Learning factorizations in estimation of distribution algorithms using affinity propagation

Roberto Santana¹, Pedro Larrañaga², and Jose A. Lozano¹

¹Intelligent Systems Group
Department of Computer Science and Artificial Intelligence
University of the Basque Country
Paseo Manuel de Lardizábal 1, 20080. San Sebastian - Donostia, Spain
²Department of Artificial Intelligence, Technical University of Madrid,
28660 Boadilla del Monte, Madrid, Spain.
roberto.santana@ehu.es, pedro.larranaga@fi.upm.es, ja.lozano@ehu.es

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 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 non-binary 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.

1 Introduction

Estimation of distribution algorithms (EDAs) [25, 35, 39] 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, an important amount of computational time and memory are required to learn these models. Furthermore, the question of deciding which kind of model to use for a given problem is not solved. The best known alternative is the use of 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 [10, 33, 38].

One of the ways of adapting the structure of the model to the characteristics of the data is by means of score metrics [23] which evaluate the accuracy of the probabilistic model approximations and allow the search in the space of models. 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 score metrics (e.g the extended compact genetic algorithm (ECGA) [18], an EDA that uses the minimum description length metric (MDL) [42]) suffer from the same drawback. While some attempts [51, 36] have been made to design EDAs based on simplified probabilistic models that try to approximate higher order interactions using simpler models, these algorithms may also require at least second order independence tests to learn the structure of the models.

In this paper, we present an EDA whose model is constructed using the mutual information between pairs of variables and an affinity propagation algorithm [11, 12]. 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 with their corresponding exemplars. The method has been praised [29] 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 [36, 20, 21, 28] that combine the classical learning and sampling methods used by EDAs with different classes of message-passing and inference methods [37, 54]. This research trend leads to a new generation of hybrid EDAs that integrate different types of machine learning strategies. Our work is also related with recent efforts [41] to decrease the asymptotic complexity of model building in EDAs. We show that the use of the affinity propagation algorithm can reduce the time complexity of learning marginal product models.

The paper is organized as follows: In the next section, we introduce the notation and give a brief overview of EDAs. In Section 3, the class of EDAs that use marginal product models is discussed. Section 4 analyzes the question of learning marginal product models from short order marginal probabilities. In Section 5, message-passing algorithms are briefly reviewed and the general scheme of the affinity propagation algorithm is presented. Section 6 introduces an EDA that learns a marginal product model using affinity propagation. Details about the algorithm's implementation and its computational cost are examined. Section 7 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 8.

2 Estimation of distribution algorithms and factorizations

We begin by introducing some notation.

2.1 Notation

Let X be a discrete random variable. A value of X is denoted x. $\mathbf{X} = (X_1, \dots, X_n)$ will denote a vector of random variables. We will use $\mathbf{x} = (x_1, \dots, x_n)$ to denote an assignment to the variables. S will denote a set of indices in $\{1, \dots, n\}$, 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 $X_j = x_j$.

2.2 Factorizations and EDAs

Similarly to a genetic algorithm (GA) [15], an EDA begins by randomly generating a sample of points. These points are evaluated using the objective function, and a subset of points is selected based on this evaluation. Hence, points with better function values have a higher chance to be selected. Then a probabilistic model of the selected solutions is built, and a new set of points is sampled from the model. The process is iterated until an optimal solution has been found or another termination criterion is fulfilled.

There are different variants of EDAs. They mainly differ in the type of probabilistic model used and the methods used to learn and sample these models. A general pseudocode of EDAs is shown in Algorithm 1. EDAs have been successfully applied to the solution of a variety of combinatorial and continuous optimization problems [25, 27, 40].

Algorithm 1: Main scheme of the EDA approach

```
1 D_0 \leftarrow Generate M individuals randomly and evaluate them

2 t=1

3 \mathbf{do} {

4 D_{t-1}^s \leftarrow Select N \leq M individuals from D_{t-1} according to a selection method

5 p_t(\mathbf{x}) = p(\mathbf{x} \mid D_{t-1}^s) \leftarrow Estimate the joint probability of selected individuals

6 D_t \leftarrow Sample M individuals (the new population) from p_t(\mathbf{x})

and evaluate them

7 } until A stop criterion is met
```

2.3 Probabilistic modeling in EDAs

One of the goals of probability modeling in EDAs is to obtain a condensed, accurate representation, of the distribution 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.

The structure of the factorizations can be partially or exactly known a priori, determined from the analysis of previous information about the problem, or learned from the data during the search process. In every case, the complexity of the factorization is related to the size of the factors involved. Different criteria can be used to classify factorizations. Among them are:

- The type of (in)dependence relationships they represent.
- The class of probabilistic models they are derived from.
- The topological relationships between the subgraphs (generally, the cliques) associated to the factors in the graphical model.

Regarding the type of independence relationships, we call marginal product factorizations to those that only represent marginal independence relationships between sets of variables. In these factorizations, no factor appears in the denominator of the factorization. In contrast, conditional independence factorizations can represent marginal and conditional independence between the variables. In these factorizations, factors will appear in the numerator and in the denominator.

Example 1 Equations (1) and (2) respectively show a marginal product, $p^a(\mathbf{x})$, and a conditional, $p^b(\mathbf{x})$, factorization of a probability distribution $p(\mathbf{x})$, with $\mathbf{X} = (X_1, \dots, X_7)$.

$$p^{a}(\mathbf{x}) = p(x_1, x_2)p(x_3, x_4)p(x_5, x_6)p(x_7)$$
(1)

$$p^{b}(\mathbf{x}) = \frac{p(x_1, x_2, x_4)p(x_1, x_3, x_5)p(x_1, x_4, x_5)p(x_6, x_7)}{p(x_1, x_4)p(x_1, x_5)}$$
(2)

Usually, factorizations are associated to the graphical models they are derived from. For instance, there are chain-shaped, tree-shaped, Bayesian-network based factorizations, etc.

