of each point to a cluster. The constraint imposes that there will be one exemplar for each cluster. Equations (5) and (6) are very different because they represent related but different problems.

The convenience of using Equation (6) lies in that we can apply an efficient optimization method based on an iterative use of a message-passing algorithm.

4 Message-Passing Algorithms and Affinity Propagation

Message passing algorithms can be seen as a general problem decomposition approach in which the construction of a solution for a given problem is organized as a distributed process where different components interact, exchanging messages between them. The messages contain (possibly contradictory) local information about the problem and the components are connected according to some structural relationship determined by the problem. The process of exchanging the messages is continued until an agreement between all the local components is reached. The distributed way in which the search for a solution is organized, and the implicit problem decomposition contained in the form of connections between the components, contribute to the efficiency of these methods and make them appropriate for model building in EDAs.

There are several variants of message-passing algorithms. They include belief propagation (BP) algorithms (Pearl, 1988; Yedidia et al., 2005), survey and warning propagation algorithms (Braunstein et al., 2005; Hartmann and Weigt, 2005; Mézard et al., 2002) and affinity propagation algorithms (Frey and Dueck, 2006, 2007; Leone et al., 2007).

4.1 Affinity Propagation

The explanation of affinity propagation presented in this section has been done following Frey and Dueck (2007), where more details and examples of the application of the algorithm can be found.

Affinity propagation is a clustering algorithm that, given a set of points and a set of similarity measures between the points, finds clusters of similar points, and for each cluster gives a representative example or *exemplar*.

Affinity propagation has several advantages over related techniques. Methods such as k-centers clustering and k-means clustering store a relatively small set of estimated cluster centers at each step. These techniques can be improved by using methods that begin with a large number of clusters and then prune them, but they still rely on random sampling and make hard pruning decisions that cannot be recovered from. In contrast, by simultaneously considering all data points as candidate centers and gradually identifying clusters, affinity propagation is able to avoid many of the poor solutions caused by unlucky initializations and hard decisions (Frey and Dueck, 2007).

A characteristic that makes affinity propagation different from other clustering algorithms is that the points directly exchange information between them regarding the suitability of each point to serve as an exemplar for a subset of other points.

The algorithm takes as input a matrix of similarity measures between each pair of points $s(\mathbf{y}^i, \mathbf{y}^k)$. Instead of requiring that the number of clusters be predetermined, affinity propagation takes as input a real number $s(\mathbf{y}^k, \mathbf{y}^k)$ for each data point $\mathbf{y}^k$. These values, which are called *preferences*, are a measure of how likely each point is to be chosen as the exemplar. The algorithm works by exchanging messages between the points until a stop condition, which reflects an agreement between all the points with respect to the current assignment of the exemplars, is satisfied. These messages can be seen as the way the points share local information in the gradual determination of the exemplars.

There are two types of messages to be exchanged between data points. The *responsibility* $r(i, k)$, sent from data point $\mathbf{y}^i$ to candidate exemplar point $\mathbf{y}^k$, reflects the accumulated evidence for how well-suited point $\mathbf{y}^k$ is to serve as the exemplar for point $\mathbf{y}^i$, taking into account other potential exemplars for point $\mathbf{y}^i$. The *availability* $a(i, k)$ sent from candidate exemplar point $\mathbf{y}^k$ to point $\mathbf{y}^i$ reflects the accumulated evidence for how appropriate it would be for point $\mathbf{y}^i$ to choose point $\mathbf{y}^k$ as its exemplar, taking into account the support from other points that point $\mathbf{y}^k$ should be an exemplar.

The availabilities are initialized to zero: $a(i, k) = 0$. Then, the responsibilities are computed using the rule

$$r(i, k) \leftarrow s(\mathbf{y}^i, \mathbf{y}^k) - max_{k'|k' \neq k}\{a(i, k') + s(\mathbf{y}^i, \mathbf{y}^{k'})\} \tag{8}$$

The responsibility update shown in Equation (8) allows all the candidate exemplars to compete for ownership of a data point. Evidence about whether each candidate exemplar would make a good exemplar is obtained from the application of the following availability update:

$$a(i, k) \leftarrow min \left\{ 0, r(k, k) + \sum_{i'|i' \notin \{i,k\}} max\{0, r(i', k)\} \right\} \tag{9}$$

In the availability update shown in Equation (9), only the positive portions of incoming responsibilities are added, because it is only necessary for a good exemplar to explain some data points (positive responsibilities), regardless of how poorly it explains points with negative responsibilities. In order to limit the influence of incoming positive responsibilities, the total sum is thresholded so that it cannot go above zero.

The *self-availability* $a(k, k)$ is updated differently

$$a(k, k) = \sum_{i' | i' \neq k} max\{0, r(i', k)\} \qquad (10)$$

For a point $\mathbf{y}^i$, the value of k that maximizes $a(i, k) + r(i, k)$ either identifies point $\mathbf{y}^i$ as an exemplar if $k = i$ $(c^i = i)$, or identifies the data point that is the exemplar for point $\mathbf{y}^i$.

Update rules described by Equations (8), (9), and (10) require only local computations. Additionally, messages are exchanged only between pairs of points with known similarities. The message-passing procedure may be terminated after a fixed number of interactions, when changes in the messages fall below a threshold, or after the local decisions stay constant for some number of iterations.

Similar to other propagation methods, *damping* should be used to confront numerical oscillations that arise in some circumstances. This technique consists of setting each message to λ times its value from the previous iteration plus $1 - \lambda$ times its prescribed updated value $(0 < \lambda < 1)$. The pseudocode of the affinity propagation algorithm is shown in Algorithm 2.

Algorithm 2: Affinity propagation

```
1  Initialize availabilities $a(i, k)$ to zero $\forall i, k$
2  **do** {
3      Update, using Equation (8), all the responsibilities given the availabilities
4      Update, using Equation (9), all the availabilities given the responsibilities
5      Combine availabilities and responsibilities to obtain the exemplar decisions
6  } **until** Termination criterion is met
```

A limitation of affinity propagation and other message passing algorithms is that the convergence of the method is not guaranteed. There are different alternatives to deal with these situations. The parameters of the algorithm can be modified or some random noise can be added to the data.

The maximization of Equation (6) is computationally intractable. However, the update rules defined for affinity propagation correspond to fixed-point recursions for minimizing a Bethe-free approximation (Yedidia et al., 2005) of this energy function. In practice, the solution obtained by affinity propagation is very good. It corresponds to a local optimum of the approximation. Nevertheless, the solution of Equation (5) is a cluster such that all the members are highly similar to one of the members (the exemplar) and it might be the case that some points are highly similar to the exemplar but dissimilar to the other members of the cluster. In this case, the satisfaction of Equation (5) can be enforced by moving these points to clusters with more similar members.

Our strategy is to use the output given by affinity propagation as an initial approximate solution to find the needed partitioning of the mutual information. We apply a very simple local optimizer that reassigns the points starting from the initial clusters found by affinity. Its pseudocode is shown in Algorithm 3.

Algorithm 3: Partitioning improvement algorithm

```
1  $iter = 0$;
2  While $iter < PImaxiter$
3      For each point $\mathbf{y}^i$
4          For each current cluster $j$
5              If $j$ contains at least two points, find average similarity $s^j_{\mathbf{y}^i}$ of $\mathbf{y}^i$ to
               points in $j$. Else $s^j_{\mathbf{y}^i} = -\infty$
6          Find cluster $j_{max}$ with maximum positive average similarity of $\mathbf{y}^i$ to points
           in $j_{max}$
7      For each point $\mathbf{y}^i$
8          Assign $\mathbf{y}^i$ to cluster $j_{max}$
9  $iter = iter + 1$
```

Algorithm 3 receives as a parameter the number of iterations that the reassignment procedure should be applied. It finds for each of the points, the most similar cluster (in terms of the average similarity to the cluster's members). A point that is more similar to a cluster other than the one it belongs to will be reassigned. The number of iterations is set to be very small. In our experiments, $PImaxiter = 3$.

5 An EDA Based on Affinity Propagation

In this section, we present an EDA that learns an MPM using affinity propagation and which takes as the similarity measure the matrix of mutual information between the variables. To our knowledge, this is the first EDA that uses a message-passing algorithm for structural learning. The rationale of the method is to group the variables in nonoverlapping sets, where strong interacting variables are contained in the same set. These sets are then used to determine the factors of the probability factorization.

In addition, and since a feasible EDA factorization requires that the maximum size of the factors should not be above a given threshold, we have conceived a modification of Algorithm 2 that allows the algorithm to find a clustering where clusters satisfy a constraint on the size of the factors. Previous implementations of affinity propagation do not take into account this type of constraint on the cluster size.

The constraint value is automatically derived from the data. The idea is that the current number of data points could provide an acceptable estimate for the marginal probability of the largest factor. Therefore, the maximum size of the clusters is determined as $\delta = \log_{r_{MAX}} N$ where r_{MAX} is the highest cardinality among all the variables.

The pseudocode of the modified affinity propagation algorithm is described in Algorithm 4. The algorithm receives as inputs an empty list of factors L, a similarity matrix MI of a set of variables V being clustered and the maximum allowed size to the clusters, δ. The algorithm updates the list of factors L.

