

MATEDA: A suite of EDA programs in Matlab

Research Report EHU-KZAA-IK-2/09
Department of Computer Science and Artificial Intelligence
University of the Basque Country

Roberto Santana^{*}, Carlos Echegoyen[†], Alexander Mendiburu[†],
Concha Bielza[‡], Jose A. Lozano[†], Pedro Larrañaga[‡],
Rubén Armañanzas[†] and Siddartha Shakya^{*}

^{*}Universidad Politécnica de Madrid

[†]Intelligent Systems Group

Department of Computer Science and Artificial Intelligence

University of the Basque Country

[‡] Departamento de Inteligencia Artificial,

Universidad Politécnica de Madrid. Campus de Montegacedo sn.

28660. Boadilla del Monte, Madrid, Spain.

^{*}Intelligent Systems Research Centre, BT Innovate, Ipswich, UK.

email:rsantana@ehu.es

Abstract

This paper describes MATEDA-2.0, a suite of programs in Matlab for estimation of distribution algorithms. The package allows the optimization of single and multi-objective problems with estimation of distribution algorithms (EDAs) based on undirected graphical models and Bayesian networks. The implementation is conceived for allowing the incorporation by the user of different combinations of selection, learning, sampling, and local search procedures. Other included methods allow the analysis of the structures learned by the probabilistic models, the visualization of particular features of these structures and the use of the probabilistic models as fitness modeling tools.

Contents

1	Introduction	4
2	Installing MATEDA-2.0	4
3	Defining and executing an EDA	5
3.1	General description of the implementation	5
3.2	Implementation of a general EDA	5
3.2.1	Input parameters	5
3.2.2	Output parameters	7

4	Probabilistic models used in MATEDA-2.0	8
4.1	Factorized distributions	8
4.2	Bayesian and Gaussian networks	10
4.2.1	Bayesian networks	10
4.2.2	Bayesian network toolbox	11
4.2.3	Gaussian networks	11
4.3	Markov networks	12
4.4	Mixtures of distributions	12
4.5	Non probabilistic models	13
5	Implementation of the EDA components	14
5.1	Seeding methods	14
5.2	Sampling methods	15
5.3	Repairing methods	16
5.4	Local optimization methods	16
5.5	Replacement methods	16
5.6	Selection methods	18
5.7	Learning methods	18
5.8	Methods to compute the statistics	20
5.9	Methods to display information during the evolution	20
5.10	Methods to evaluate the stopping conditions	21
6	Functions and testbed problems implemented in MATEDA-2.0	21
6.1	Single objective functions	21
6.1.1	Additively decomposable functions	21
6.2	Multi-objective functions	22
6.2.1	Multi-objective additive decomposable functions	22
6.2.2	Function generator methods	23
7	How to use MATEDA-2.0 for a given problem	24
7.1	Steps to run an EDA in MATEDA-2.0	24
7.2	Examples of the optimization problems implemented	25
7.2.1	Tree-EDA for the Ising model	25
7.2.2	FDA optimization of the Hydrophobic-Polar (HP) protein model	26
7.2.3	Bayesian network based EDA for a multi-objective 3-satisfiability problem	28
7.2.4	Mixture of multivariate Gaussian distributions for a spacecraft trajectory optimization problem	28
8	Analysis of data related to the points generated and the evaluation of the (possible multiple) objective functions	30
8.1	Fitness related measures	30
8.2	<i>A posteriori</i> analysis of the dependencies	32
8.2.1	Parallel coordinate visualization of objectives	33
9	Analysis of the probabilistic models	34
9.1	Data structures to represent the models	34
9.2	Methods implemented for the analysis and visualization of the structures	36

9.2.1	<code>ViewSummStruct</code> method	36
9.2.2	<code>ViewInGenStruct</code> method	37
9.2.3	<code>ViewEdgDepStruct</code> method	37
9.2.4	<code>ViewPCStruct</code> method	38
9.2.5	<code>ViewDenDroStruct</code> method	40
9.2.6	<code>ViewGlyphStruct</code> method	41
10	Function approximation module	42
10.1	Probabilistic models of (possibly multi-objective) fitness functions	42
10.2	MATEDA-2.0 methods for fitness function approximation	44
11	Conclusions	45
11.1	EDAs that can be implemented with MATEDA-2.0	45

1 Introduction

EDAs [62, 69, 80, 91] are evolutionary algorithms based on estimation and sampling from probabilistic models and able to overcome some of the drawbacks exhibited by traditional genetic algorithms (GAs) [37, 46]. Additionally, the probabilistic models used by EDAs can represent a priori information about the problem structure, allowing a more efficient search of optimal solutions. Learning algorithms can also be used to reveal previously unknown information about the structure of black box optimization problems.

Several EDAs have been proposed for discrete, continuous and mixed problems. These algorithms mainly differ in the class of probabilistic models employed and the learning and sampling methods they use. However, it has been acknowledged that the selection and replacement strategies used can also determine important differences in the EDAs behavior. Sometimes, it is difficult to decide which is the best EDA choice for a given problem and the user would like to compare at least a short list of combinations of EDA operators in terms of their efficacy and efficiency. Other times, it would be convenient to modify only one component of the EDA, keeping the rest of the components unchanged. MATEDA-2.0 is conceived for these situations. The idea is that the user can easily evaluate a number of variants of EDAs before taking a decision about the final implementation.

Another important use of EDAs is to reveal previously unknown information about the search space that has been captured during the execution of the algorithm. This information can be encoded in the probabilistic models learned or in the points visited during the search. Similarly, the probabilistic models can be employed as models of functions. It is useful the design of methods that permit to evaluate the quality of the probabilistic models as function predictors.

MATEDA-2.0 includes three main modules integrated by programs that intend to fulfill the following objectives:

- Optimization module: Implementation of different EDAs for single and multi-objective problems.
- Data analysis and visualization module: For detecting, extracting and visualizing characteristic features in the structures of the models learned during the evolution.
- Function approximation module: Creation and validation of models of the functions based on the probabilistic models learned by EDAs.

2 Installing MATEDA-2.0

MATEDA-2.0 employs the Matlab Bayes Net (BNT) toolbox [82] and the BNT structure learning package [65]. These programs, which are freely available from the authors website¹, should be installed previously to the MATEDA-2.0 installation. Some of the MATEDA-2.0 routines also employs the MATLAB statistical toolbox and the affinity propagation clustering algorithm[31]².

¹They can be respectively downloaded from <http://www.cs.ubc.ca/murphyk/Software/BNT/bnt.html> and <http://banquiseasi.insarouen.fr/projects/bntslp/>

²The Matlab implementation of affinity propagation is available from <http://www.psi.toronto.edu/affinitypropagation/>

In order to install the software, follow these steps:

1. Unpack the file `IntEDA.tar.gz` and copy the files to a directory named `MATEDA`.
2. Edit file `InitEnvironment.m` updating the paths `'path_MATEDA'`, `'path_FullBNT'` and `'path_BNT_SLP'`.
3. Set the current Matlab directory to the `MATEDA` directory.
4. Execute program `InitEnvironments.m`. Several warnings but no error should appear.

The file `ScriptMateda.m` contains several examples of EDAs implementation. If the reader is familiarized with the Matlab environment and EDAs, these examples should be sufficient for a basic EDA implementation. Otherwise, the following sections provide a detailed explanation of the `MATEDA-2.0` components.