Example 2 The factorization determined by a Bayesian network is as follows:

$$p(x_1, \dots, x_n) = \prod_{i=1}^n p(x_i | \mathbf{x}_{pa_i}) = \prod_{i=1}^n \frac{p(x_i, \mathbf{x}_{pa_i})}{p(\mathbf{x}_{pa_i})}$$
(3)

where \mathbf{X}_{pa_i} is the set of parents of variable X_i .

Finally, and regarding the relationships that hold between the factors and which can be exposed using graphical representations, factorizations that satisfy the running intersection property (RIP) are called *valid* [34]. Those for which the RIP does not hold are considered invalid and may be further classified [43] in 'ordered' and 'messy'. We say that a factorization is "ordered" when an ordering of all the maximal factors in the factorization exists such that for every factor there exists at least one variable that is not contained in the previous factors in the ordering. When it is impossible to find such an ordering, we say that the factorization is "messy". The satisfaction of the running intersection property determines that every valid factorization is an ordered one.

Most of EDAs employ valid factorizations. In these cases, the structure of a factorization can be directly recovered from a chordal graph as done in the factorized distribution algorithm (FDA) [34] or determined by a Bayesian network learned from the data [10, 38, 33, 51, 36].

FDA can work with invalid factorizations but in this case the convergence properties proved for when valid factorizations are employed do not hold [34]. On the other hand, EDAs that work with messy factorizations [43, 44] need to employ Gibbs sampling techniques [13] or message-passing schemes [37, 54] in order to sample new solutions.

3 EDAs based on marginal product models

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) [35] and the extended compact GA (ECGA) [18]. We devote some time to analyze these algorithms and the learning procedures they employ.

3.1 UMDA

UMDA uses a probabilistic model where all variables are considered independent. The probability associated to each solution is computed as a product of univariate marginal probabilities. The probabilistic model of UMDA, which is the simplest MPM of those used by EDAs, is described by Equation (4).

$$p_{UMDA}(\mathbf{x}) = \prod_{i=1}^{n} p(x_i) \tag{4}$$

Since in this case variables are independent, the structure of the model is fixed and only the univariate marginal probabilities are learned from the data. Theoretical results derived for the UMDA [32] expose the relationship between this kind of EDAs and GAs that use uniform and one-point crossover. Other algorithms that use this type of models are the population based incremental learning (PBIL) [2] and the compact GA (cGA) [17]. The two latter algorithms update the univariate probabilities using global information about the search, i.e. information collected from previous generations.

A natural generalization of the univariate models are MPMs that comprise higher order factors [3, 48]. This approach has been followed in situations where interactions between the variables of the problem are known a priori and the size of the factors allows one to define computationally feasible factorizations. MPMs can also be learned from data, as is the case of the EDA presented in the next section.

UMDA and other EDAs based on univariate models have been mainly investigated for binary problems [2, 35, 16, 50, 56]. However, successful applications to discrete problems of higher cardinality have been also reported [45].

3.2 ECGA

ECGA uses a factorization of the probability where variables are separated in non-overlapping factors. At each generation, these factors are found by minimizing the MDL metric of the model representing the selected solutions. In the model, each factor is assumed to be independent of the rest. The pseudocode of ECGA is shown in Algorithm 2.

Let χ be the alphabet cardinality of the variables¹, m the number of dependency sets of the model, 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 $j \in \{1, ..., \chi^{k_i}\}$ for the definition set S_i .

In ECGA [18], learning the MPM in every generation is approached as a constrained optimization problem:

¹Although 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 cardinalities

- 1 Set $t \Leftarrow 0$. Generate M points randomly and evaluate them
- 2 do {
- 3 Undergo tournament selection at a rate r
- Find a MPM by minimization of a MDL metric using a greedy search
- 5 **if** the model has converged, stop
- 6 Generate M new points using the learned model
- 7 $t \Leftarrow t + 1$
- 8 } until A termination criterion is met

Minimize
$$C_m + C_p$$
 Subject to
$$\chi^{k_i} \leq N \, \forall i \in \{1, \dots, m\}$$
 (5)

where C_m represents the model complexity and is given by

$$C_m = \log_{\chi}(N+1) \sum_{i=1}^{m} (\chi^{k_i} - 1)$$
 (6)

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

$$C_p = \sum_{i=1}^{m} \sum_{j=1}^{\chi^{k_i}} N_{ij} \log_{\chi} \left(\frac{N}{N_{ij}} \right)$$

$$\tag{7}$$

The greedy search heuristic used by ECGA 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 of the greedy algorithm, 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.

Once the factors have been learned, the parameters are estimated using the maximum likelihood approach and new solutions are generated by independently sampling each factor. ECGA can use different stop conditions. A maximum number of generations or a predefined value of the fitness function achieved by the algorithm are two commonly used criteria.

4 Learning MPMs from short order marginal probabilities

There are two main approaches for learning probabilistic models from data [24]: learning based on detecting conditional independencies by means of independence tests, and score+search algorithms. Hybrid algorithms that combine these approaches have also been proposed [52].

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.