Since the algorithm is recursive, a maximum number of recursive calls $depth$ is passed as a parameter. The variable $ncalls$ is an auxiliary variable of the program which tracks the actual number of recursive calls. In the current version of AffEDA, and after making a number of preliminary experiments, the AffEDA parameters have been set as $depth = 30$, $AFmaxiter = 1{,}000$ and $PM = 3$. However, the quality of the models learned could be improved by manipulating these parameters.

Algorithm 4: Affinity propagation with constrained clusters

```
1  Find the connected components of matrix $MI$
2  Add to $L$ all the connected components with size less than or equal to 3
3  Randomly split all homogeneous connected components in clusters of size $\delta$
   and add them to $L$
4  For each remaining connected component $i$
5      Construct $MI_i$ by considering MI values between points in the connected
       component $i$
6      $ncalls = 0$
7      $Clustered = 0$
8      While $ncalls < depth$ and $Clustered = 0$
9          Call Algorithm 2 using variables in component $V_i$ and $MI_i$
10         If Algorithm 2 converged, call Algorithm 3 with the output clusters
11         If Algorithm 2 converged and the number of clusters given by Algorithm 3
           is at least two
12             $Clustered = 1$
13         Else
14             $ncalls = ncalls + 1$
15         If $Clustered = 0$
16             Randomly split the variables in the connected component $i$ in clusters
               of size $\delta$ and add to $L$ all the splitted clusters
17         If $Clustered = 1$ and at least one of the clusters found has size less than or
           equal to $\delta$
18             Add to $L$ all clusters with size less than or equal to $\delta$
19             Add to $V'$ variables in clusters with size above $\delta$
20             If $V' \neq \emptyset$ call Algorithm 4 with variables in $V'$ and using $MI_{V'}$
21         Else if $Clustered = 1$ and none of the clusters found has size less than or
           equal to $\delta$
22             For each of the clusters $c_j$, call Algorithm 4 with the variables in $V_{c_j}$
               and using $MI_{c_j}$
```

Algorithm 4 starts by finding the connected components of the similarity matrix. Obviously, variables that are not connected in the matrix (the similarity between them is zero) will not belong to the same cluster and therefore we can conduct the clustering process in each of these components separately. At Step 2, all connected components with three or fewer variables are added to the list of factors. At Step 3, the algorithm determines those connected components for which the mutual information between their variables is the same (homogeneous components). These clusters cannot be further divided using the mutual information as a criterion, therefore, at Step 3 they are randomly divided in groups of size δ and added to L. Homogeneous components may be common when the diversity in the population is lost and many copies of few individuals dominate the population.

For each of the remaining components (nonhomogeneous, with size above 3) V_i, the algorithm computes the reduced matrix of mutual information MI_i and calls Algorithm 2 a maximum of $depth$ times varying the parameters used for affinity propagation (i.e., the damping factor is increased). If the maximum number of calls has been reached and the algorithm has not converged or it has converged to a single cluster, the set of variables is randomly split in clusters of δ variables.

If the algorithm converged and at least one of the clusters found has size less than or equal to δ, the feasible clusters are inserted in L and the rest of the variables, where they exist, are grouped together to recursively call Algorithm 4 again. If the algorithm converged and none of the clusters found has size less than or equal to δ, then Algorithm 4 is called for each of the (unfeasible) clusters found.

The algorithm is guaranteed to terminate either because all feasible clusters have been added to L or because random splitting has been invoked for homogeneous connected components or irreducible clusters.

Algorithm 4 is the learning component of our AffEDA, whose pseudocode is shown in Algorithm 5. AffEDA may use different types of selection methods.

Algorithm 5: AffEDA

```
1  $D_0 \leftarrow$ Generate $M$ individuals randomly
2  $t = 1$
3  **do** {
4      $D^s_{t-1} \leftarrow$ Select $N \leq M$ individuals from $D_{t\text{-}1}$ according to a selection method
5      Compute the mutual information between every pair of variables
6      Apply Algorithm 4 to obtain an MPM of constrained size
7      $D_t \leftarrow$ Sample $M$ individuals according to the distribution $p(\mathbf{x}, t) = \prod_{i=1}^{m} p^s_{S_i}(x_{S_i}, t-1)$
8      $t \Leftarrow t + 1$
9  } **until** A termination criterion is met
```

5.1 Analysis of the Computational Complexity

In this section, we analyze the computational cost of AffEDA. This cost is very difficult to estimate due to the recursive nature of Algorithm 4 used by AffEDA.

We use $r_{MAX} = max_{i \in 1,\dots,n} |X_i|$ to represent the highest cardinality among the variables, where m is the number of factors in the MPM factorization, δ is the maximum size of the factors, and r^{δ}_{MAX} is a bound on the size of the probability table associated with the biggest factor.

The initialization step consists of randomly initializing all the solutions in the first population. It has complexity $O(nM)$.

The computational cost of the evaluation step is problem dependent. It will also depend on the function implementation. Let $cost_f$ be the running time associated with the evaluation of function f, then the running time complexity of this step is $O(Mcost_f)$.

The complexity of the selection step depends on the selection method used. For tournament selection, the complexity is $O(\tau M)$, where τ is the size of the tournament.

The cost of the model learning step can be further divided into the cost of structural and parametric learning.

The structural learning step includes the computation of the matrix of mutual information and the application of Algorithm 4.

- The calculation of the mutual information has complexity $O(N\, n^2 r^2_{MAX})$.

- The efficient implementation of the affinity propagation algorithm has a quadratic complexity in the number of data points, which in our case corresponds to the number of variables and also depends on the number of interactions. Notice however that the identification of connected and homogeneous components can reduce this estimate since there are fewer variables involved in the clustering of each component. Where $AFmaxiter$ is the maximum number of iterations, an upper bound for the complexity of Algorithm 2 is $O(n^2 AFmaxiter)$.
- The cost of the partitioning improvement algorithm (Algorithm 3) used by Algorithm 4 depends on the maximal number of iterations ($PImaxiter$) and the number of clusters found. Assuming that the number of clusters is αn ($\alpha \in \{\frac{1}{n}, \ldots, \frac{n}{n}\}$), the complexity of the algorithm is $O(n^2 PImaxiter)$.
- As previously stated, the complexity of Algorithm 4 is very difficult to estimate since the time spent depends on the structure of the matrix of mutual information and the recursive calls. We roughly estimate its cost as $O(n^2 depth(PImaxiter + AFmaxiter))$ where $depth$ is a maximum number of recursive calls.

The total cost of the structural learning step can be estimated as $O(N n^2 r_{MAX}^2 + n^2 depth(PImaxiter + AFmaxiter))$.

The cost of parametric learning is the cost of computing the marginal probabilities used by AffEDA, that is, the m marginal probability tables from the selected population, and it has complexity $O(m(N\,\delta + r_{MAX}^{\delta}))$.

The cost of the sampling step is the cost of sampling M individuals from the MPM. It has a complexity order $O(mMr_{MAX}^{\delta})$.

The total computational cost of AffEDA is $O(G(N n^2 r_{MAX}^2 + n^2 depth(PImaxiter + AFmaxiter) + m(N + \delta + r_{MAX}^{\delta}) + mMr_{MAX}^{delta} + Mcost_f))$, where the population size M and the number of generations G change according to the problem difficulty.

5.2 Comparison of AffEDA with Other Related EDAs

In this section, we want to emphasize some differences between AffEDA and other EDAs that learn the structure of the model from data, in particular EDAs that obtain higher-order models by grouping or clustering small order dependence sets.

In EDAs, we identify a number of ways in which higher-order models have been learned by grouping or clustering small order dependence sets in factors that comprise highly interacting sets of variables. Among them:

- Use of iterative proportional fitting (IPF; Deming and Stephan, 1940) which is a method to compute higher order marginal approximations. IPF allows the algorithm to find a maximum entropy distribution given a set of constraints (in this case the constraints correspond to the known marginal distributions). In Ochoa et al. (2003), a junction-tree based implementation of IPF is used to approximate higher-order marginals in the polytree approximation distribution algorithm (PADA; Soto et al., 1999). Since PADA only uses one and two conditional marginal distributions to learn its graphical model (polytrees), the approximation of higher-order probability marginals is essential. The numerical results presented in Ochoa et al. (2003) and Höns (2006) showed remarkable improvements in the version of PADA that added IPF with respect to UMDA and simpler PADA variants.

- Use of a heuristic algorithm (Bron and Kerbosch, 1973) that finds all the maximal cliques of an independence graph to determine the clusters of variables with the highest sum of bivariate interactions (measured using the χ^2 statistics). This approach combines the use of (up to order one) independence tests with the clustering method. In Santana (2003, 2005), it was used in the context of EDAs to respectively learn junction graphs and Kikuchi approximations on Markov networks. When used the algorithm was used to learn Kikuchi approximations, and for the functions considered, the algorithm exhibited similar results to EDAs that use Bayesian network based models.

- Use of a mutual information-based approximation of the network scoring metric in the structural learning step of the estimation of dependency networks algorithm (EDNA; Gámez et al., 2007). Dependency networks (Heckerman et al., 2000) are closely related to Bayesian networks, but similar to Markov networks, they are able to represent cycles between the variables. EDNA uses the mutual information values to look for the parent set of each variable. The idea is to avoid the expensive computation of the full conditional probability of each variable given a set of candidate parents. In Gámez et al. (2008), the use of bivariate statistics to approximate higher-order marginals is extended to the simulation and parametric learning tasks of EDNA.