3 Defining and executing an EDA

The main component of `MATEDA-2.0` is the definition of an EDA in terms of its parameters and components. The pseudocode of this EDA is described in Algorithm 1.

3.1 General description of the implementation

The pseudocode of the general EDA is described in Algorithm 1. Each of the main methods that can be implemented by the user using `MATEDA-2.0` are emphasized in Algorithm 1.

3.2 Implementation of a general EDA

The general EDA program `RunEDA.m` is called as:

```
[AllStat,Cache] = RunEDA(PopSize,n,F,Card,cache,edaparams);
```

where the input and output parameters have the following meaning:

3.2.1 Input parameters

- *PopSize*: Population size.
- *n*: Number of variables.
- *F*: Name of the Matlab file that implements the (possibly multiobjective) function.
- *Card*: Cardinalities of the variables for the discrete problems or range of each variable for the continuous problem.
- *cache*: A vector specifying which components of the algorithm will be stored. $cache(i) = 1$ determines whether the i -th component of EDA ($i = 1, 2, \dots, 5$) will be saved in each generation and $cache(i) = 0$ otherwise. The five components considered are the following:

Algorithm 1: Estimation of distribution algorithm

```
1 Set  $t \leftarrow 0$ .
2 do {
3   If  $t = 0$ .
4     Generate an initial population  $D_0$  using a seeding method.
5     If required, apply a repairing method to  $D_0$ .
6     Evaluate (all the objectives of) population  $D_0$  using an evaluation method.
7     If required, apply a local optimization method to  $D_0$ .
8   Else.
9     Sample a  $D_{Sampled}$  population from the model using a sampling method.
10    If required, apply a repairing method to  $D_{Sampled}$ .
11    Evaluate (all the objectives of) population  $D_{Sampled}$  using an evaluation method.
12    If required, apply a local optimization method to  $D_{Sampled}$ .
13    Create a  $D_t$  population from populations  $D_{t-1}$ ,  $D_{Sampled}$ , and  $D_t^S$  using a replacement method.
14    Select a set  $D_t^S$  of points according to a selection method.
15    Compute a probabilistic model of  $D_t^S$  using a learning method.
16     $t \leftarrow t + 1$ 
17 } until The evaluation of the termination criteria method is true.
```

1. Entire population.
2. Selected population.
3. Probabilistic model.
4. Fitness values of the entire population.
5. Fitness values of the selected population.

- *edaparams*: An array of cells specifying all the components and parameters used by the EDA. The i -th row of **edaparams** has the form:
{*type_of_method*, *name_of_implementation*, *implementation_parameters*}.
type_of_method defines an EDA component. It is a string that can take one of the following values:

- 'seeding_pop_method'
- 'sampling_method',
- 'repairing_method'
- 'local_opt_method'
- 'replacement_method'
- 'selection_method'
- 'learning_method'
- 'statistics_method'
- 'verbose_method'

– 'stop_cond_method'

name_of_implementation is the name of a Matlab program where the EDA component has been implemented. It can be added by the user or be one of the methods included in MATEDA-2.0. *implementation_parameters* is a cell array containing the parameters used by the program *name_of_implementation* which are passed to it during the execution of `RunEDA`.

3.2.2 Output parameters

- *AllStat*: For each generation k the cell array $AllStat\{k,:\}$ contains the following information:
 - $AllStat\{k,1\}$: Matrix of 5 rows and *number_objectives* columns. Each row shows information about maximum, mean, median, minimum, and variance values of the corresponding objective in the current population
 - $AllStat\{k,2\}$: Stores the best individual.
 - $AllStat\{k,3\}$: Number of different individuals.
 - $AllStat\{k,4\}$: Matrix of 5 rows and n columns. Each row shows information about maximum, mean, median, minimum, and variance values of the corresponding variable in the current population.
 - $AllStat\{k,5\}$: Number of function evaluations until generation k .
 - $AllStat\{k,6\}$: Matrix with the time, in seconds, spent at the main EDA steps, each of the 8 columns stores the times elapsed in the following steps: sampling, repairing, evaluation, local optimization, replacement, selection, learning and total (which represents the time spent by the previous 7 and other EDA operations).
- If $cache(i) = 1$, for each generation k , $Cache\{i,k\}$ will store the corresponding component of the EDA. The order is the one presented in the explanation of the input parameter *cache*.

Example 1 (Implementation of a continuous Gaussian UMDA).

```

1  PopSize = 500;
2  n = 30;
3  F = 'sum';
4  Card(1,:) = zeros(1,n);
5  Card(2,:) = 5*ones(1,n);
6  cache = [0,0,0,0,0];
7  edaparams{1} = {'learning_method','LearnGaussianUnivModel',{}};
8  edaparams{2} = {'sampling_method','SampleGaussianUnivModel',{PopSize,1}};
9  edaparams{3} = {'replacement_method','elitism',{1,'fitness_ordering'}};
10 edaparams{4} = {'selection_method','prop_selection',{}};
11 [AllStat,Cache] = RunEDA(PopSize,n,F,Card,cache,edaparams);

```

The code shown above is an implementation of a continuous Gaussian univariate marginal distribution algorithm (G-UMDA) [60, 61] using proportional selection and elitism of one individual. Initially, the parameters of the EDA are defined as: population size ($PopSize = 500$), number of variables ($n = 30$) and fitness function ($F = 'sum'$), which corresponds to the sum of the variables values, i.e. $f(x) = \sum_{i=1}^n x_i$. The problem has a continuous representation and

therefore the range of values are defined by instantiating the two rows matrix *Card*, in such a way that $\forall i, 0 \leq x_i \leq 5$. *cache* = [0, 0, 0, 0, 0] indicates that the information about the EDA evolution will not be stored.

edaparams{1} defines the learning method used which is implemented in function `LearnGaussianUnivModel.m`. This method learns the mean and variance for each of the variables and outputs them as the model of the selected population. It does not receive any other parameter apart from the selected population size. *edaparams*{2} defines the sampling method used which is implemented in function `SampleGaussianUnivModel.m`. It receives the probabilistic model and samples a new population of *PopSize* individuals using the mean and variance description of the data. `SampleGaussianUnivModel.m` receives two parameters which are passed in the array `{PopSize, 1}`.

edaparams{3} defines the replacement method which is implemented in function `elitism.m`. To determine the best (elite) solutions, program `elitism.m` requires the specification of the number of elitist solutions and a criterion to order the solutions. These are passed as parameters in the array `{1, fitness_ordering}` where `fitness_ordering.m` is a function that orders the individuals according to their fitness values. The individual with highest fitness value is the first in the ordering. *edaparams*{4} defines the selection method which is implemented in function `prop_selection.m`.

4 Probabilistic models used in MATEDA-2.0

The learning and sampling algorithms used by EDAs depend on the class of probabilistic models. Usually, instances of the same class of probabilistic models (e.g Bayesian network, Markov chain, etc.) are learned in each generation. However, we can think of cases where a different class of probabilistic model could be learned in each generation [107]. Switching the class of models according to the characteristics of the data to be modeled is a natural way to introduce adaptation in EDAs. MATEDA-2.0 allows the user to learn different classes of models in each generation. The only requirement is that the model learned at generation *t* be compatible with the sampling algorithm used at the same generation.

In this section, we present the main types of probability models implemented in MATEDA-2.0.

4.1 Factorized distributions

Some probability distributions can be expressed as a product of marginal probability distributions, each of which is called a factor. Factorized distributions, or factorizations, are an effective way to obtain a condensed representation of otherwise very difficult to store probability distributions. In the following analysis we focus on factorizations of discrete distributions.