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 pair-wise interactions and will combine them but without the need to compute higher-order tables to evaluate the quality of the model. In EDAs, we identify three ways in which this approach has been followed:

- Use of iterative proportional fitting (IPF) [7] which is a method to compute higher order marginal approximations. IPF allows to find a maximum entropy distribution given a set of constraints (in this case the constraints correspond to the known marginal distributions). In [36], a junction-tree based implementation of IPF is used to approximate higher-order marginals in the polytree approximation distribution algorithm (PADA) [52]. 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 [36] and [20] showed remarkable improvements in the version of PADA that added IPF with respect to UMDA and simpler PADA variants.
- Use of a heuristic algorithm [5] 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 χ² statistics). This approach combines the use of (up to order one) independence
 tests with the clustering method. In [43], it was used in the context of EDAs to learn junction
 trees and Kikuchi approximations. For the problems tested, the algorithm exhibited similar
 results to EDAs that use Bayesian network based models.
- Use of a dependency structure matrix (DSM) combined with a MDL technique to cluster the mutual information values between every pair of variables. The obtained clusters are used as the factors of the factorization. A DSM is a matrix that contains information about the pairwise interactions between the components of a system. In [55], a MDL-based DSM clustering metric is introduced to evaluate clustering arrangements. A greedy optimization method is used to search in the space of possible clusterings. The method is incorporated into an EDA (DSMGA) which learns MPMs from mutual information. Two variants of the algorithm, DSMGA+ and DSMGA++ are respectively proposed to deal with hierarchical and overlapping problems.

While the first two algorithms incorporate some sort of structural learning by means of independence tests, the DSMGA method exclusively rests on the use of a clustering of the mutual information. Similarly to the ECGA, the MDL minimization approach applied to the clustering of mutual information allows to automatically determine the number of clusters. On the other hand, the quality of the model found will heavily depend on the behavior of the greedy algorithm. Later, we propose a learning method that recovers a clustering of the matrix of mutual information by means of an affinity-propagation-based clustering algorithm.

4.1 Clustering of the mutual information as an optimization problem

The problem of finding an accurate partition of the mutual information matrix can be posed as an optimization problem.

4.1.1 Clustering of a similarity matrix

In the general clustering problem, $\mathbf{y}^1, \dots, \mathbf{y}^Q$ will represent the Q data points. $\mathbf{c} = c^1, \dots, c^Q$ will represent a set of Q hidden labels corresponding to the Q data points. Each value c^i ($1 \le c^i \le Q$) indicates the cluster each data point \mathbf{y}^i belongs to. 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. We want to maximize the similarity between points that belong to the same cluster. The function to be maximized is:

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

$$\tag{8}$$

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 is unknown a priori but, since they do not overlap, its maximal number is Q.

In our particular instantiation of the clustering problem, we are interested in clustering variables (i.e. Q = n), 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 $i, j \in \{1, ..., n\}$. The mutual information is a symmetric similarity measure.

Function (8) measures the sum of the average distances between pairs of points in each cluster. The problem representation is flexible, allowing to represent clusterings with a different number of clusters. However, it has two main drawbacks. Not every assignment of c^i is valid. For instance, if $c^i = j$ implying that \mathbf{y}^i belongs to the cluster where \mathbf{y}^j is, then $c^j = j$ should be fulfilled, implying that \mathbf{y}^j belongs to the cluster it serves to define. The other drawback is that the representation is highly redundant.

Accomplishing the maximization of expression (8) is difficult because we have to determine at the same time the number of clusters and the cluster membership of each point. Therefore, the problem is more complex than the k-partitioning problem [1] for which the number of clusters is known in advance. We deal with the clustering problem using a relaxation approach: 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. Secondly, a local optimization method is applied to the suboptimal solution to find a better (possibly optimal) clustering solution.

4.1.2 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.

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. To avoid invalid configurations, constraints have to be imposed to the solutions. The problem is then posed as the problem of maximizing the net similarity \mathcal{S} [12]:

$$S(\mathbf{c}) = E(\mathbf{c}) + \sum_{l=1}^{n} \delta_l(\mathbf{c}) = \sum_{i=1}^{n} s(\mathbf{y}^i, \mathbf{y}^{c^i}) + \sum_{l=1}^{n} \delta_l(\mathbf{c})$$
(9)

where

$$\delta_l(\mathbf{c}) = \begin{cases} -\infty & \text{if } c^l \neq l \text{ but } \exists i : c^i = l \\ 0 & \text{otherwise} \end{cases}$$
 (10)

In comparison with Equation (8), the function represented by Equation (9) automatically specifies the membership of each point to a cluster. The constraint imposes that there will be one exemplar for each cluster. To address the optimization problem, we employ an optimization method based on a recursive application of a message-passing algorithm.

5 Message-passing algorithms and affinity propagation

Message-passing algorithms use the passing of messages to iteratively update marginal distributions until a convergence criterion is satisfied. This class of methods includes belief propagation (BP) algorithms [37, 54], survey and warning propagation algorithms [4, 19, 30] and affinity propagation algorithms [11, 12, 26].

The goal of BP is inference in different domains. There are two types of BP methods: sumpropagation and max-propagation. In the first case, the objective is to obtain a set of consistent marginal probabilities without adding the probabilities corresponding to all the (possibly exponential number of) necessary configurations. In the latter case, the goal is to obtain the most probable configuration of the model. These algorithms have been directly applied to optimization [53] and in every case the optimization approach has been posed as an inference problem. In most of cases maximum-propagation has been applied. Survey and warning message-passing algorithms have been conceived for the solution of constrained combinatorial problems, particularly the satisfiability (SAT) problem. They can be applied in more general problem decomposition and solving schemes called decimation schemes [4]. Finally, affinity propagation algorithms have been proposed to solve general clustering problems where an important aim is to obtain a point that serves to describe each cluster, i.e. an exemplar.