- Use of a dependency structure matrix (DSM) combined with a MDL technique to cluster pairwise interactions above a given threshold. A DSM is a matrix that contains information about the pairwise interactions between the components of a system. In Yu et al. (2003), this technique is used to extract building block (BB) information and use the information to accomplish BB-wise crossover. The maximal number of clusters and a threshold on the strength of the nonlinearity required to determine whether two variables interact are given as parameters of the method. The search for the optimal clustering is done applying an auxiliary evolutionary strategy (ES). In Yu (2006), the nonlinearity measure of interaction is replaced by the mutual information and a hill-climber replaces ES for the DSM clustering. Two variants of the algorithm, DSMGA+ and DSMGA++ are respectively proposed to deal with hierarchical and overlapping problems.

The PADA versions presented in the literature (Ochoa et al., 2003; Höns, 2006) compute higher-order probabilistic tables only after the structure of the model has been learned. The approximation obtained from the combination of pairwise interactions is merely used for parametric learning. In AffEDA, the approximation is used to construct the model structure, and after it has been learned, higher-order probabilistic tables are learned.

The clustering algorithm used by the MN-FDA and MN-EDA (Santana, 2005) is based on an exhaustive search of all the maximal cliques of a graph. The values of the χ^2 test and a threshold on the strength of the interactions are taken into account for the computation of the clusters (cliques). Finding an adequate threshold can be a difficult task and the algorithm is very sensitive to this value. AffEDA does not use any threshold on the values of the mutual information before the application of the affinity propagation method. This means that all the information contained in the matrix of mutual information is used.

The most recent version of EDNA (Gámez et al., 2008) uses higher-order marginal approximation in the structural and parametric learning steps of the algorithm. In the construction of the structure, the goal is to add to the candidate set of parents of a given variable X_i, the variable X_j with the highest mutual information with respect to X_i, but having a small degree of relation with current parents of X_i. The mutual information between X_i and X_j is compared with the average mutual information between X_j and the variables already included in the parent set. Note that this is a different goal from the one pursued by AffEDA which looks for a cluster with strong mutual information between all the variables. The construction of the dependency structure is done using a greedy algorithm and this is a fundamental difference with the message passing method used by AffEDA.

The most recent version of DSMGA (Yu, 2006) uses clustering of a DSM constructed using the mutual information values. DSMGA also needs to determine a threshold on the values of mutual information previous to the clustering. Mutual information values above the threshold determine that the corresponding pair of variables will potentially be together in a cluster (i.e., $DSM(X_i, X_j) = 1$, otherwise $DSM(X_i, X_j) = 0$). Therefore the matrix clustered in DSMGA is binary while AffEDA does clustering of a real value matrix. Thresholding generally makes the clustering process easier, but also implies that less information is used during the clustering, and that if mutual information slightly changes around the threshold, there could be big changes in the structures of the models learned. In this sense, we consider that our learning algorithm is more robust. While the MDL metric employed by DSMGA does not require the size of the clusters to be constrained, as done by AffEDA, DSMGA needs to constrain the maximum number of clusters. We remark that while the cluster size is clearly constrained by efficiency considerations, it is more difficult to set a bound on the number of clusters. DSMGA uses an evolutionary strategy or hill climbing to find the optimal clustering and AffEDA uses a completely different message passing scheme where the construction of a solution is distributed between the different components of the process.

Another important difference between the previous EDAs and AffEDA is related to the type of sampling method used. AffEDA uses PLS on nonoverlapped factors. PADA uses PLS, but since the polytree model is connected, conditional probabilities are used in PLS. EDNA and MN-EDA are able to learn cyclic probabilistic models and therefore need the application of very costly sampling algorithms (e.g., Gibbs sampling). Since DSMGA is closer to traditional GAs, it employs bitwise intelligent crossover.

5.2.1 Comparison with the ECGA

It is important to compare the computational complexity of AffEDA and ECGA. Since both algorithms only differ in the structural learning algorithm they use, in this section we compare the computational complexity of their structural learning algorithms.

In every iteration, Algorithm 1 used by ECGA may evaluate up to $\frac{n}{2}$ different models. If we assume that cost of evaluating the model depends on the number of the selected individuals, the size of the maximum factor δ and the cardinality of the variables r_{MAX}, the inner loop of Algorithm 1 has complexity $O(N\, r_{MAX}^{\delta} n^3)$. Considering that the number of merges is a function of n, the cost of Algorithm 1 is $O(N r_{MAX}^{\delta} n^3)$. This is higher than the cost estimated for the structural learning algorithm used by AffEDA which is $O(n^2(N\, r_{MAX}^2 + depth(PImaxiter + AFmaxiter)))$, showing that even if the use of higher-order marginals is needed for an accurate model, the number of times and the complexity of the marginals that have to be computed during the construction of the model can be reduced, that is, more efficient ways to learn higher-order marginal distributions can be devised.

Recently Duque et al. (2008), and in parallel to the work presented in this paper, presented preliminary results of an efficiency enhancement in ECGA's model building step. The model building enhancement reduces the complexity of the learning process from $O(n^3)$ to $O(n^2)$. It consists of relaxing the greedy search heuristic by avoiding consideration of every possible merge of pairs during the determination of the blocks. Although this approach may degrade the accuracy of the learned model, it showed no accuracy loss for $f_{deceptive4}$ which was the only function used in the experiments.

5.3 Implementation of the Algorithm

AffEDA implementation has been assembled from available implementations of ECGA and of the affinity propagation algorithm. We have implemented two versions of AffEDA. One is implemented in Matlab and the other one in C++.

The χ-ary ECGA has been conceived for the solution of problems with χ-ary alphabets, that is, problems with discrete representation. The source code[2] in Matlab is an extension of the original binary-coded ECGA. The programs are documented in Sastry and Orriols-Puig (2007). A detail of the implementation relevant for our analysis is that the model learning step (the greedy search heuristic) is an independent program. Therefore, the main changes made to the ECGA software to obtain the AffEDA implementation have been:

- The introduction of a procedure that computes the matrix of mutual information from the selected set.
- The replacement of the greedy search heuristic by our modified affinity propagation algorithm that clusters the mutual information, taking into account the constraints and adding the partitioning improvement algorithm.

The modified version of the affinity propagation algorithm uses the original Matlab implementation of affinity propagation[3] and makes recursive calls to this program. A version of AffEDA that uses truncation selection is available[4] as part of the EDA software MATEDA (Santana et al., 2009).

Also, the C++ source code[5] of the χ-ary ECGA allows the replacement of the original learning component of the MPM by the implemented affinity propagation based learning component. The programs are documented in de la Ossa et al. (2006). The changes made in the C++ source code follow the same rationale as those explained for the Matlab implementation of AffEDA. The modified C++ version of the affinity propagation algorithm uses the original C++ implementation of affinity propagation.[6]

6 Experiments

In this section we evaluate the behavior of AffEDA, comparing its results with those achieved by ECGA. We analyze the quality of the solutions and the time complexity and

[2]http://www.illigal.uiuc.edu/pub/src/ECGA/eCGAmatlab.zip

[3]http://www.psi.toronto.edu/affinitypropagation/apcluster.m

[4]http://www.sc.ehu.es/ccwbayes/members/rsantana/software/matlab/MATEDA.html

[5]http://www.illigal.uiuc.edu/pub/src/ECGA/chiECGA.tgz

[6]http://www.psi.toronto.edu/affinitypropagation/apcluster.txt

running times of the algorithms for functions with different domains types of difficulty. The scalability of AffEDA is also investigated. We used the Matlab implementation of ECGA and AffEDA discussed in Section 5.3 to conduct preliminary experiments. The experiments, the results of which are presented in this paper, have been done using the C++ implementations. We start by introducing the testbed functions and problems used in our comparisons and the rationale behind our choice.

6.1 Function Benchmark

In the function benchmark used to test the algorithms we include a number of additively decomposable deceptive functions (Deb and Goldberg, 1991; Goldberg, 1989) and a nonadditively decomposable problem corresponding to a simplified protein model (Dill, 1985).

6.1.1 Deceptive Functions

Let $u(\mathbf{x}) = \sum_{i=1}^{n} x_i$. $f(\mathbf{x})$ is a unitation function if $\forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, u(\mathbf{x}) = u(\mathbf{y}) \Rightarrow f(\mathbf{x}) = f(\mathbf{y})$. A unitation function is defined in terms of its unitation value $u(\mathbf{x})$, or using a simpler notation, u. Unitation functions serve for the definition of binary deceptive functions, so called because they show the deceptive nature of the simple GA behavior for functions with strong interactions (Deb and Goldberg, 1991; Goldberg, 1989).