We can identify two components of the factorization.

1. The structure of the factorization, which contains information about which variables belong to each of the factors and the relationships with other factors.

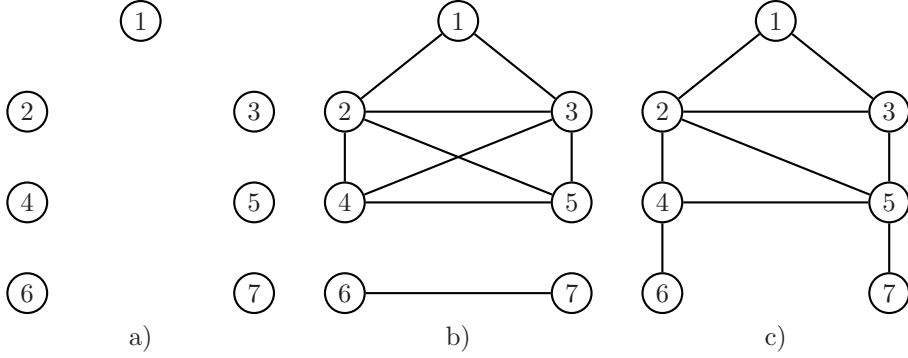


Figure 1: Graph representing factorizations of different complexity. a) Univariate factorization. b) Factorization with maximum clique of size 4. c) Factorization with maximum clique of size 3.

2. The parameters of the factorizations, which are the parameters of each of the factors. In the case of discrete factorizations these are usually the probability values of each factor configuration.

MATEDA-2.0 represents a factorization using two components:

1. *Cliques*, which represent the variables of each factor, specifying whether they are also included in previous factors or have not appeared before.
2. *Tables*, which contain a probability table for each of the factors.

Each row of *Cliques* is a clique. The first column is the number of overlapping variables with respect to previous cliques in *Cliques*. The second column is the number of new variables. Then, overlapping variables are listed, and finally new variables are listed. *Tables* $\{i\}$ stores the marginal tables for clique i .

Example 2 shows different undirected graphs (Figure 1), their associate factorizations and the corresponding MATEDA-2.0 representation (Cliques are respectively shown in Equations 4, 5 and 6).

As illustrated by Example 2, the data format used by *Cliques* serves to store the wide class of ordered factorizations [101], which includes marginal product factorizations [40, 80, 109], Markov chain factorizations [103], factorizations constructed from junction trees [79, 84] and junction graphs [110].

Example 2 (Representation of different factorizations in MATEDA-2.0).

$$p^a(\mathbf{x}) = \prod_{i=1}^7 p(x_i) \quad (1)$$

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

$$p^c(\mathbf{x}) = \frac{p(x_1, x_2, x_3)p(x_2, x_3, x_5)p(x_2, x_4, x_5)p(x_4, x_6)p(x_5, x_7)}{p(x_2, x_3)p(x_2, x_5)p(x_4)p(x_5)} \quad (3)$$

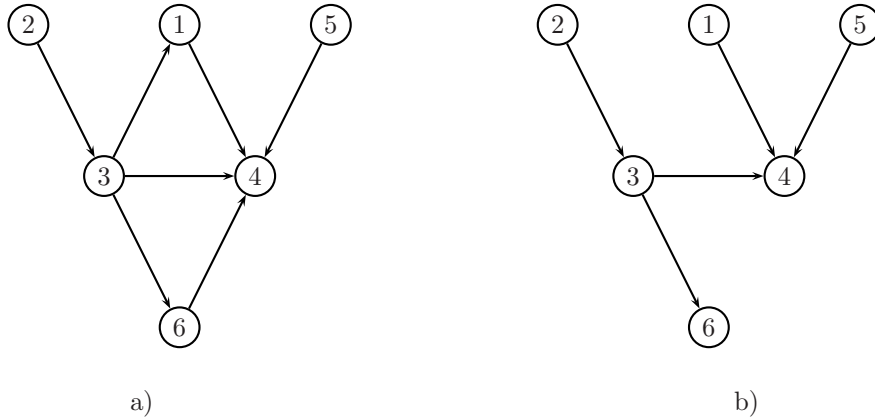


Figure 2: Structures of: a) General Bayesian network; b) Polytree.

$$Cliques = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 3 \\ 0 & 1 & 4 \\ 0 & 1 & 5 \\ 0 & 1 & 6 \\ 0 & 1 & 7 \end{pmatrix} \quad (4)$$

$$Cliques = \begin{pmatrix} 0 & 3 & 1 & 2 & 3 & 0 \\ 2 & 2 & 2 & 3 & 4 & 5 \\ 0 & 2 & 6 & 7 & 0 & 0 \end{pmatrix} \quad (5)$$

$$Cliques = \begin{pmatrix} 0 & 3 & 1 & 2 & 3 \\ 2 & 1 & 2 & 3 & 5 \\ 2 & 1 & 2 & 5 & 4 \\ 1 & 1 & 4 & 6 & 0 \\ 1 & 1 & 5 & 7 & 0 \end{pmatrix} \quad (6)$$

Figures 1 a) b) c) show the graphs that correspond to the structure of the factorizations represented by Equations (1), (2) and (3) respectively. The MATEDA-2.0 representation of these factorizations are respectively shown in (4), (5) and (6).

4.2 Bayesian and Gaussian networks

4.2.1 Bayesian networks

Bayesian networks are graphical models based on directed acyclic graphs. They have been used for probabilistic inference in domains such as expert systems [21, 63] classification problems [5, 32], and optimization [29, 88].

In a Bayesian network, where the discrete variable X_i has r_i possible values, $x_i^1, \dots, x_i^{r_i}$, the local distribution $p(x_i \mid \mathbf{pa}_i^{j,S}, \theta_i)$ is an unrestricted discrete distribution:

$$p(x_i^k \mid \mathbf{pa}_i^{j,S}, \theta_i) = \theta_{x_i^k \mid \mathbf{pa}_i^j} \equiv \theta_{ijk} \quad (7)$$

where $\mathbf{pa}_i^{1,S}, \dots, \mathbf{pa}_i^{q_i,S}$ denotes the values of \mathbf{Pa}_i^S , the set of parents of X_i in the structure S . q_i is the number of possible different instances of the parent variables of X_i , hence $q_i = \prod_{X_g \in \mathbf{Pa}_i^S} r_g$.

The local parameters are given by $\theta_i = ((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i}$. Parameter θ_{ijk} is the conditional probability of variable X_i being in its k -th state given that the set of parents is in its j -th configuration.

4.2.2 Bayesian network toolbox

MATEDA-2.0 uses the Matlab Bayes Net (BNT) toolbox [82] and the BNT structure learning package [65] for the implementation of EDAs that use Bayesian and Gaussian models. We include a very brief description of the main features of the toolboxes used by MATEDA-2.0. Further details could be consulted in [65].

In the BNT toolbox, a directed acyclic graph (dag) is a structure that serves to represent the structure of a Bayesian network (bnet). Example 3 shows the different steps in the creation of a Bayesian network from a given dag. First, the Bayesian network is initialized using the dag and the cardinality of the variables. In the second step, the type of each variable is specified (tabular for discrete values and Gaussian for continuous variables). Finally, the parameters of the network are learned from the data. The BNT toolbox includes methods for learning the Bayesian network structure from the data. They are analyzed in Section 5.7.