5.1 Affinity propagation

The explanation of affinity propagation presented in this section has been done following [12], 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, find clusters of similar points, and for each cluster gives a representative example or *exemplar*.

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 exemplar. The algorithm works by exchanging messages between the points until a stop condition is satisfied.

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:

$$r(i,k) \leftarrow s(\mathbf{y}^i, \mathbf{y}^k) - \max_{k'|k' \neq k} \{a(i,k') + s(\mathbf{y}^i, \mathbf{y}^{k'})\}$$
(11)

The responsibility update shown in Equation (11) allows all the candidate exemplars 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:

$$a(i,k) \leftarrow min \left\{ 0, r(k,k) + \sum_{i'|i'\notin\{i,k\}} max\{0, r(i',k)\} \right\}$$
 (12)

In the availability update shown in Equation (12) 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. 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:

$$a(k,k) = \sum_{i'|i'\neq k} \max\{0, r(i',k)\}$$
(13)

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 (11), (12) and (13) 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, after that changes in the messages fall below a threshold, or after the local decisions stay constant for some number of iterations.

Similarly 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 < λ < 1). The pseudocode of affinity propagation algorithm is shown in Algorithm 3.

When affinity propagation converges, it gives an approximate solution to the maximization of Equation (9), but as discussed in Section 4.1.1, our final goal is to obtain a solution to Equation (8). 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

Algorithm 3: Affinity propagation

```
Initialize availabilities a(i, k) to zero ∀i, k
do {
Update, using Equation (11), all the responsibilities given the availabilities
Update, using Equation (12), all the availabilities given the responsibilities
Combine availabilities and responsibilities to obtain the exemplar decisions
autil Termination criterion is met
```

that reassigns the points starting from the initial clusters found by affinity. Its pseudocode is shown in Algorithm 4.

Algorithm 4: Partitioning improvement algorithm

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

Algorithm 4 receives as a parameter the number of iterations that the reassignment procedure should be applied.

6 An EDA based on affinity propagation

In this section, we present an EDA that learns a 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 message-passing algorithm for structural learning. The rationale of the method is to group the variables in non-overlapping 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 3 that allows 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 constraints 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 5. 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 . Since the algorithm is recursive, a maximum number of recursive calls depth and the current number of calls, ncalls, are also passed as parameters. The algorithm updates the list of factors L.

Algorithm 5: Affinity propagation with constrained clusters

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

Algorithm 5 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 and less 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 can not 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.

For each of the remaining components (non-homogeneous, with size above 3) V_i , the algorithm computes the reduced matrix of mutual information MI_i and calls the affinity propagation Algorithm 3 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 equal or less than δ , the feasible clusters are inserted in L and the rest of the variables, where they exist, are grouped together to recursively call Algorithm 5 again. If the algorithm converged and none of the clusters found has size equal or less than δ , then Algorithm 5 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 5 is the learning component of our AffEDA, whose pseudocode is shown in Algorithm 6. AffEDA may use different types of selection methods.

Algorithm 6: AffEDA

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

6.1 Implementation of the algorithm

Usually, implementation details do not receive much attention in the analysis of the results achieved by EDAs. AffEDA implementation has been assembled from available implementations of ECGA and of the affinity propagation algorithm. The availability and usability of these implementations were important for the rapid implementation of our method. They may also allow the introduction and validation of other variations of the algorithm. Therefore, we spend some time to briefly discuss the main characteristics of these implementations. 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, i.e. problems with discrete representation. The source code² in Matlab is an extension of the original binary-coded ECGA. The programs are documented in [49]. A detail of the implementation

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

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.
- To replace 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 propagation³ and makes recursive calls to this program.

Also the C++ source code⁴ 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 [6]. The changes made in the C++ source code follow the same rationale to those explained for the Matlab implementation of AffEDA. The modified C++ version of the affinity propagation algorithm uses the original C++ implementation of affinity propagation⁵.

6.2 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 5 used by AffEDA.

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

6.2.1 Initialization

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

6.2.2 Evaluation

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

6.2.3 Selection

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.

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

⁴http://www.illigal.uiuc.edu/pub/src/ECGA/chiECGA.tgz

⁵http://www.psi.toronto.edu/affinitypropagation/apcluster.txt

6.2.4 Probabilistic model learning algorithms

The cost of this step can be further divided into the cost of structural and parametrical learning.

The structural learning step includes the computation of the matrix of mutual information and the application of Algorithm 5.

- The calculation of the mutual information has complexity $O(n^2 r_{MAX}^2)$.
- 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. Therefore, the complexity of Algorithm 3 is $O(n^2)$.
- The cost of the partitioning improvement algorithm (Algorithm 4) used by Algorithm 5 depends on the maximal number of iterations (maxiter) and the number of clusters found. Assuming that the number of clusters is αn $(\alpha \in \{\frac{1}{n}, \dots, \frac{n}{n}\})$, the complexity of the algorithm is $O(n^2 maxiter)$.
- As previously stated, the complexity of Algorithm 5 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 $O(n^2 depth)$ where depth is a maximum number of recursive calls.

The total cost of the structural learning step can be estimated as $O(n^2 depth \ maxiter)$.

The cost of parametrical learning is the cost of computing the marginal probabilities used by AffEDA, i.e. the m marginal probability tables from the selected population, it has complexity O(mN).

6.2.5 Sampling

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

6.2.6 Total computational costs of the algorithm

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

7 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 running times of the algorithms for functions with different domains of difficulty. The scalability of AffEDA is also investigated. In all the experiments, we use the C++ implementations of ECGA and AffEDA discussed in Section 6.1. We start by introducing the testbed functions and problems used in our comparisons and the rationale behind our choice.