$f_{deceptivek}(\mathbf{x})$ (introduced as the Trap function in Ackley, 1987) is a deceptive function formed by the additive sum of deceptive subfunctions with definition sets that do not overlap. It represents a class of parametric deceptive function, where k is a parameter that determines the order of the subfunctions

$$f_{deceptivek}(\mathbf{x}) = \sum_{i=1}^{\frac{n}{k}} f_{dec}^{k}(x_{k\cdot(i-1)+1} + x_{k\cdot(i-1)+2} + \cdots + x_{k\cdot i}) \tag{11}$$

where

$$f_{dec}^{k}(u) = \begin{cases} k-1 & \mathit{for}\ u = 0 \\ k-2 & \mathit{for}\ u = 1 \\ k-i-1 & \mathit{for}\ u = i \\ k & \mathit{for}\ u = k \end{cases}$$

We include in our testbed the $f_{deceptive}(\mathbf{x})$ function (Goldberg, 1987) for which the difference between the optimum fitness value and the closest suboptimum value is smaller than for function $f_{deceptivek}$.

$$f_{deceptive}(\mathbf{x}) = \sum_{i=1}^{\frac{n}{3}} f_{Gdec}(x_{3i\text{-}2} + x_{3i\text{-}1} + x_{3i}) \tag{12}$$

$$f_{Gdec}(u) = \begin{cases} 0.9 & \mathit{for}\ u = 0 \\ 0.8 & \mathit{for}\ u = 1 \\ 0.0 & \mathit{for}\ u = 2 \\ 1.0 & \mathit{for}\ u = 3 \end{cases}$$

The analysis of the behavior of discrete EDAs has focused mainly on binary problems. However, to study the robustness of EDAs and confront a general class of applications, an analysis of nonbinary problems is necessary. To this end, we use a deceptive

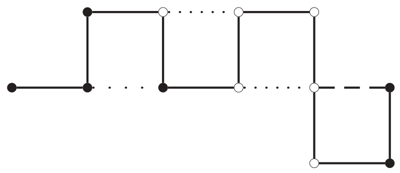

Figure 1: One possible configuration of sequence $HHHPHPPPPPHH$ in the HP model. There is one HH (represented by a dotted line with wide spaces), one HP (represented by a dashed line) and two PP (represented by dotted lines) contacts.

function defined on nonbinary variables.

$$f^{c}_{deceptivek}(\mathbf{x}) = \sum_{i=1}^{\frac{n}{k}} F_{dec}(x_{k\cdot(i-1)+1}, x_{k\cdot(i-1)+2}, \ldots, x_{k\cdot i}, k, c) \tag{13}$$

$$F_{dec}(x_1, \ldots, x_k, k, c) = \begin{cases} k \cdot (c-1), & for \sum_{i=1}^{k} x_i = k \cdot (c-1) \\ k \cdot (c-1) - \sum_{i=1}^{k} x_i - 1 & otherwise \end{cases} \tag{14}$$

The general deceptive function $f^{c}_{deceptivek}(\mathbf{x})$ of order k (Santana et al., 2002) is formed as an additive function composed by the function $F_{dec}(x_1, \ldots, x_k, k, c)$ evaluated on substrings of size k and cardinality c, that is, $x_i \in \{0, \ldots, c-1\}$. This function is a generalization of the binary $f_{deceptivek}(\mathbf{x})$ to variables with nonbinary values. Note that $f_{deceptivek}(\mathbf{x}) = f^{2}_{deceptivek}(\mathbf{x})$.

6.1.2 The HP Protein Model

Under specific conditions, a protein sequence folds into a native 3D structure. The problem of determining the protein native structure from its sequence is known as the protein structure prediction problem. In order to solve this problem, a protein model is chosen and an energy is associated to each possible protein fold. The search for the protein structure is transformed into the search for the optimal protein configuration given the energy function. We use a class of coarse-grained models called the hydrophobic-polar (HP) model (Dill, 1985).

The HP model considers two types of residues: hydrophobic (H) residues and hydrophilic or polar (P) residues. In the model, a protein is considered as a sequence of these two types of residues, which are located in regular lattice models forming self-avoided paths. Given a pair of residues, they are considered neighbors if they are adjacent either in the chain (connected neighbors) or in the lattice but not connected in the chain (topological neighbors). The total number of topological neighboring positions in the lattice (z) is called the lattice coordination number. Figure 1 shows one possible configuration of sequence $HHHPHPPPPPHH$ in the HP model.

A solution $\mathbf{x}$ can be interpreted as a walk in the lattice, representing one possible folding of the protein. We use a discrete representation of the solutions. For a given sequence and lattice, X_i will represent the relative move of residue i in relation to the previous two residues. Taking as a reference the location of the previous two residues in the lattice, X_i takes values in $\{0, 1, \ldots, z-2\}$, where $z-1$ is the number of movements

Table 1: HP instances used in the experiments.

Inst.	Size	$H^2(\mathbf{x}^*)$	$H^3(\mathbf{x}^*)$	Sequence
$s7$	60	-36		$PPH^3PH^8P^3H^{10}PHP^3H^{12}P^4H^6PHHPHP$
$s10$	100	-48		$P^6HPH^2P^5H^3PH^5PH^2P^4H^2P^2H^2PH^5PH^{10}PH^2PH^7$ $P^{11}H^7P^2HPH^3P^6HPH$
$s11$	100	-50		$P^3H^2P^2H^4P^2H^3PH^2PH^2PH^4P^8H^6P^2H^6P^9HPH^2$ $PH^{11}P^2H^3PH^2PHP^2HPH^3P^6H^3$
$s14$	103		-54	$P^2H^2P^5H^2P^2H^2PHP^2HP^7HP^3H^2PH^2P^6HP^2HPH$ $P^2HP^5H^3P^4H^2PH^2P^5H^2P^4H^4PHP^8H^5P^2HP^2$
$s15$	124		-71	$P^3H^3PHP^4HP^5H^2P^4H^2P^2H^2P^4HP^4HP^2HP^2H^2P^3$ $H^2PHPH^3P^4H^3P^6H^2P^2HP^2HPHP^2HP^7$ $HP^2H^3P^4HP^3H^5P^4H^2PH^4$

allowed in the given lattice. These values respectively mean that the new residue will be located in one of the $z-1$ numbers of possible directions with respect to the previous two locations. Therefore, values for X_1 and X_2 are meaningless. The locations of these two residues are fixed. For example, the codification of the HP configuration shown in Figure 1 is $\mathbf{x} = (0, 0, 0, 2, 2, 0, 0, 2, 2, 1, 0, 0)$.

The codification used is called relative encoding, and has been experimentally compared to a different encoding called absolute encoding in Krasnogor et al. (1999), showing better results. In the experiments presented in this section we use 2D and 3D regular lattices. For general regular d-dimensional lattices, $z = 2d$.

For the HP model, an energy function that measures the interaction between topological neighbor residues is defined as $\epsilon_{HH} = -1$ and $\epsilon_{HP} = \epsilon_{PP} = 0$. The HP problem consists of finding the solution that minimizes the total energy. The problem of finding such a minimum energy configuration is NP-complete for the 2D lattice (Crescenzi et al., 1998). Performance-guaranteed approximation algorithms of bounded complexity have been proposed to solve this problem (Hart and Istrail, 1996), but the error bound guarantee is not small enough for many applications. Work on evolutionary search applied to protein structure prediction and protein folding for lattice models and real proteins have been surveyed in Greenwood and Shin (2002). A well documented review of current approaches to protein structure prediction is Ginalski et al. (2005).

Some of the HP instances used in our experiments, and shown in Table 1, have previously been used in several papers (Cutello et al., 2007; Hsu et al., 2003; Santana et al., 2008). The values shown in Table 1 correspond to the best-known solutions ($H^k(x^*)$) for the k-d regular lattice. The instances shown in the table are among the longest used in the literature. Instance $s7$ has been shown to be deceptive for a number of EDAs that consider short order dependencies (Santana et al., 2008) and therefore we are interested in studying the way ECGA and AffEDA behave on it.

6.2 Numerical Results

6.2.1 Scalability of the Algorithm

We begin by comparing the scalability of AffEDA with that of ECGA for function $f_{deceptivek}(\mathbf{x})$ with $k \in \{3, 4, 5\}$. We have adopted the experimental framework used in Harik et al. (2006), where a systematic study of ECGA is done, showing its advantage over simple GAs. The number of deceptive subproblems considered in Harik et al. (2006) ranges from 2 to 27 for function $f_{deceptive3}(\mathbf{x})$, and from 2 to 20 for functions $f_{deceptive4}(\mathbf{x})$ and $f_{deceptive5}(\mathbf{x})$. The minimal number of function evaluations required to

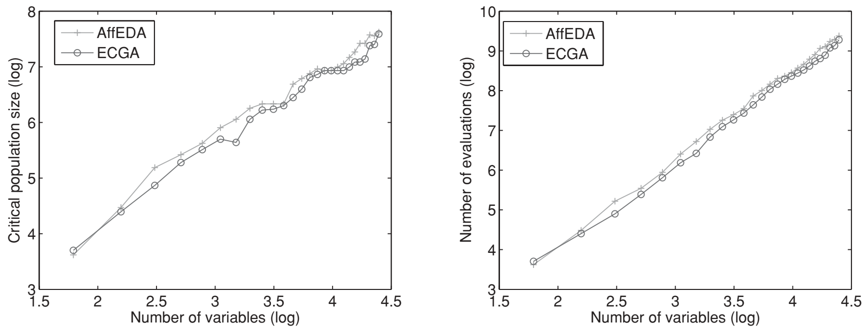

Figure 2: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive3}(\mathbf{x})$.