Example 3 (Bayesian network learning in BNT).

```

1 n = 4;
2 Card = 2*ones(1,n);
3 dag(1,:) = [0 0 0 1];
4 dag(2,:) = [0 0 0 0];
5 dag(3,:) = [0 1 0 1];
6 dag(4,:) = [0 0 0 1];
7 init_bnet = mk_bnet(dag,Card);
8 for=1:n
9   init_bnet.CPDi = tabular_CPD(init_bnet,i);
10 end,
11 bnet = learn_params(init_bnet,data);

```

4.2.3 Gaussian networks

In a Gaussian network [119], each variable $X_i \in \mathbf{X}$ is continuous and each local density function is the linear-regression model:

$$f(x_i \mid \mathbf{pa}_i^S, \theta_i) \equiv \mathcal{N}(x_i; m_i + \sum_{x_j \in \mathbf{pa}_i} b_{ji}(x_j - m_j), v_i) \quad (8)$$

where $\mathcal{N}(x; \mu, \sigma^2)$ is a univariate normal distribution with mean μ and variance σ^2 . Given this form, a missing arc from X_j to X_i implies that $b_{ji} = 0$ in the former linear-regression model. The local parameters are given by $\theta_i = (m_i, \mathbf{b}_i, v_i)$, where $\mathbf{b}_i = (b_{1i}, \dots, b_{i-1i})^t$ is a column vector.

The interpretation of the components of the local parameters is as follows: m_i is the unconditional mean of X_i , v_i is the conditional variance of X_i given

Pa_i , and b_{ji} is a linear coefficient reflecting the strength of the relationship between X_j and X_i .

4.3 Markov networks

Given an undirected graph $G = (V, E)$ we have the following definitions [39]:

Definition 1. *The neighborhood $N(X_i)$ of a vertex $X_i \in \mathbf{X}$ is defined as $N(X_i) = \{X_j : (X_j, X_i) \in E\}$. The set of edges uniquely determines a neighborhood system on the associated graph G .*

Definition 2. *The boundary of a set of vertices, $\mathbf{X}_S \subseteq \mathbf{X}$, is the set of vertices in $\mathbf{X} \setminus \mathbf{X}_S$ that neighbors at least one vertex in \mathbf{X}_S . The boundary of \mathbf{X}_S is denoted as $bd(\mathbf{X}_S)$.*

Definition 3. *The closure of a set of vertices, $\mathbf{X}_S \subseteq \mathbf{X}$, is the set of vertices $cl(\mathbf{X}_S) = \mathbf{X}_S \cup bd(\mathbf{X}_S)$.*

Definition 4. *A probability $p(\mathbf{x})$ is called a Markov random field with respect to the neighbor system on a graph G if, $\forall \mathbf{x} \in \mathbf{X}, \forall i \in \{1, \dots, n\}, p(x_i | \mathbf{x} \setminus x_i) = p(x_i | bd(x_i))$.*

Definition 5. *A probability $p(\mathbf{x})$ on a graph G is called a Gibbs field with respect to the neighborhood system on the associated graph G when it can be represented as follows:*

$$p(\mathbf{x}) = \frac{1}{Z} e^{-H(\mathbf{x})} \quad (9)$$

where $H(\mathbf{x}) = \sum_{C \in \mathcal{C}} \Phi_C(\mathbf{x})$ is called the energy function, being $\Phi = \{\Phi_C \in \mathcal{C}\}$ the set of clique potentials, one for each of the maximal cliques in G . The value of $\Phi_C(\mathbf{x})$ depends on its local configuration on the clique C . The normalizing constant Z is the corresponding partition function, $Z = \sum_{\mathbf{x}} e^{-H(\mathbf{x})}$.

In MATEDA-2.0, *Cliques* are used to represent the neighborhood structure in models based on Markov networks [100, 121, 123]. In this particular case, the first column of *Cliques*($i, :$) represents the number of neighbors for variable X_i . The second, is the number of new variables (for Markov networks, only one new variable X_i appears in each clique.). Then, the neighbor variables are listed and finally variable X_i is added.

The parameters of the Markov network are represented using the *Tables* structure which stores the conditional probabilities of each variable given its neighbors.

4.4 Mixtures of distributions

A mixture of distributions [30, 72] is defined to be a distribution of the form:

$$Q(x) = \sum_{j=1}^m \lambda_j \cdot f_j(x) \quad (10)$$

with $\lambda_j > 0, j = 1, \dots, m, \sum_{j=1}^m \lambda_j = 1$.

f_j are called component densities or mixture components, and λ_j are called mixture proportions or mixture coefficients. m is the number of components

of the mixture. A mixture of distributions can be viewed as containing an unobserved choice variable z , which takes values $k \in \{1, \dots, m\}$ with probability λ_j . In some cases the choice variable z is known.

Examples of mixtures distributions include:

- Mixtures of Gaussian distributions [30].
- Bayesian multi-nets [35].
- Mixtures of trees [73].

In EDAs, mixtures of distributions have been extensively applied to solve discrete [11, 87, 93, 114, 105] and continuous [11, 17, 33, 127] optimization problems.

In Mateda-2.0, a mixture of models is represented as an array of components and coefficients. Each element of the array stores all the relevant information about the corresponding model. A mixture can contain as components models of different classes. Example 4 shows the way in which a mixture of Gaussian distributions can be learned from a cluster of solutions.

Example 4 (Constructing a mixture of Gaussian distributions).

```

1 % nclusters: number of clusters where solutions in AuxPop are
2 grouped
3 % ind: index of the cluster each solution belongs to.
4 for i=1:nclusters,
5     idx = find(ind==i);
6     nmembers = size(idx,1); % Number
7     model1,i = mean(AuxPop(idx,:)); % Vector of means for each cluster
8     model2,i = std(AuxPop(idx,:)); % Vector of standard deviation for each cluster
9     model3,i = nmembers/PopSize; % Coefficient for the mixture, proportional to the
10 end % number of points in the cluster

```

4.5 Non probabilistic models

Even if MATEDA-2.0 is conceived for EDAs where modeling of the solutions is done using probabilistic models, there may be situations in which population based algorithms that use other types of modeling techniques (e.g. algorithms that use modeling based on neural networks) or do not use any type of modeling (e.g. genetic algorithms [37]) are to be compared with EDAs. Assuming that the main difference of these algorithms with EDAs lies in the variation operation they use (i.e. the evaluation and selection steps are essentially identical), these algorithms could be implemented in MATEDA-2.0 using two alternatives:

- Split the implementation of the variation operator in two parts. In the first part, a *model* is learned. In the second, the model is used to sample solutions.
- Define a learning method where nothing is done and implement the variation operators in the sampling method.

Example 5 presents a model of one-point crossover operator of a GA. The models correspond to a list of triples, where the first value of each triple corresponds to the crossover point for the creation of an individual and the other two values are the indices of the parents in the selected population. In this simple

example, there will be generated 10 new individuals. The first two offsprings will be formed, during the sampling step, by crossing parents 1 and 3 at point 5.

Example 5 (Model of one-point crossover for a GA).

$$Cliques = \begin{pmatrix} 5 & 3 & 1 \\ 6 & 4 & 2 \\ 2 & 6 & 2 \\ 8 & 5 & 3 \\ 7 & 2 & 5 \end{pmatrix} \quad (11)$$

5 Implementation of the EDA components

There is a common way in which all EDA components are invoked from the main program RunEDA using the Matlab function `eval`. This is:

`[Output] = eval([eda_method, '(fixed_params,user_params)']);` where *eda_method* is the name of the Matlab program that implements the method. *fixed_params* are a set of parameters determined by the RunEDA implementation and *user_params* are parameters defined by the users and passed to the program using *eda_params*. In the following section we briefly describe the format of the implementations for each of the EDA components. Afterwards, several examples that illustrate the use of these components are discussed.