7.1 Function benchmark

In the function benchmark used to test the algorithms we include a number of additively decomposable deceptive functions and a non-additively decomposable function corresponding to a simplified protein model.

7.1.1 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 in a simpler way 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.

 $f_{deceptivek}(\mathbf{x})$ [34] is a deceptive function formed by the additive sum of deceptive sub-functions 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 sub-functions:

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

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

$$(14)$$

The difference between the optimum fitness value and the closest sub-optimum value (usually called the signal size) has an influence on the difficulty of the function for the evolutionary algorithm. A smaller signal size may make it more difficult to distinguish between optimal and suboptimal solutions during the search, requiring higher population size. Therefore, in order to investigate the effect of the signal size in the behavior of AffEDA, we include in our testbed the $f_{deceptive}(\mathbf{x})$ function [14].

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

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

$$(15)$$

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

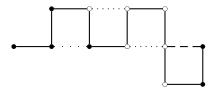


Figure 1: One possible configuration of sequence HHHPHPPPPHHH in the HP model. There is one HH (represented by a dotted line with wide spaces), one HP (represented by a dashed line) and two PP (represented by dotted lines) contacts.

$$f_{deceptivek}^{c}(\mathbf{x}) = \sum_{i=1}^{\frac{n}{k}} F_{dec}(x_{k\cdot(i-1)+1} + x_{k\cdot(i-1)+2} + \dots + x_{k\cdot i}, k, c)$$
 (16)

$$F_{dec}(x_1, \dots, x_k, k, c) = \begin{cases} k \cdot (c-1), & for \sum_{i=1}^k x_i = k \cdot (c-1) \\ k \cdot (c-1) - \sum_{i=1}^k x_i - 1 & otherwise \end{cases}$$
(17)

The general deceptive function $f_{deceptivek}^c(\mathbf{x})$ of order k [47] 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, i.e. $x_i \in \{0, \ldots, c-1\}$. This function is a generalization of the binary $f_{deceptivek}(\mathbf{x})$ to variables with non-binary values.

7.2 The HP protein model

Under specific conditions, a protein sequence folds into a native 3-d structure. The problem of determining the protein native structure from its sequence is known as the protein structure prediction problem. 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 model called the hydrophobic-polar (HP) model [8].

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 HHHPHPPPPHHH 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 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. The codification used is called relative encoding, and has been experimentally compared to a different encoding called absolute encoding in [22], showing better results. In the experiments presented in this section we use 2-d and 3-d 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 HP instances used in our experiments, and shown in Table 1, have previously been used in several papers (see [46] and cites therein). 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 [46] and therefore we are interested in studying the way ECGA and AffEDA behave on it.

Table 1:	HP	instances	used	in	the	experiments.
----------	----	-----------	------	----	-----	--------------

size	$H^2(\mathbf{x}^*)$	$H^3(\mathbf{x}^*)$	sequence
·			
60	-36		$PPH^{3}PH^{8}P^{3}H^{10}PHP^{3}H^{12}P^{4}H^{6}PHHPHP$
100	-48		$P^6HPH^2P^5H^3PH^5PH^2P^4H^2P^2H^2PH^5PH^{10}PH^2PH^7$
			$P^{11}H^7P^2HPH^3P^6HPH$
100	-50		$P^{3}H^{2}P^{2}H^{4}P^{2}H^{3}PH^{2}PH^{2}PH^{4}P^{8}H^{6}P^{2}H^{6}P^{9}HPH^{2}$
			$PH^{11}P^2H^3PH^2PHP^2HPH^3P^6H^3$
103		-54	$P^{2}H^{2}P^{5}H^{2}P^{2}H^{2}PHP^{2}HP^{7}HP^{3}H^{2}PH^{2}P^{6}HP^{2}HPH$
		<u> </u>	$P^{2}HP^{5}H^{3}P^{4}H^{2}PH^{2}P^{5}H^{2}P^{4}H^{4}PHP^{8}H^{5}P^{2}HP^{2}$
124		-71	$P^{3}H^{3}PHP^{4}HP^{5}H^{2}P^{4}H^{2}P^{2}H^{2}P^{4}HP^{4}HP^{2}HP^{2}H^{2}P^{3}$
121		, ,	$H^2PHPH^3P^4H^3P^6H^2P^2HPHP^2HP^7$
			11 1 11 11 1 11 1 11 1 11 11 11 11 11 1
			$HP^2H^3P^4HP^3H^5P^4H^2PH^4$
	60 100 100	100 —48 100 —50 103	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$

7.3 Numerical results

7.3.1 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 [18], where a systematic study of ECGA is done, showing its advantage over simple GAs. The number of deceptive subproblems considered in [18] 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 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.

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. 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 clearly the best contender for all the functions. However, both algorithms have a similar scalability. Another observation is the way in which the complexity of the problems is increased with k. The increase in this complexity is noticeable. These results are in agreement with what is expected, i.e. the number of evaluations grows exponentially with k.

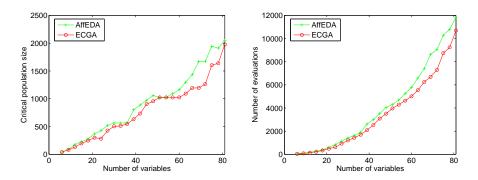


Figure 2: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive3}(\mathbf{x})$.

We also conduct scalability experiments for function $f_{deceptive}(\mathbf{x})$. The number of deceptive problems considered in [18] ranges varies 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). This means that AffEDA is more sensitive to the signal size than ECGA.