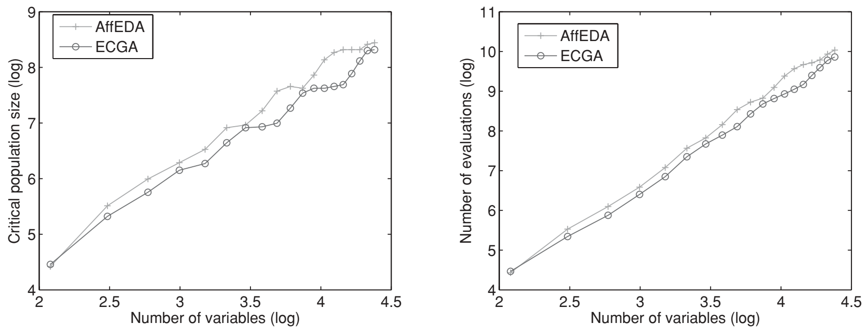

Figure 3: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive4}(\mathbf{x})$.

solve all but one subproblem is recorded, that is, the optimal solution with an error of $\alpha = \frac{1}{m}$, where m is the number of subproblems.

In order to find the minimal sufficient (critical) population size needed to achieve a target solution, we start with a population size $M = 16$ and double the population size every time the algorithm fails to find the target solution in at least one of 30 consecutive trials. The results for the critical population size are averaged over 30 experiments. For each experiment, the number of subproblems solved with a given population size is averaged over another 30 runs. Therefore, the average number of function evaluations is calculated from 900 independent runs. For all the experiments, tournament selection without replacement is used with tournament size 16. The algorithm stops when a solution that has all but one solved subproblem has been found or the population has converged to a unique individual.

Figures 2, 3, and 4 show the critical population size and number of function evaluations required to solve all but one subproblem when n is increased. It can be seen from the figures that for a small number of variables the performance of the algorithms is almost the same. When the number of variables is increased, ECGA is better than AffEDA. However, as is shown in the figures, both algorithms have similar scalability. Another observation is the way in which the complexity of the problems is increased

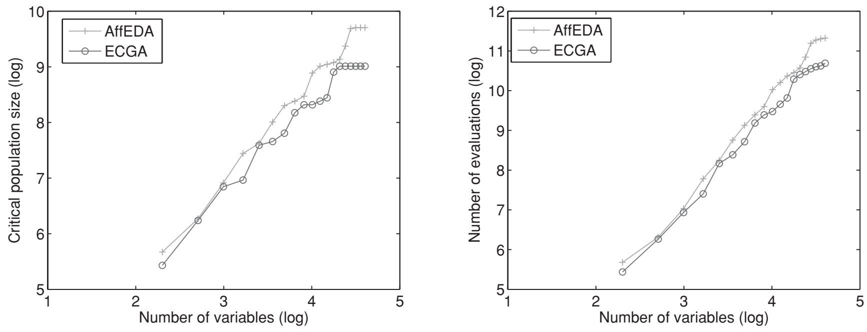

Figure 4: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive5}(\mathbf{x})$.

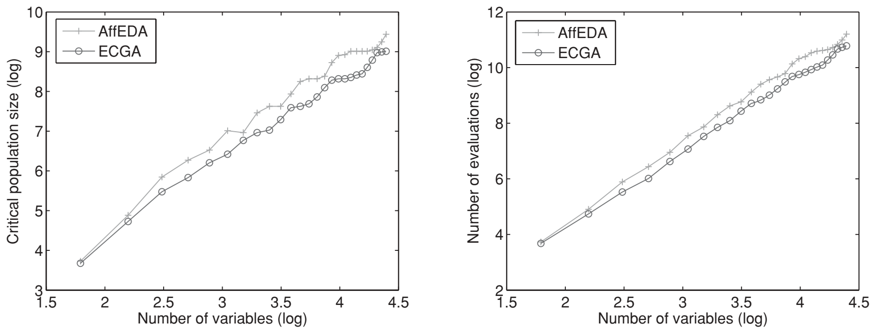

Figure 5: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive}(\mathbf{x})$.

with k. The increase in this complexity is noticeable. These results are in agreement with what is expected, that is, the number of evaluations grows exponentially with k.

We also conduct scalability experiments for function $f_{deceptive}(\mathbf{x})$. The number of deceptive problems considered in Harik et al. (2006) ranges from 2 to 27. Figure 5 shows the critical population size and number of function evaluations required to solve all but one subproblem. Also, in this case, for a small number of variables, the performance of the algorithms is almost the same. However, when the number of variables is increased, AffEDA needs a bigger population size and a higher number of function evaluations than ECGA to solve function $f_{deceptive}(\mathbf{x})$. The difference is slightly more significant than in the case of function $f_{deceptive3}(\mathbf{x})$ (see Figure 2).

6.2.2 Computational Cost of the Algorithms

In Section 5.1, we derived an estimate of the computational cost of AffEDA which is $O(G(Nn^2r_{MAX}^2 + n^2depth(PImaxiter + AFmaxiter) + m(N + \delta + r_{MAX}^{\delta}) + mMr_{MAX}^{delta} + Mcost_f))$. In this section we present empirical evidence on the efficiency advantage of AffEDA over ECGA.

ECGA and AffEDA are run using their respective critical population sizes that have been computed in the previous section for functions $f_{deceptive3}(\mathbf{x})$, $f_{deceptive4}(\mathbf{x})$

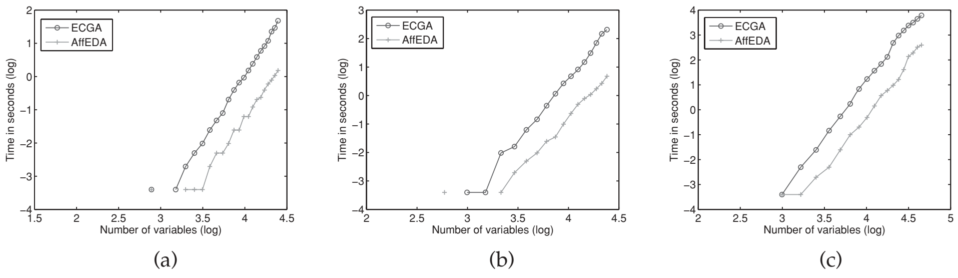

Figure 6: Scalability of the computational time of AffEDA and ECGA when the number of variables is increased for functions: (a) $f_{deceptive3}(\mathbf{x})$, (b) $f_{deceptive4}(\mathbf{x})$, and (c) $f_{deceptive5}(\mathbf{x})$.

and $f_{deceptive5}(\mathbf{x})$. Figure 6 shows the logarithm of the average time needed by these algorithms to solve all but one subproblem of the functions when n is increased. Time was measured in seconds. For some small values of n, the algorithms were able to find the best solution in less than 1 s in all the experiments. Since Figure 6 shows the logarithm of the time, these cases are not plotted in the graphs. It can be seen that as n is increased, AffEDA requires much less time than ECGA than for all the functions.

We make another experiment to evaluate the difference in the time required by both algorithms. The main goal of the following experiments is to show that since AffEDA is faster than ECGA, the user will have to wait less time to achieve l successful runs of the former algorithm. Similar to previous experiments, we consider that a run is successful when a solution is found for which at least all but one subproblem have been solved. This solution is called acceptable. The time of a successful run is the time consumed until an acceptable solution is found. Similarly, the time of an unsuccessful run is the time spent by the algorithm that does not find an acceptable solution. In this case, the algorithm stops when the population has converged to a unique individual.

If for achieving l successful runs the algorithm needs to be executed $l^* \geq l$ times, we need to add to the total time consumed by the l successful runs the time consumed by the unsuccessful runs (i.e., the time consumed by each of the remaining $l^* - l$ runs). Note that for $l = 1$, what we measure is the time needed to find an acceptable solution.

Each experiment consists of running each EDA an $l^* \geq l$ number of times until 30 successful outcomes of the algorithm have been attained. Functions $f_{deceptivek}(\mathbf{x})$ with $k \in \{3, 4, 5\}$ are used in our experiments. For these functions, the number of variables were $n = 81$, $n = 80$, and $n = 100$, respectively. The average critical population sizes computed for ECGA in the experiments shown in Section 6.2.1 were used as a base to determine the population size used in these experiments. Note that this is a constraint for AffEDA, since it is run using a population size smaller than the critical size.

Due to the fact that the replacement strategy needs the population size to be a multiple of the tournament size, the population size we use is the first multiple of 16 higher than the average critical population size estimated for ECGA. The population sizes used are shown in Table 2.

After every successful outcome, we compute the total time elapsed. The time consumed by unsuccessful outcomes is included in this sum. Thirty experiments are conducted and from them the average expected time (in seconds) to find the lth successful outcome is computed. Table 2 shows the mean and standard deviation needed by

Table 2: Mean $\mu(t)$ and standard deviation $\sigma(t)$ of the time (in seconds) of a successful run of AffEDA and ECGA for different functions and number of variables.