5.1 Seeding methods

Seeding methods are used to initialize the starting population when some knowledge about the problem is available or the user wants to bias the search to certain areas of the space.

In RunEDA, the seeding procedure is invoked at the first population as:

`Pop=eval([seeding_pop_method, '(n,PopSize,Card,seeding_pop_params)']);`

The seeding methods implemented in MATEDA-2.0 are the following (see help function-name for details on the input parameters used by the methods):

- `seed_thispop`: Seeds a given population.
- `RandomInit`: Samples a population of solutions from the uniform distribution.
- `Bias_Init`: Biased initialization of a population of binary vectors, where the probability of generating 1 is p and the probability of generating 0 is $1 - p$.
- `seeding_unitation_constraint`: Generates a population of binary vectors where all vectors have the same number of ones. It might be used for problems with unitation constraints [112, 113].

By default, *seeding_pop_method* = *RandomInit*.

5.2 Sampling methods

Sampling methods serve to generate the new population from the probabilistic model learned by the EDA. The type of sampling method is therefore dependent of the class of probabilistic method used. Traditionally, probabilistic logic sampling (PLS) [44] has been the choice in EDAs, but other proposals incorporate Gibbs sampling [34, 101, 121, 123], and methods that find the most probable configurations [74, 102, 124]. MATEDA-2.0 implements variants of all these sampling methods.

In `RunEDA`, the sampling procedure is invoked at every generation, except the first one where seeding is applied, as:

```
NewPop =eval(['sampling_method','(n,model,Card,SelPop,SelFunVal',  
,sampling_params)']);
```

where `model` is a cell array containing the description of the probabilistic model. The inclusion of the selected population and its evaluation as parameters of the sampling method allow the implementation of sampling algorithms that may start from previously found solutions (e.g. Gibbs sampling).

The following sampling methods have been implemented (see help function-name for details on the input parameters used by the methods):

- `MOAGeneratePopulation`: Samples a Markov network using Gibbs Sampling.
- `SampleFDA`: Samples from a factorized model using PLS.
- `SampleGaussianUnivModel`: Samples from a univariate Gaussian model.
- `SampleGaussianFullModel`: Samples from a full multivariate Gaussian model.
- `SampleMixtureofUnivGaussianModels`: Samples from a mixture of univariate Gaussian models.
- `SampleMixtureofFullGaussianModels`: Samples from a mixture of full multivariate Gaussian models.
- `SampleBN`: Samples a population of individuals from a Bayesian or Gaussian network using PLS.
- `SampleMPE-BN`: Samples a population of discrete solutions in which the first individual corresponds to the most probable configuration given by the model and the remaining individuals are sampled using PLS.

The `SampleBN` and `SampleMPE-BN` programs invoke methods defined in the BNT toolbox. EDAs that use multivariate Gaussian distributions have shown to suffer from stagnation due to a very fast (at exponential rate) decrease of the variance [7]. A partial remedy is to use an artificial expansion of the variance. Therefore, in MATEDA-2.0 some of the EDAs that use Gaussian models include as a feature a variance scaling parameter. Other adaptive schedules [8, 24, 38, 95] for modifying the variance during sampling could be implemented in this framework.

5.3 Repairing methods

Repairing methods are commonly used to guarantee the feasibility of solutions. They modify (repair) a given individual to guarantee that the constraints are satisfied.

In `RunEDA` the repairing procedure is invoked at every generation as:

```
[Pop] = eval([repairing_method, '(Pop,Card,repairing_params)'])
```

The following repairing methods have been implemented (see help function-name for details on the functions parameters):

- **SetInBounds_repairing**: For a problem with continuous representation, this program changes the values of each out of range variable to the minimum (respectively maximum) bounds if variables are under (respectively over) the variables ranges.
- **SetWithinBounds_repairing**: For a problem with continuous representation, this program truncates the values of each out of range variable to a random value within the feasible range.
- **HP_repairing**: Recursive procedure, originally introduced in [20], to repair a grid path to avoid self-intersections, guaranteeing feasibility.

5.4 Local optimization methods

Local optimization methods are used to improve the solutions sampled by EDAs. The combination of EDAs and local optimization methods [78, 89, 108, 133] have been shown to produce important improvements in the efficiency of the optimization process.

In `RunEDA` the local optimization method is invoked at every generation as:

```
[Pop, NumbEvals] = eval([local_opt_method, '(k,Pop,FunVal,local_opt_params)'])
```

FunVal is a matrix with fitness values of all the individuals in *Pop* for all the objectives. Since the local optimization program receives the entire population, it is possible to implement optimization methods that use information about the population to modify each individual. Otherwise, the local optimization method can be applied to each individual separately. *NumbEvals* is the number of evaluations made during the local optimization step.

5.5 Replacement methods

Replacement methods are used to combine the individuals generated in the current population with individuals from previous generations. In EDAs, they can be used as an effective way to promote diversity in the population (e.g. restricted tournament replacement [86]). In addition, research on the role of replacement methods in EDAs [67] has shown that the type of replacement method can influence the accuracy of the probabilistic models learned by the algorithms. Niching methods [70] may also be implemented at this step.

In `RunEDA` the replacement method is invoked at every generation except the first one as:

```
[Pop, FunVal] = eval([replacement_method, '(OldPop, SelPop, NewPop, OldFunVal, SelFunVal, NewPopFunVal, replacement_params)']);
```

where *OldPop*, *SelPop* and *NewPop* are respectively the population at step $t - 1$, the selected population and the population generated by sampling (and possibly repaired or improved using local optimization). *OldFunVal*, *SelFunVal*, and *NewPopFunVal* are matrices with the fitness values, for all the objectives, of the individuals in *OldPop*, *SelPop* and *NewPop*.

The following replacement methods have been implemented:

- **elitism**: Adds the k best individuals of the previous population to the current population.
- **best_elitism**: Joins the selected individuals with the sampled individuals to form the next population. It should be enforced that $SampledPopSize = PopSize - SelPopSize$.
- **popaggregation**: Joins the current population with the sampled population and selects the best $PopSize$ individuals as the new population.
- **none**: $Pop = NewPop$.
- **RT_replacement**: Creates a new population (*NewPop*) applying restricted tournament replacement.

All these methods require the definition of an ordering criterion to determine the best individuals. While this criterion may be straightforward for the single objective optimization, there are many more options available for the case of multi-objective optimization.

In `RunEDA`, when an ordering of the individuals is required, an ordering method is invoked as:

```
[Ind] = eval([find_bestinds_method, '(Pop, FunVal)'])
```

where *find_bestinds_method* is the name of the ordering method, passed as a parameter of the main EDA component that uses it (e.g. replacement method), *Pop* is a population and *FunVal* the corresponding evaluations of the individuals for all the objectives. The method outputs an index *Ind* with the ordering.