7.3.2 Computational cost of the algorithms

We now compare the running times of AffEDA and ECGA. We consider that a run is successful when a solution is found for which at least all but one subproblem have been solved. Our

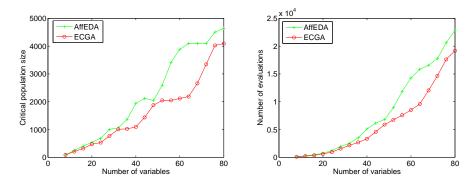


Figure 3: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive4}(\mathbf{x})$.

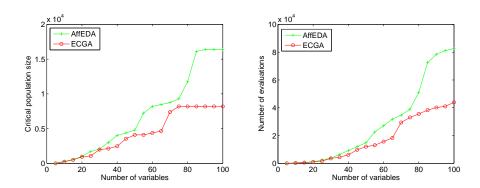


Figure 4: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive5}(\mathbf{x})$.

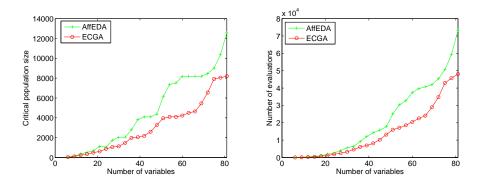


Figure 5: Critical population size and average number of evaluations of AffEDA and ECGA for function $f_{deceptive}(\mathbf{x})$.

experiment has two goals: To determine the average running time of a successful run for both algorithms, and to compute the average expected time to find l successful runs, with $1 \le l \le 30$.

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 respectively n=81, n=80 and n=100. The average critical population sizes computed for ECGA in the experiments shown in Section 7.3.1 were used as a base to determine the population size used in these experiments. Since the replacement strategy needs the population size to be a multiple of the tournament size, we found the population size a multiple of 16 and closest to the critical population size. The population sizes used are shown in Table 2.

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.

F	n	M	AffE	EDA	ECGA		
			$\mu(t)$	$\sigma(t)$	$\mu(t)$	$\sigma(t)$	
$f_{deceptive3}$	81	1984	192.90	16.53	713.53	47.34	
$f_{deceptive 4}$	80	4096	573.70	28.50	1645.00	33.01	
$f_{deceptive 5}$	100	8192	6352.37	1844.50	9625.27	3478.98	

After every successful outcome, we compute the total time elapsed. The time consumed by unsuccessful outcomes is included in this sum. 30 experiments are conducted and from them the average expected time (in seconds) to find the l^{th} successful outcome is computed. The average time needed to find the optimum is computed from the 900 successful runs. Table 2 shows the mean and standard deviation needed by AffEDA and ECGA to attain 30 successful runs of the algorithm. Figure 6 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 $f_{deceptive5}(\mathbf{x})$ (Figure 6 c)). However, in this latter case the difference in the time consumed by both algorithms is reduced.

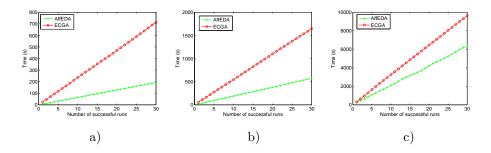


Figure 6: Elapsed time before finding l, not necessarily consecutive, successful runs of AffEDA and ECGA for functions: a) $f_{deceptive3}(\mathbf{x})$, b) $f_{deceptive4}(\mathbf{x})$ and c) $f_{deceptive5}(\mathbf{x})$.

7.3.3 Influence of the variables' cardinality

We investigate the performance of AffEDA and ECGA for function $f_{deceptivek}^c(\mathbf{x})$, $k \in \{3,4,5\}$, $c \in \{3,4,5,6\}$. 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. 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 251504. 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 $f_{deceptivek}^c(\mathbf{x})$ functions with different values of k and c. Results are average from 900 experiments.

F	n		AffEDA		ECGA			
		M	e	t	M	e	t	
$f_{deceptive3}^3$	30	4096	11919	1.50	2389	7256	1.52	
$f_{deceptive3}^4$		10103	37100	4.60	8465	31594	8.41	
$f_{deceptive3}^5$		32768	130417	15.45	32768	130671	35.08	
$f_{deceptive3}^6$		65536	296077	34.46	131072	563901	153.60	
$f_{deceptive4}^3$	32	10103	29992	4.23	8738	26351	12.14	
$f_{deceptive4}^4$		65536	233454	31.21	128887	436032	197.56	
$f_{deceptive5}^3$	30	37137	96192	17.78	61167	154519	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 $f_{deceptive3}^5(\mathbf{x})$, $f_{deceptive3}^4(\mathbf{x})$, $f_{deceptive4}^4(\mathbf{x})$ and $f_{deceptive5}^3(\mathbf{x})$ is smaller than those needed by ECGA. 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. A fact that illustrates the complexity of the functions under analysis is that none of the algorithms were able to reach 30 consecutive successful runs for five of the functions tested using a population size as high as 251504. Notice 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.

7.3.4 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: non-binary, 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. Results for ECGA are included for comparison.

Table 4: Results of AffEDA and ECGA								
inst.	d	M	T	AffEDA		E	CGA	
				best	$\mu(e)$	best	$\mu(e)$	
s7	2	32768	64	-35	-32.20	-35	-33.32	
	2	32768	256	-35	-33.84	-35	-33.98	
s10	2	32768	64	-42	-39.96	-43	-39.58	
	2	32768	256	-43	-40.20	-42	-40.10	
s11	2	32768	64	-46	-43.58	-46	-42.82	
	2	32768	256	-47	-44.14	-46	-44.04	
s14	3	65536	64	-48	-43.56	-49	-44.36	
	3	65536	256	-50	-44.02	-47	-44.32	
s15	3	65536	64	-59	-55.14	-60	-55.00	
	3	65536	256	-60	-55.44	-62	-55.94	

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 the tournament size used. 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. It has to be pointed out though, that for the 2-dimensional lattice problems the algorithms were not able to achieve the best fitness solutions found by EDAs that use mixture and Markov probabilistic models [46] which allow overlapping sets of variables.