			AffEDA		ECGA	
F	n	M	$\mu(t)$	$\sigma(t)$	$\mu(t)$	$\sigma(t)$
$f_{deceptive3}$	81	1,984	192.90	16.53	713.53	47.34
$f_{deceptive4}$	80	4,096	573.70	28.50	1,645.00	33.01
$f_{deceptive5}$	100	8,192	6,352.37	1,844.50	9,625.27	3,478.98

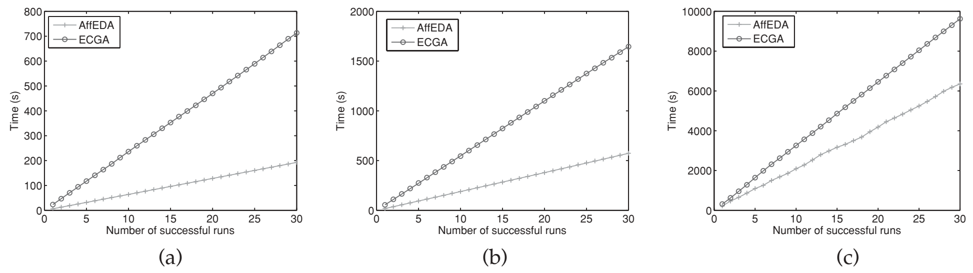

Figure 7: Elapsed time before finding l, not necessarily consecutive, successful runs of AffEDA and ECGA for functions: (a) $f_{deceptive3}(\mathbf{x})$, (b) $f_{deceptive4}(\mathbf{x})$, and (c) $f_{deceptive5}(\mathbf{x})$. For each experiment, the number of variables, the population size, and the maximum number of generations are kept fixed.

AffEDA and ECGA to attain 30 successful runs of the algorithm. The average time needed to find the optimum is computed from the 900 successful runs.

Figure 7 displays the average elapsed time needed to achieve l successful runs, with $1 \le l \le 30$. It can be seen that the time needed by AffEDA is less than that needed by ECGA for all the functions. This is the case even if AffEDA is run using half of its critical population size as happens for function $f_{deceptive5}(\mathbf{x})$, as in Figure 7(c). However, as the complexity of the function increases, the difference in the time consumed by both algorithms is reduced. This can be explained by the fact that AffEDA does not use its critical population size and the total time cost of its unsuccessful runs is increased.

6.2.3 Influence of the Variables' Cardinality

We investigate the performance of AffEDA and ECGA for function $f^{c}_{deceptivek}(\mathbf{x})$, $k \in \{3, 4, 5\}$, $c \in \{3, 4, 5, 6\}$. We are interested in evaluating both algorithms when the size of the definition sets and the number of values of each variable are increased. In this case, the number of variables is fixed. In order to be consistent with previous experiments, we find the critical population size required to solve all but one subproblem in 30 consecutive runs of the algorithms. Additionally, we compute the average time needed by each algorithm to find the best solution. The method used to determine the critical population size was the same as in previous experiments. The maximum population size (M) used was 251,504. Even for this population, the algorithms were not able to reach 30 successful runs in some experiments. The number of experiments conducted was 30. The results of the experiments are shown in Table 3, where results for those combinations of values k and c for which a critical population size could not be found are omitted.

Table 3: Critical population size (M), number of evaluations (e) and time (t), (in seconds), needed by AffEDA and ECGA to solve $f^{c}_{deceptivek}(\mathbf{x})$ functions with different values of k and c. Results are averaged over 900 experiments.

		AffEDA			ECGA		
F	n	M	e	t	M	e	t
$f^{3}_{deceptive3}$	30	4,096	11,919	1.50	2,389	7,256	1.52
$f^{4}_{deceptive3}$		10,103	37,100	4.60	8,465	31,594	8.41
$f^{5}_{deceptive3}$		32,768	**130,417**	15.45	32,768	130,671	35.08
$f^{6}_{deceptive3}$		65,536	**296,077**	34.46	131,072	563,901	153.60
$f^{3}_{deceptive4}$	32	10,103	29,992	4.23	8,738	26,351	12.14
$f^{4}_{deceptive4}$		65,536	**233,454**	31.21	128,887	436,032	197.56
$f^{3}_{deceptive5}$	30	37,137	**96,192**	17.78	61,167	154,519	75.36

A first conclusion from the analysis of Table 3 is that in all cases AffEDA takes less time (t) than ECGA to find the solutions. This result is consistent with the previous experiments and shows that the efficiency of the algorithm is kept when the cardinality of the variables is increased. Another remarkable fact is that the number of function evaluations needed by AffEDA to solve all but one subproblem of functions $f^{5}_{deceptive3}(\mathbf{x})$, $f^{6}_{deceptive3}(\mathbf{x})$, $f^{4}_{deceptive4}(\mathbf{x})$, and $f^{3}_{deceptive5}(\mathbf{x})$ is smaller than those needed by ECGA. In order to emphasize this fact, these cases appear in bold in the table.

The difference in the behavior of the algorithms seems to indicate that ECGA is more sensitive to the increase in the cardinality of the variables, particularly if this increment is combined with an increment in the size of the definition sets. We hypothesize that the estimates of the higher-order marginal probabilities used to learn the ECGA model degrade too much when the size of tables is increased. Approximations constructed by the bivariate estimates used by AffEDA might be more accurate in these cases.

A fact that illustrates the complexity of the functions under analysis is that none of the algorithms was able to reach 30 consecutive successful runs for five of the functions tested using a population size as high as 251,504. Note that we kept the tournament size fixed at 16 and this and other parameters used by both EDAs were not tuned to improve the convergence results. Also, it is important to emphasize that due to the way in which the critical population size has been computed, population sizes shown in Table 3 should be considered as an upper bound of the minimally required population size to solve this problem.

6.2.4 Accuracy and Computational Cost of the Structure Learning Step

We investigate in detail the differences between ECGA and AffEDA by looking at the accuracy of the structures they learn and the time spent to learn these structures. For additively decomposable functions, it has been shown that an EDA that uses the perfect structure of the model (i.e., the factorized distribution algorithm) can efficiently solve the function. The efficiency of EDAs that do structural learning of the model is also linked to their ability to recover the problem structure.

For a number of additive functions, and for different population sizes, we compute how many of the original interactions of the problem are captured in the structures learned by AffEDA and ECGA. We also compute how many of other structural dependencies are learned. The first class of dependencies we call "correct" and the second

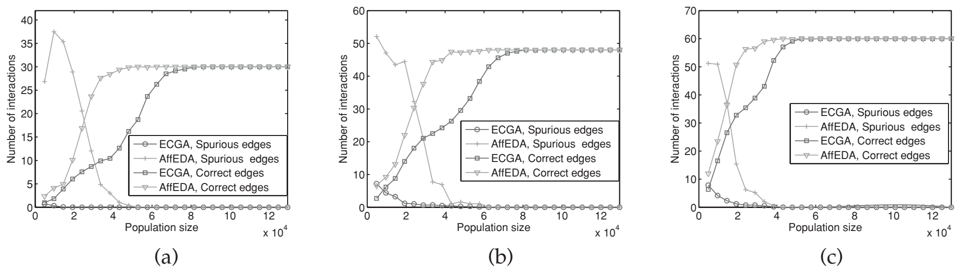

Figure 8: Number of correct and spurious edges learned in the first generation of AffEDA and ECGA when the population size is increased for functions: (a) $f^{6}_{deceptive3}(\mathbf{x})$, (b) $f^{4}_{deceptive4}(\mathbf{x})$, and (c) $f^{3}_{deceptive5}(\mathbf{x})$.

class we call "spurious." We constrain our analysis to the models learned in the first generation because the algorithms can take different number of generations, making it difficult to analyze and visualize the information of the models learned at different generations.

Figure 8 shows the number of correct and spurious edges learned in the first generation of AffEDA and ECGA when the population size is increased from $N = 4{,}800$ to $N = 129{,}600$ for three functions, (a) $f^{6}_{deceptive3}(\mathbf{x})$, (b) $f^{4}_{deceptive4}(\mathbf{x})$, and (c) $f^{3}_{deceptive5}(\mathbf{x})$. The results are the average of 30 independent experiments for each of the functions.

It can be observed that from a high enough value of the population size, both algorithms are able to learn a perfect model of the structure. However, ECGA needs a higher population size for all the problems. The difference is remarkable considering that the population size is increased by 4,800 in each step. There are also differences in the behavior of both algorithms when confronted with a small population size. In these cases, AffEDA learns a high number of spurious edges and ECGA learns very few dependencies. As the population size is incremented, both algorithms move to a model with all correct interactions and without spurious correlations.

We also investigate the time spent by the algorithms in the first generation for the different population sizes. We decided to compute the total time spent in the first generation and not only the time spent by the learning step. This decision was due to the fact that the more complex models learned by AffEDA could be more costly to sample than those learned by ECGA. Therefore, the total time gives a wider perspective of the differences determined by using the two learning algorithms.

Figure 9 shows the time spent in the first generation of AffEDA and ECGA when the population size is increased for three functions, (a) $f^{6}_{deceptive3}(\mathbf{x})$, (b) $f^{4}_{deceptive4}(\mathbf{x})$, and (c) $f^{3}_{deceptive5}(\mathbf{x})$. It can be seen that the time required by AffEDA is less than that of ECGA in all the cases.

The results presented in this section for functions $f^{6}_{deceptive3}(\mathbf{x})$, $f^{4}_{deceptive4}(\mathbf{x})$, and $f^{3}_{deceptive5}(\mathbf{x})$ show that the learning algorithm used by AffEDA is able to construct more accurate models than ECGA in less time. The experiments also offer some clues that could explain the premature convergence of AffEDA for other decomposable functions when the population size is below the critical value. We hypothesize that in these cases, the models learned by AffEDA may contain a high number of spurious correlations, misleading the search away from the promising areas.