The following ordering methods have been implemented:

- **fitness_ordering**: Individuals are ordered according to the ranking of its fitness values (best is the maximum). For multi-objective functions, individuals are ordered according to the average ranking for all the objectives (i.e. for each objective an ordering is done, and they are averaged later).
- **Pareto_ordering**: Individuals are ordered according to the front they belong to. Individuals in the first front (nondominated solutions) come first. Then individuals that are only dominated by those in the first front and so on. There is no ordering criteria for individuals in the same front. See `ParetoRank_ordering` for more refined ordering.
- **ParetoRank_ordering**: Individuals are firstly ordered according to the front they belong to. Then, in each front, they are ordered according to the average rank of their fitness functions. The first front (non-dominated solutions) comes first. Then individuals that are only dominated by those in the first front and so on.

5.6 Selection methods

Selection methods determine which the set of individuals that will be modeled is. Usually, it is expected that these individuals correspond to a sample of a current promising region of the search space. Selection methods are essential for EDAs. Therefore, some work has been devoted to study the role of selection operators in EDAs [51, 67, 71, 104]. In particular, to investigate their influence in the relationship interactions-dependencies. It has been shown [67] that while some selection methods (e.g. truncation selection) are good at keeping the original interactions between the problem variables, other selection methods can be better in terms of the number of function evaluations to reach the optimum (e.g. tournament selection).

In RunEDA the selection method is invoked at every generation as:

```
[SelPop,SelFunVal] = eval([selection_method,'(Pop,FunVal,
selection_params)'])
```

where the input and output parameters are as described for previous methods.

The following selection methods have been implemented:

- **exp_selection**: The selection probability of each individual is exponential of *base* to its fitness. The selected population is sampled using stochastic universal sampling (SUS) [3]. Implemented for single objective functions.
- **prop_selection**: The selection probability of each individual is proportional to its fitness. The selected population is sampled using SUS. Implemented for single objective functions.
- **NonDominated_selection**: The set of non dominated individuals is selected. Implemented for multi-objective functions. The number of selected individuals may change at each generation.
- **truncation_selection**: Individuals are ordered according to the given ordering (see previous section for implementations of the ordering criterion). The best $T * PopSize$ are selected where T is the truncation coefficient.

5.7 Learning methods

In EDAs, the probabilistic models are constructed using learning methods that build the models extracting relevant information from the data and (possibly) a priori knowledge about the model structure. Regarding the way learning is done in the probability model, EDAs can be divided into two classes [59]. One class groups the algorithms that do a parametric learning of the probabilities, and the other one comprises those algorithms where structural learning of the model is also done. Parametric and structural learning are also known as model fitting and model selection, respectively.

In RunEDA the learning method is invoked at every generation as:

```
[model] = eval([learning_method,'(k,n,Card,SelPop,
SelFunVal,learning_params)'])
```

where k is the number of the current generation and *model* has the structure explained in Section 4. The structure and parameters contained in *model* have to be consistent with the sampling algorithm used in the same generation.

To learn Bayesian and Gaussian networks, MATEDA-2.0 uses a number of methods implemented in the Matlab Bayes Net (BNT) toolbox [82] and the BNT structure learning package [65]. The following learning methods have been implemented:

- **LearnBN:** Learns a Bayesian network from data using the following approaches:
 - PC algorithm [125] as implemented in [82].
 - Tree learning algorithm using the mutual information [18] as implemented in [65].
 - Greedy score+search [19] with a randomly generated initial node ordering and Bayesian information criterion (BIC) [117] scores as implemented in [82].
 - Greedy score+search with an initial node ordering constructed from a learned tree structure [65], and Bayesian and BIC scores [19] as implemented in [82].
- **LearnFDA:** Creates a factorized model from a given fixed undirected structure.
- **LearnMargProdModel:** Learns a marginal product [109] model using affinity propagation [31] on the matrix of mutual information.
- **LearnMOAModel:** The Markov network model learned by the Markov optimization algorithm (MOA) [123] is learned. The structure of the model can be given or learned from the data.
- **LearnGaussianUnivModel:** Learns a Gaussian univariate marginal product model [61].
- **LearnGaussianFullModel:** Learns a full Gaussian multivariate model [61].
- **LearnMixtureofUnivGaussianModels:** Learns a mixture of univariate Gaussian models using clustering.
- **LearnMixtureofFullGaussianModels:** Learns a mixture of full multivariate Gaussian models using clustering.
- **LearnGaussianNetwork:** Learns a Gaussian network from data using the following approaches:
 - Tree learning algorithm using the BIC score as implemented in [65].
 - Greedy score+search with a randomly generated initial node ordering and BIC score [19] as implemented in [82].
 - Greedy score+search [19] with an initial node ordering constructed from a learned tree structure [65], and BIC score as implemented in [82].

The implemented learning methods incorporate parameters that add flexibility to the EDAs. For instance, EDAs that use mixtures of Gaussian models are based on a clustering of the solutions. Clustering can be done in the space of variables, the space of objectives, or both together. User defined clustering methods can be employed (currently available are affinity propagation and k-means) with different distance measures. Similarly, it could be also possible to add adaptation to learning by setting the coefficients of the mixture's components according to the accuracy of each component approximation [99].

Notice that the computational cost in terms of time and memory is very high for some of the Bayesian and Gaussian network learning methods making them appropriate for only small problems. See file `ScriptMateda.m` for examples of learning algorithms used for problems of many variables.

5.8 Methods to compute the statistics

To evaluate the performance of an EDA it is necessary to compute a number of statistics that describe the behavior of the algorithm. MATEDA-2.0 allows the user to implement a procedure that processes the information and computes a number of relevant statistics that could be used by the methods that display information about the algorithm during the evolution.

In `RunEDA`, the method to compute the statistics is invoked at every generation as:

```
[AllStat] = eval([statistics_method,'(k,Pop,FunVal,time_operations,
number_evaluations,AllStat,statistics_params)'])
```

where *time_operations* is a cell array containing the time (in seconds) spent by each component of the EDA at each generation, *number_evaluations* is a vector with the number of evaluations in each generation and *AllStat* is an array that contains the statistics at each generation and is updated by the method for computing the statistics.

The following method for computing the statistics has been implemented:

- `simple_pop_statistics`: Computes the following statistics in each EDA generation and stores them in *AllStat*.
 - Information about maximum, mean, median, minimum, and variance values of every objective in the current population.
 - Best individual (according to an ordering criterion).
 - Number of different individuals.
 - Information about maximum, mean, median, minimum, and variance values of every variable in the current population.

5.9 Methods to display information during the evolution

An important feedback about the behavior of EDAs is received from the type of information displayed during the evolution of the optimization algorithm. In MATEDA-2.0, this method may be implemented by the user.

In `RunEDA`, the method to display the EDA information is invoked at every generation as:

```
eval([verbose_method,'(k,AllStat,verbose_params,auxedaparams)'])
```

where *AllStat* is the cell array containing the information computed by the

statistics method and *auxedaparams* contains the description of all the EDA methods.

The following method for displaying the information about the EDA behavior has been implemented:

- **simple_verbose**: Displays the information computed by the **simple_pop_statistics**. In addition, the number of evaluations and the time spent by each EDA component are displayed. A description of each the methods used by the EDA is displayed as a preamble to the execution of the algorithm.

The method used to compute and store the statistics and the one used to display them should be consistent.

5.10 Methods to evaluate the stopping conditions

The stopping criterion used in EDA influences the efficiency of the algorithm. Several stopping criteria can be used, separately or combined. In MATEDA-2.0 the stopping criteria can be defined by the user by means of a method that determines whether the algorithm should or not stop.