We conduct an additional experiment to compare the efficiency of AffEDA and ECGA for the HP problem. Both algorithms are allowed to run in computers with the same architecture for 24 hours, executing as many runs of each EDA as possible. Both algorithms use tournament size 256 and a maximum of 200 generations. After the time period has elapsed, we compute the number of completed executions for each algorithm and evaluate 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.

Table 5: Results of AffEDA and ECGA M**ECGA** inst.d**AffEDA** bestntimesrunsbestntimesruns2 32768 -355 s7best17 23 -352 32768 best2-3413 -3410 2 s1032768 best116 -421 -421 2 32768 -417 -414 best22 -47-471 32768 best116 1 10 s112 2 32768best2-461 -463 65536 best1-451 -45s142 3 65536 -44-441 best21 -62s153 65536best1-603 1 3 65536-581 -571 best2

The quality of the best solution was similar in each case except for instance S15 for which ECGA achieves a better result.

8 Conclusions and future work

In this paper we have introduced AffEDA, an EDA based on the use of MPMs. The distinguished feature of our algorithm is that the factorization is efficiently learned from the data using affinity propagation. We have empirically shown that the algorithm is much faster than ECGA for all the problems considered. Our experiments have also shown that AffEDA is more efficient than ECGA to solve deceptive non-binary 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.

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 [36, 20] or to sample a set of solutions by means of the k most probable configurations found applying max-propagation [21, 28, 31, 51].

We notice, that while the original affinity propagation algorithm [12] does not impose constraints on the size of the clusters found, our recursive version allows the incorporation of this sort of constraints. Our modified algorithm could be also 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 [41]) places emphasisis 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 more accurate learning algorithms does not always imply better convergence results, as the case of problems with high cardinality illustrate. Another contribution of this

paper lies in the investigation of ECGA for the class of non-binary 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) [10], 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 [9]. This investigation has been carried out for small problems, for which exact learning is computationally feasible. To conduct 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 [26] has produced a more robust version of the algorithm, less dependent to 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 to go beyond the class of MPMs analyzed in this paper.
- 3. 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.

Acknowledgment

This work has been partially supported by the Etortek, Saiotek and Research Groups 2007-2012 (IT-242-07) programs (Basque Government), TIN2005-03824 and Consolider Ingenio 2010 - CSD2007-00018 projects (Spanish Ministry of Education and Science) 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

- [1] L. Babel, H. Kellerer, and V. Kotov. The k-partitioning problem. *Mathematical Methods of Operations Research*, 47(1):59–82, 1998.
- [2] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.
- [3] P. A. Bosman and D. Thierens. Crossing the road to efficient ideas for permutation problems. In L. Spector, E. Goodman, A. Wu, W. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo,

- S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2001*, pages 219–226, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
- [4] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.
- [5] C. Bron and J. Kerbosch. Algorithm 457—finding all cliques of an undirected graph. *Communications of the ACM*, 16(6):575–577, 1973.
- [6] L. de la Ossa, K. Sastry, and F. G. Lobo. χ-ary extended compact genetic algorithm in C++. IlliGAL Report 2006013, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2006.
- [7] W. E. Deming and F. F. Stephan. 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, 1940.
- [8] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.
- [9] C. Echegoyen, J. A. Lozano, R. Santana, and P. Larrañaga. Exact Bayesian network learning in estimation of distribution algorithms. In *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pages 1051–1058. IEEE Press, 2007.
- [10] R. Etxeberria and P. Larrañaga. Global optimization using Bayesian networks. In A. Ochoa, M. R. Soto, and R. Santana, editors, *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 151–173, Havana, Cuba, 1999.
- [11] B. J. Frey and D. Dueck. Mixture modeling by affinity propagation. In *Proceedings of the 2005 Conference Advances in Neural Information Processing Systems 18, NIPS*, pages 379–386. MIT Press, 2006.
- [12] B. J. Frey and D. Dueck. Clustering by passing messages between data points. Science, 315:972–976, 2007.
- [13] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.
- [14] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In L. Davis, editor, Genetic Algorithms and Simulated Annealing, pages 74–88. Pitman Publishing, London, UK, 1987.
- [15] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, 1989.
- [16] C. González, J. A. Lozano, and P. Larrañaga. Analyzing the PBIL algorithm by means of discrete dynamical systems. *Complex Systems*, 12(4):465–479, 2001.