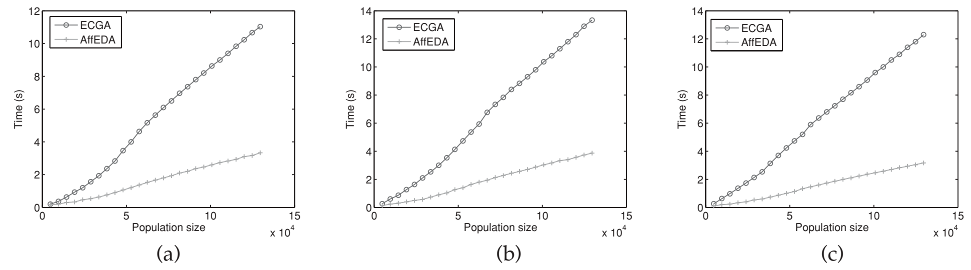

Figure 9: Time spent in the first generation of AffEDA and ECGA when the population size is increased for functions (a) $f^6_{deceptive3}(\mathbf{x})$, (b) $f^4_{deceptive4}(\mathbf{x})$, and (c) $f^3_{deceptive5}(\mathbf{x})$.

Table 4: Results of AffEDA and ECGA for different protein instances of the HP problem.

				AffEDA		ECGA	
Inst	*d*	*psize*	*T*	*best*	$\mu(e)$	*best*	$\mu(e)$
*S*7	2	32,768	64	−35	−32.20	−35	−33.32
*S*7	2	32,768	256	−35	−33.84	−35	−33.98
*S*10	2	32,768	64	−42	−39.96	−43	−39.58
*S*10	2	32,768	256	−43	−40.20	−42	−40.10
*S*11	2	32,768	64	−46	−43.58	−46	−42.82
*S*11	2	32,768	256	−47	−44.14	−46	−44.04
*S*14	3	65,536	64	−48	−43.56	−49	−44.36
S14	3	65,536	256	−50	−44.02	−47	−44.32
*S*15	3	65,536	64	−59	−55.14	−60	−55.00
*S*15	3	65,536	256	−60	−55.44	−62	−55.94

6.2.5 Results for the HP Model

We compare the behavior of algorithms AffEDA and ECGA for the HP instances shown in Table 1. The objective of our experiment is to investigate the performance of AffEDA for a class of problems that combines different domains of difficulty: nonbinary, higher order interactions between the variables and the existence of constraints. Although the MPMs seem not to be the most appropriate models for representing the type or dependencies arising in the HP problem, we research to what extent AffEDA can cope with this class of difficult problems. The results for ECGA are included for comparison.

Table 4 shows the results achieved by AffEDA and ECGA for different protein instances of the HP problem. In the table, *Inst* indicates the instance from Table 1, d refers to the d-regular lattice in which instances were folded, *psize* is the population size, and T is the tournament size used. The variables *best* and $\mu(e)$ are respectively the best fitness and average fitness found from the set of independent experiments conducted. Both algorithms were run for a maximum of 200 generations. The number of experiments was 50. Due to the computational cost, they were executed in a cluster of computers of different architectures. Therefore, computing times are not available from these experiments. However, regarding the differences in the time spent by the algorithms, the same trend appreciated in previous experiments was manifested.

Regarding the results, there are few differences between both algorithms. We applied the Wilcoxon rank sum test for equal medians on the average results of the two

Table 5: Results of AffEDA and ECGA for different protein instances of the HP problem.

				AffEDA			ECGA		
Inst	*d*	*M*		*runs*	*best*	*ntimes*	*runs*	*best*	*ntimes*
*S*7	2	32,768	*Best*1	33	−35	7	23	−35	5
	2	32,768	*Best*2		−34	13		−34	10
*S*10	2	32,768	*Best*1	16	−42	1	8	−42	1
	2	32,768	*Best*2		−41	7		−41	4
*S*11	2	32,768	*Best*1	16	−47	1	10	−47	1
	2	32,768	*Best*2		−46	1		−46	2
*S*14	3	65,536	*Best*1	5	−45	1	3	−45	1
	3	65,536	*Best*2		−44	2		−44	1
*S*15	3	65,536	*Best*1	5	−60	1	3	−62	1
	3	65,536	*Best*2		−58	1		−57	1

algorithms for each problem. Observed differences were not statistically significant. It has to be pointed out, that for the 2D lattice problems, the algorithms were not able to achieve the best fitness solutions found by EDAs that use mixture and Markov probabilistic models (Santana et al., 2008), which allow overlapping sets of variables.

We conducted an additional experiment to compare the efficiency of AffEDA and ECGA for the HP problem. Both algorithms were allowed to run in computers with the same architecture for 24 h, executing as many runs of each EDA as possible. Both algorithms used tournament size 256 and a maximum of 200 generations. After the time period elapsed, we computed the number of completed executions for each algorithm and evaluated the quality of the solutions obtained. These results are shown in Table 5. The value corresponding to the best and second best solution are shown, as well as the number of times in which these solutions were found. An analysis of the table reveals that in every case, more executions of AffEDA were possible in the same time. The quality of the best solution was similar in each case except for instance *S*15, for which ECGA achieves a better result. However, the number of times in which the best and second best solutions are achieved is higher in AffEDA than in ECGA.

7 Conclusions and Future Work

In this paper, we introduced AffEDA, an EDA based on the use of MPMs. The distinguishing feature of our algorithm is that the factorization is efficiently learned from the data using affinity propagation. We have analytically and empirically shown that the structural learning algorithm used by AffEDA is faster than the one used by ECGA for all the problems considered. Our experiments have also shown that AffEDA is more efficient than ECGA to solve deceptive nonbinary problems with different cardinality. The behaviors of AffEDA and ECGA have been tested on a class of simplified protein problems and for this class of problems both algorithms exhibit a similar behavior in terms of quality of solutions, where AffEDA is faster than ECGA.

We have introduced the use of message-passing algorithms such as affinity propagation for learning the structure of a probabilistic model. To the knowledge of the authors, there are no other reports on applications of message-passing algorithms in this domain. In EDAs, previous applications of message-passing algorithms have been constrained either to learn a set of consistent parameters for a given probabilistic model (Ochoa et al., 2003; Höns, 2006) or to sample a set of solutions by means of the *k* most probable configurations found applying max-propagation (Höns et al., 2007; Mendiburu et al., 2007; Mühlenbein and Höns, 2006; Soto, 2003).

We note that while the original affinity propagation algorithm (Frey and Dueck, 2007) does not impose constraints on the size of the clusters found, our recursive version allows the incorporation of this sort of constraint. Our modified algorithm could also be applied to clustering problems that use different similarity measures.

Our findings from the results achieved by AffEDA and ECGA indicate that in addition to the choice of the probabilistic model, care must be taken in the selection of the learning algorithm. While other approaches to model-building speed up (e.g., sporadic model building, Pelikan et al., 2008) place emphasis on the frequency in which the structural learning is applied, we focus on the balance between the accuracy of the learning algorithm and its time complexity. Furthermore, we have shown that applying structural learning algorithms that use higher-order dependencies does not always imply better convergence results, as the case of problems with high cardinality illustrates. We note that the different approaches to speeding up model learning could be combined. Another contribution of this paper lies in the investigation of ECGA for the class of nonbinary discrete problems, for which few studies have been accomplished.

Finally, we mention some of the areas worth researching further.

1. The question of the relationship between the accuracy of the learning algorithm to retrieve the probabilistic model and its cost in terms of time transcends the case of EDAs that use MPMs. For some EDAs, such as the estimation of Bayesian network algorithm (EBNA; Etxeberria and Larrañaga, 1999), the question of whether the use of “exact” learning methods, which are guaranteed to learn the best model, increases the algorithm’s efficiency has been investigated (Echegoyen et al., 2007). This investigation has been carried out for small problems, for which exact learning is computationally feasible. A similar analysis for the case of MPMs could provide a better understanding of the behavior of ECGA and AffEDA.

2. Recent research on the affinity propagation algorithm (Leone et al., 2007) has produced a more robust version of the algorithm, less dependent on the external tuning of the parameters. The results obtained show that affinity propagation can be relaxed to learn more than one exemplar, leading to more stable clusterings. These improvements could allow the future use of this message-passing method for clustering variables in overlapping groups, allowing the algorithm to go beyond the class of MPMs analyzed in this paper.

3. Affinity propagation uses the preferences or self similarity values to bias the selection of the exemplars. This parameter could be employed to add a priori information about the problem structure to the model-learning phase. Variables that are known to belong to different factors can be assigned a higher self similarity value.

4. Finally, although our analysis has been restricted to problems with integer representation, the algorithms proposed in this paper could be adapted for problems with continuous representation for which a similarity measure of the interactions between every pair of variables is available.

Acknowledgments

This work has been partially supported by the Saiotek and Research Groups 2007-2012 (IT-242-07) programs (Basque Government), TIN2008-06815-C02-01, TIN2008-06815-C02-02, TIN 2007-62626, the CajalBlueBrain project, and Consolider Ingenio 2010-CSD2007-00018 projects (Spanish Ministry of Science and Innovation), and COMBIOMED network in computational biomedicine (Carlos III Health Institute). The SGI/IZO-SGIker UPV/EHU (supported by the Spanish Program for the Promotion of Human Resources within the National Plan of Scientific Research, Development and Innovation—Fondo Social Europeo and MCyT) is gratefully acknowledged for its generous allocation of computational resources.