In RunEDA, the method to evaluate the stopping conditions is invoked at every generation as:

```
continue_evolution = eval([stop_cond_method, '(k,Pop,FunVal,stop_cond_params)'])
```

The stopping condition criteria will use information about the current generation, the genotypic structure of the population and the function values.

The following methods for displaying the information about the EDA behavior have been implemented:

- **maxgen**: The algorithm stops when a maximum number of generations has been reached.
- **maxgen_maxval**: The algorithm stops either when a maximum number of generations has been reached or a given value of the function has been reached.

6 Functions and testbed problems implemented in MATEDA-2.0

To validate the behavior of the EDAs, MATEDA-2.0 includes the implementation of a number of optimization functions. In this section we present some of the functions that have been implemented and a number of methods used to generate instances of these functions. The use of EDAs to address more realistic problems are discussed in the next section.

6.1 Single objective functions

6.1.1 Additively decomposable functions

An additively decomposable function (ADF) is represented as follows:

$$f(\mathbf{x}) = \sum_{i=1}^m f_i(\mathbf{x}_{s_i}) \quad (12)$$

where \mathbf{x}_{s_i} are subvectors of \mathbf{x} called the definition sets of the function, and f_i are its defined subfunctions, or just the subfunctions, of $f(\mathbf{x})$.

ADFs are a clear example of functions where problem structure is known. In an ADF, structurally related variables are usually considered as those that belong to the same definition set. ADFs have been extensively employed to evaluate EDAs. In particular, deceptive ADFs have been used to illustrate the capacity of EDAs to deal with the linkage problem affecting simple genetic algorithms.

In MATEDA-2.0, an ADF can be implemented as the sum of its subfunctions for each definition set. The following ADFs have been implemented as examples:

- Goldberg deceptive function [36].
- *Trap* function of order k [1].

Single objective functions can also be implemented as particular case of multi-objective functions discussed in the next section.

6.2 Multi-objective functions

6.2.1 Multi-objective additive decomposable functions

MATEDA-2.0 includes a number of methods that allow the definition of multi-objective decomposable functions. This class of functions have been proposed [54, 55, 92] to investigate the behavior of EDAs for multi-objective problems. In some cases [54, 55, 92], the rationale behind the use of these functions has been to obtain objectives that compete in all or most partitions of an appropriate problem decomposition.

In MATEDA-2.0, general multi-objective decomposable functions can be defined by the user by means of the global variables *FunctionStructure* and *FunctionTables*. The first global variable defines which problem variables are involved in the evaluation of each of the objectives. *FunctionTables* defines the values of an objective for each configuration of the variables it depends on.

The main difference between this type of multi-objective decomposable functions and ordinary ADFs is that each subfunction will map to a different objective. As a result, we can have different objectives defined on the same subset of variables.

The following methods have been implemented:

- *EvaluateGeneralFunction*: Evaluates a vector on a multimodal function whose structure and values are respectively defined as global variables *FunctionStructure* and *FunctionTables*.
- *PartialEvaluateGeneralFunction*: Evaluates a vector on a multimodal function whose structure and values are respectively defined as global variables but only a subset of the objectives are evaluated.
- *SumEvaluateGeneralFunction*: Evaluates a vector on a multimodal function whose structure and values are respectively defined as global variables. The Sum of the objectives is given as output.

- `SumPartialEvaluateGeneralFunction`: Evaluates a vector on a multimodal function whose structure and values are respectively defined as global variables but only a subset of the objectives are evaluated and the sum of these objectives is given as output.

One multi-objective decomposable function can be transformed to a single objective function by adding the values of all the objectives. This transformation, that could be seen as the opposite direction of the so-called *multiobjectivization* technique [56] can be done in MATEDA-2.0 using `SumEvaluateGeneralFunction` method. Additionally, one may be interested in evaluating just a subset of objectives. This can be accomplished by specifying a global variable `SelectedObjectives` and applying the method `PartialEvaluateGeneralFunction`.

Example 6 shows the generation and optimization of a multi-objective decomposable function using MATEDA-2.0.

Example 6 (Optimization of multi-objective decomposable functions).

```

1 PopSize = 1000;
2 n = 50;
3 cache = [1,1,1,1,1];
4 Card = 2*ones(1,n);
5 MaxGen = 30;
6 global FunctionTables;
7 global FunctionStructure;
8 global FunctionAccCard;
9 global SelectedObjectives;
10 % The circular structure is created
11 FunctionStructure = CreateListFactorsCircularNK(n,4);
12 % Values are read from a file
13 FunctionTables = ReadFunctionsFromData('testNK_fnt_N50_k4Inst_1.txt',
14 FunctionStructure,Card);
15 % Auxiliary structure for evaluation
16 FunctionAccCard = FindListCard(FunctionStructure,Card);
17 SelectedObjectives = [1:4:48]; % Only some of the objectives are evaluated
18 % General function that uses the global variables.
19 F = 'PartialEvaluateGeneralFunction';
20 selparams(1:2) = {0.5, 'ParetoRank_ordering'};
21 edaparams{1} = {'selection_method', 'truncation_selection', selparams};
22 edaparams{2} = {'replacement_method', 'best_elitism', {'ParetoRank_ordering'}};
23 edaparams{3} = {'stop_cond_method', 'max_gen', {MaxGen}};
24 [AllStat,Cache]=RunEDA(PopSize,n,F,Card,cache,edaparams);

```

The code shown above illustrates the way in which multi-objective decomposable functions can be generated and used in MATEDA-2.0. In this example, the multi-objective function is based on the NK circular fitness landscape [53]. Each objective corresponds to a subfunction where each variable depends on its $\frac{k}{2}$ previous and $\frac{k}{2}$ subsequent neighbors. The structure is generated using `CreateListFactorsCircularNK` method, which together with other function generator methods is explained in the next section. Only a subset of 12 objectives is selected for optimization. The default probabilistic model (Tree-EDA) is used with `ParetoRank_ordering` selection.

6.2.2 Function generator methods

Function generator methods can be defined by the user to generate multiple instances of a given decomposable function. MATEDA-2.0 includes a number of auxiliary methods with this purpose. These methods can also be used to save

and read the structure and values of the function from files. A multi-objective implementation of the NK landscape model is included as an example of how to use the function generator methods.

- **CreateNKFunctions**: Generates random values for the table entries of the NK landscape.
- **CreateGaussianValuesForFactors**: Generate function values sampling from a Normal distribution.
- **CreateListFactorsNK**: Creates the structure of an instance of the NK random landscape, i.e. the k neighbors for each of the variables are randomly selected.
- **CreateListFactorsCircularNK**: Creates the structure of an instance of the NK landscape where each variable depends on its $\frac{k}{2}$ previous and $\frac{k}{2}$ subsequent neighbors in a circular way.
- **CreateRandomFunctions**: Generates uniform random values for the table entries of a given structure.
- **SaveFunctionStructure**: Saves the structure of a function in the form of a factor graph [58].
- **AllNKInstances**: Generates m instances of a random NK model. The structure and function of all instances are saved in files.
- **ReadFunctionsFromData**: Reads the values of a function from a file.
- **ReadFactorGraphFromData**: Reads the structure of a given function from a file.

7 How to use MATEDA-2.0 for a given problem

7.1 Steps to run an EDA in MATEDA-2.0

In order to solve a problem using MATEDA-2.0, the following steps should be followed:

1. Create or define the file of the function to be optimized.
2. Define the type of representation to be used.
3. Create the vector with the range of values (for the continuous case) or the cardinality of the variables (for the discrete case).
4. Choose each EDA component and determine the parameters to be passed to the algorithm.
5. Execute RunEDA.m.