- [17] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. IEEE Transactions on Evolutionary Computation, 3(4):287–297, 1999.
- [18] G. R. Harik, F. G. Lobo, and K. Sastry. Linkage learning via probabilistic modeling in the ECGA. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pages 39–62. Springer-Verlag, 2006.
- [19] A. K. Hartmann and M. Weigt. Phase Transitions in Combinatorial Optimization Problems: Basics, Algorithms and Statistical Mechanics. John Wiley & Sons, 2005.
- [20] R. Höns. Estimation of Distribution Algorithms and Minimum Relative Entropy. PhD thesis, University of Bonn, Bonn, Germany, 2006.
- [21] R. Höns, R. Santana, P. Larrañaga, and J. A. Lozano. Optimization by max-propagation using Kikuchi approximations. Technical Report EHU-KZAA-IK-2/07, Department of Computer Science and Artificial Intelligence, University of the Basque Country, November 2007.
- [22] N. Krasnogor, W. E. Hart, J. Smith, and D. A. Pelta. 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, editors, *Proceedings of the Genetic and Evolutionary Com*putation Conference, volume 2, pages 1596–1601, Orlando, Florida, USA, 1999. Morgan Kaufmann.
- [23] P. Larrañaga. Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, chapter An introduction to probabilistic graphical models, pages 25–54. Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.
- [24] P. Larrañaga. Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, chapter A review on estimation of distribution algorithms, pages 55–98. Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.
- [25] P. Larrañaga and J. A. Lozano, editors. Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.
- [26] M. Leone, Sumedha, and M. Weigt. Clustering by soft-constraint affinity propagation: Applications to gene-expression data. *Bioinformatics*, 23(20):2708–2715, 2007.
- [27] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Springer-Verlag, 2006.
- [28] A. Mendiburu, R. Santana, and J. A. Lozano. Introducing belief propagation in estimation of distribution algorithms: A parallel framework. Technical Report EHU-KAT-IK-11/07, Department of Computer Science and Artificial Intelligence, University of the Basque Country, October 2007.
- [29] M. Mézard. Where are the good exemplars? Science, 315:949–951, 2007.
- [30] M. Mézard, G. Parisi, and R. Zechina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812–812, 2002.

- [31] H. Mühlenbein and R. Höns. The factorized distributions and the minimum relative entropy principle. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pages 11–38. Springer-Verlag, 2006.
- [32] H. Mühlenbein and T. Mahnig. Evolutionary computation and beyond. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, Foundations of Real-World Intelligence, pages 123–188. CSLI Publications, Stanford, California, 2001.
- [33] H. Mühlenbein and T. Mahnig. Evolutionary synthesis of Bayesian networks for optimization. In M. Patel, V. Honavar, and K. Balakrishnan, editors, *Advances in Evolutionary Synthesis of Intelligent Agents*, pages 429–455. MIT Press, Cambridge, Mass., 2001.
- [34] H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.
- [35] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, Parallel Problem Solving from Nature - PPSN IV, volume 1141 of Lectures Notes in Computer Science, pages 178–187, Berlin, 1996. Springer Verlag.
- [36] A. Ochoa, R. Höns, M. R. Soto, and H. Mühlenbein. A maximum entropy approach to sampling in EDA- the single connected case. In *Progress in Pattern Recognition, Speech and Image Analysis*, volume 2905 of *Lectures Notes in Computer Science*, pages 683–690. Springer Verlag, 2003.
- [37] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo, California, 1988.
- [38] M. Pelikan. Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms. Studies in Fuzziness and Soft Computing. Springer, 2005.
- [39] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.
- [40] M. Pelikan, K. Sastry, and E. Cantú-Paz, editors. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*. Studies in Computational Intelligence. Springer, 2006.
- [41] M. Pelikan, K. Sastry, and D. E. Goldberg. Sporadic model building for efficiency enhancement of the hierarchical BOA. Genetic Programming and Evolvable Machines, 2008. To appear.
- [42] J. J. Rissanen. Modelling by shortest data description. Automatica, (14):465–471, 1978.
- [43] R. Santana. Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97, 2005.

- [44] R. Santana, P. Larrañaga, and J. A. Lozano. Mixtures of Kikuchi approximations. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, Proceedings of the 17th European Conference on Machine Learning: ECML 2006, volume 4212 of Lecture Notes in Artificial Intelligence, pages 365–376. Springer Verlag, 2006.
- [45] R. Santana, P. Larrañaga, and J. A. Lozano. Side chain placement using estimation of distribution algorithms. *Artificial Intelligence in Medicine*, 39(1):49–63, 2007.
- [46] R. Santana, P. Larrañaga, and J. A. Lozano. Protein folding in simplified models with estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 2008. To appear.
- [47] R. Santana, A. Ochoa, and M. R. Soto. 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*. NAISO Academic Press, 2002.
- [48] R. Santana, F. B. Pereira, E. Costa, A. Ochoa, P. Machado, A. Cardoso, and M. R. Soto. Probabilistic evolution and the busy beaver problem. In D. Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference GECCO-2000*, pages 261–268, Las Vegas, Nevada, USA, 8 July 2000.
- [49] K. Sastry and A. Orriols-Puig. Extended compact genetic algorithm in Matlab. IlliGAL Report 2007009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2007.
- [50] S. Shakya, J. McCall, and D. Brown. Using a Markov network model in a univariate EDA: An empirical cost-benefit analysis. In H.-G. Beyer and U.-M. O'Reilly, editors, *Proceedings of Genetic and Evolutionary Computation Conference GECCO-2005*, pages 727–734, Washington, D.C., USA, 2005. ACM Press.
- [51] M. R. Soto. A Single Connected Factorized Distribution Algorithm and its Cost of Evaluation. PhD thesis, University of Havana, Havana, Cuba, July 2003. In Spanish.
- [52] M. R. Soto, A. Ochoa, S. Acid, and L. M. Campos. Bayesian evolutionary algorithms based on simplified models. In A. Ochoa, M. R. Soto, and R. Santana, editors, *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 360–367, Havana, Cuba, March 1999.
- [53] C. Yanover and Y. Weiss. Approximate inference and protein-folding. In S. Becker, S. Thrun, and K. Obermayer, editors, Advances in Neural Information Processing Systems 15, pages 1457–1464. MIT Press, Cambridge, MA, 2003.
- [54] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.
- [55] T.-L. Yu. A Matrix Approach for Finding Extrema: Problems with Modularity, Hierarchy and Overlap. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 2006.

[56] Q. Zhang. On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm. *IEEE Transactions on Evolutionary Computation*, 8(1):80–93, 2004.