References

Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, pp. 170–204.

Babel, L., Kellerer, H., and Kotov, V. (1998). The k-partitioning problem. *Mathematical Methods of Operations Research*, 47(1):59–82.

Braunstein, A., Mézard, M., and Zecchina, R. (2005). Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226.

Bron, C., and Kerbosch, J. (1973). Algorithm 457—Finding all cliques of an undirected graph. *Communications of the ACM*, 16(6):575–577.

Crescenzi, P., Goldman, D., Papadimitriou, C. H., Piccolboni, A., and Yannakakis, M. (1998). On the complexity of protein folding. *Journal of Computational Biology*, 5(3):423–466.

Cutello, V., Nicosia, G., Pavone, M., and Timmis, J. (2007). An immune algorithm for protein structure prediction on lattice models. *IEEE Transactions on Evolutionary Computation*, 11(1):101–117.

de la Ossa, L., Sastry, K., and Lobo, F. G. (2006). χ-ary extended compact genetic algorithm in C++. IlliGAL Report 2006013, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Deb, K., and Goldberg, D. E. (1991). Analyzing deception in trap functions. IlliGAL Report 91009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Deming, W. E., and Stephan, F. F. (1940). On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *Annals of Mathematical Statistics*, 11(4):427–444.

Dill, K. A. (1985). Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509.

Duque, T. S. P. C., Goldberg, D. E., and Sastry, K. (2008). Enhancing the efficiency of the ECGA. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature, PPSN X*, Vol. 5199 of *Lecture Notes in Computer Science*, pp. 165–174. Berlin: Springer.

Echegoyen, C., Lozano, J. A., Santana, R., and Larrañaga, P. (2007). Exact Bayesian network learning in estimation of distribution algorithms. In *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pp. 1051–1058.

Etxeberria, R., and Larrañaga, P. (1999). Global optimization using Bayesian networks. In A. Ochoa, M. R. Soto, and R. Santana (Eds.), *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pp. 151–173.

Frey, B. J., and Dueck, D. (2006). Mixture modeling by affinity propagation. In *Proceedings of the 2005 Conference Advances in Neural Information Processing Systems 18, NIPS*, pp. 379–386.

Frey, B. J., and Dueck, D. (2007). Clustering by passing messages between data points. *Science*, 315:972–976.

Gámez, J. A., Mateo, J. L., and Puerta, J. M. (2007). EDNA: Estimation of dependency networks algorithm. In J. Mira and J. R. Álvarez (Eds.), *Bio-inspired Modeling of Cognitive Tasks, Second International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2007*, Vol. 4527 of *Lecture Notes in Computer Science* (pp. 427–436). Berlin: Springer.

Gámez, J. A., Mateo, J. L., and Puerta, J. M. (2008). Improved EDNA (estimation of dependency networks algorithm) using combining function with bivariate probability distributions. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation GECCO-2008*, pp. 407–414.

Ginalski, K., Grishin, N. V., Godzik, A., and Rychlewski, L. (2005). Practical lessons from protein structure prediction. *Nucleid Acids Research*, 33(6):1874–1891.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing* (pp. 74–88). London: Pitman Publishing.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Greenwood, G. W., and Shin, J.-M. (2002). On the evolutionary search for solutions to the protein folding problem. In G. B. Fogel and D. W. Corne (Eds.), *Evolutionary computation in bioinformatics* (pp. 115–136). San Mateo, CA: Morgan Kaufmann.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, Ann Arbor. Also IlliGAL Report No. 97005.

Harik, G. R., Lobo, F. G., and Sastry, K. (2006). Linkage learning via probabilistic modeling in the ECGA. In M. Pelikan, K. Sastry, and E. Cantú-Paz (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 39–62). Berlin: Springer.

Hart, W. E., and Istrail, S. C. (1996). Fast protein folding in the hydrophobic-hydrophilic model within three-eights of optimal. *Journal of Computational Biology*, 3(1):53–96.

Hartmann, A. K., and Weigt, M. (2005). *Phase transitions in combinatorial optimization problems: Basics, algorithms and statistical mechanics*. New York: John Wiley & Sons.

Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R., and Kadie, C. M. (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243.

Höns, R. (2006). *Estimation of distribution algorithms and minimum relative entropy*. PhD thesis, University of Bonn, Bonn, Germany.

Höns, R., Santana, R., Larrañaga, P., and Lozano, J. A. (2007). Optimization by max-propagation using Kikuchi approximations. Tech. Rep. EHU-KZAA-IK-2/07, Department of Computer Science and Artificial Intelligence, University of the Basque Country.

Hsu, H.-P., Mehra, V., Nadler, W., and Grassberger, P. (2003). Growth algorithms for lattice heteropolymers at low temperatures. *Journal of Chemical Physics*, 118(1):444–451.

Krasnogor, N., Hart, W. E., Smith, J., and Pelta, D. A. (1999). Protein structure prediction with evolutionary algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 1596–1601.

Larrañaga, P., and Lozano, J. A. (Eds.). (2002). *Estimation of distribution algorithms. A new tool for evolutionary computation*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Leone, M., Sumedha, and Weigt, M. (2007). Clustering by soft-constraint affinity propagation: Applications to gene-expression data. *Bioinformatics*, 23(20):2708–2715.

Mendiburu, A., Santana, R., and Lozano, J. A. (2007). Introducing belief propagation in estimation of distribution algorithms: A parallel framework. Tech. Rep. EHU-KAT-IK-11/07, Department of Computer Science and Artificial Intelligence, University of the Basque Country.

Mézard, M. (2007). Where are the good exemplars? *Science*, 315:949–951.

Mézard, M., Parisi, G., and Zechina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812.

Mühlenbein, H., and Höns, R. (2006). The factorized distributions and the minimum relative entropy principle. In M. Pelikan, K. Sastry, and E. Cantú-Paz (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 11–38). Berlin: Springer.

Mühlenbein, H., and Mahnig, T. (2001). Evolutionary synthesis of Bayesian networks for optimization. In M. Patel, V. Honavar, and K. Balakrishnan (Eds.), *Advances in Evolutionary Synthesis of Intelligent Agents* (pp. 429–455). Cambridge, MA: MIT Press.

Mühlenbein, H., and Paaß, G. (1996). From recombination of genes to the estimation of distributions. I. Binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature, PPSN IV*, Vol. 1141 of *Lecture Notes in Computer Science* (pp. 178–187). Berlin: Springer.

Neapolitan, R. E. (2003). *Learning Bayesian networks*. Upper Saddle River, NJ: Prentice Hall.

Ochoa, A., Höns, R., Soto, M. R., and Mühlenbein, H. (2003). A maximum entropy approach to sampling in EDA—The single connected case. In *Progress in Pattern Recognition, Speech and Image Analysis*, Vol. 2905 of *Lecture Notes in Computer Science* (pp. 683–690). Berlin: Springer.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm. Toward a new generation of evolutionary algorithms*. Berlin: Springer.

Pelikan, M., Goldberg, D. E., and Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20.

Pelikan, M., Sastry, K., and Goldberg, D. E. (2008). Sporadic model building for efficiency enhancement of the hierarchical BOA. *Genetic Programming and Evolvable Machines*, 9(1):53–84.

Rissanen, J. J. (1978). Modelling by shortest data description. *Automatica*, (14):465–471.

Santana, R. (2003). A Markov network based factorized distribution algorithm for optimization. In *Proceedings of the 14th European Conference on Machine Learning (ECML-PKDD 2003)*, Vol. 2837 of *Lecture Notes in Artificial Intelligence* (pp. 337–348). Berlin: Springer.

Santana, R. (2005). Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97.

Santana, R., Echegoyen, C., Mendiburu, A., Bielza, C., Lozano, J. A., Larrañaga, P., Armañanzas, R., and Shakya, S. (2009). MATEDA: A suite of EDA programs in Matlab. Tech. Rep. EHU-KZAA-I2/09, Department of Computer Science and Artificial Intelligence, University of the Basque Country.

Santana, R., Larrañaga, P., and Lozano, J. A. (2008). Protein folding in simplified models with estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 12(4):418–438.

Santana, R., Ochoa, A., and Soto, M. R. (2002). Solving problems with integer representation using a tree based factorized distribution algorithm. In *Electronic Proceedings of the First International NAISO Congress on Neuro Fuzzy Technologies*.

Sastry, K., and Orriols-Puig, A. (2007). Extended compact genetic algorithm in Matlab. IlliGAL Report 2007009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Soto, M. R. (2003). *A single connected factorized distribution algorithm and its cost of evaluation* (In Spanish). PhD thesis, University of Havana, Havana, Cuba.

Soto, M. R., Ochoa, A., Acid, S., and Campos, L. M. (1999). Bayesian evolutionary algorithms based on simplified models. In A. Ochoa, M. R. Soto, and R. Santana (Eds.), *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pp. 360–367.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312.

Yu, T.-L. (2006). *A matrix approach for finding extrema: Problems with modularity, hierarchy and overlap*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Yu, T.-L., Goldberg, D. E., and Chen, Y.-P. (2003). A genetic algorithm design inspired by organizational theory: A pilot study of a dependency structure matrix driven genetic algorithm. IlliGAL Report 2003007, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.