MATEDA-2.0 does function maximization. For minimization problems, the fitness function $\hat{f}(\mathbf{x})$ has to be modified (e.g. $f(\mathbf{x}) = -\hat{f}(\mathbf{x})$). The choice of the EDA components depends on several factors. Among them:

- Type of variable representation (discrete or continuous).
- Domain of definition for each variable (discrete problems of high cardinality may not be treated using complex models).
- Existence of prior information about the problem (e.g. when a feasible factorization is known it can be employed).
- Computational cost of the fitness function.

7.2 Examples of the optimization problems implemented

This section presents some examples of the application of MATEDA-2.0 to a number of optimization problems. Some of the problems implemented and available with the code are the following:

- `EvalIsing`: Ising model [49].
- `EvaluateEnergy`: HP simplified protein model [23].
- `EvaluateEnergyFunctional`: Functional protein model [45].
- `GaussProtein`: Off-lattice HP protein model [126].
- `EvaluateSAT`: Multi-objective max 3-satisfiability problem [118].

In the next sections we show different EDA approaches to some of these problems.

7.2.1 Tree-EDA for the Ising model

The generalized Ising model [49] is described by the energy functional (Hamiltonian):

$$E = - \sum_{i < j \in L} J_{ij} \sigma_i \sigma_j - \sum_{i \in L} h_i \sigma_i \quad (13)$$

where L is the set of sites called a lattice. Each spin variable σ_i at site $i \in L$ either takes the value 1 or -1 . One specific choice of values for the spin variables is called a configuration. The constants J_{ij} are the interaction coefficients. The ground state is the configuration with minimum energy. We address the problem of maximizing $f(x) = -E(x)$. The Ising model has been extensively employed as a testbed of EDAs [13, 47, 74, 76, 90, 121].

Example 7 describes the application of the Tree-EDA to the optimization of the Ising model. The algorithm uses the most probable configuration sampling [74] (i.e. the most probable individual from the net is added to the population during sampling). The algorithm stops when the optimum is found or the maximum number of generations is reached.

Example 7 (Tree-EDA for the Ising problem).

```

1 global lattice; % The lattice and the interactions
2 global inter; % are defined as global variables
3 Pop-Size = 500;
4 n = 64;
5 cache = [0,0,0,0,0];
```

```

6 Card = 2*ones(1,n);
7 F = 'EvalIsing'; % Ising function
8 MaxGen = 150;
9 MaxVal = 86;
10 [lattice, inter] = LoadIsing(n, 1); % The Ising instances is read
11 stop_cond_params = {MaxGen,MaxVal}; % The algorithm stops when
12 % optimum is found or maxgen is reached
   edaparams{1} = {'stop_cond_method','maxgen_maxval',stop_cond_params};
13 edaparams{2} = {'sampling_method',' SampleMPE_BN',{PopSize}};
14 [AllStat,Cache]=RunEDA(PopSize,n,F,Card,cache,edaparams);

```

The code above implements the application of the Tree-EDA to the optimization of the Ising model. A characteristic of this application, and a suggested solution for problems where some extra information is needed during the function evaluation step, is the use of global variables. For the Ising problem implementation, variables *lattice* and *inter* are accessed while reading the Ising instance from a file and during the evaluation step.

7.2.2 FDA optimization of the Hydrophobic-Polar (HP) protein model

We present in this section an FDA application to the HP protein folding problem [23]. It can be approached as a discrete problem with constraints and we use a Markov-chain-like probabilistic model as the structure of the FDA [103, 110].

The HP protein model [23] considers two types of residues: hydrophobic (H) residues and hydrophilic or polar (P) residues. In this model, a protein is considered 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).

In the optimization approach, the search for the protein structure is transformed into the search for the optimal configuration given an energy function. 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. In the linear representation of the sequence, hydrophobic residues are represented with the letter H and polar ones, with P. In the graphical representation, hydrophobic proteins are represented by black beads and polar proteins, by white beads.

In our representation, 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 $\{0, 1, \dots, z-2\}$, 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. A solution \mathbf{x} can be seen as a walk in the lattice, representing one possible folding of the protein. The codification used is called relative encoding, and has been experimentally compared to absolute encoding in [57], showing better results.

Folder `../MATEDA/functions/protein` contains an implementation of HP protein model that can be used with different EDA implementations.

Example 8 shows the code that implements the Markov chain FDA for an HP sequence of 64 residues. Other sequence configurations can be tried by modifying

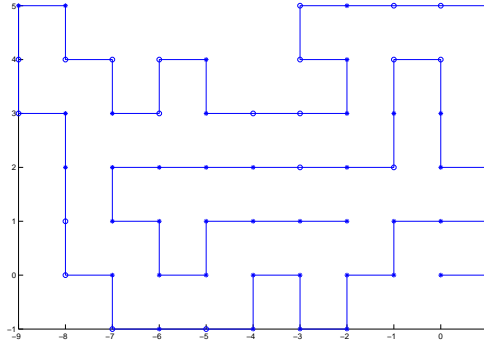


Figure 3: Best solution found by the Markov chain FDA after 100 generations.

InitConf variable. The cardinality of the variables is 3 and the Markov model considers dependence of each variable to the previous one. The fixed structure of the model (*Cliques*) is constructed using the `CreateMarkovModel` method.

The EDA uses a repairing method based on backtracking [20] which tries to enforce that each sequence is folded forming a self-avoided path in the lattice. At the end of the run, the solution found by the EDA is visualized using the method `PrintProtein`. Figure 3 shows the best solution found by one run of this algorithm after 100 generations.

Example 8 (Markov Chain FDA for the HP protein model).

```

1  global InitConf; % This is the HP protein instance
2  % defined as a sequence of zeros and ones
3  InitConf = [zeros(1,12),1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,1,0,0,1,1,
4  ,0,0,1,1,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,0,1,zeros(1,12)];
5  PopSize = 500;
6  NumVar = size(InitConf,2);
7  cache = [0,0,0,0,0];
8  Card = 3*ones(1,NumVar);
9  maxgen = 100;
10 % The Markov chain model(Cliques) is constructed specifying the number of
11 % conditioned (previous) variables. In the example below this number is
12 % 1., i.e.  $p(x) = p(x_0)p(x_1|x_0) \dots p(x_n|x_{n-1})$ 
13 Cliques = CreateMarkovModel(NumVar,1);
14 F = 'EvaluateEnergy'; % HP protein evaluation function

14 edaparams{1} = {'learning_method','LearnFDA',Cliques};
15 edaparams{2} = {'sampling_method','SampleFDA',PopSize};
16 % Repairing method used to guarantee that solutions do not self-intersect
17 edaparams{3} = {'repairing_method','HP_repairing',{}};
18 edaparams{4} = {'stop_cond_method','max_gen',{maxgen}};
19 [AllStat,Cache]=RunEDA(PopSize,NumVar,F,Card,cache,edaparams)

20 % To draw the resulting solution use function PrintProtein(vector),
21 % where vector is the best solution found.
22 vector = AllStat{maxgen,2};
23 PrintProtein(vector);

```

7.2.3 Bayesian network based EDA for a multi-objective 3-satisfiability problem

Let $\mathbf{U} = \{U_1, U_2 \dots U_n\}$ be a set of n Boolean variables. A (partial) truth assignment for \mathbf{U} is a (partial) function $T : \mathbf{U} \rightarrow \{true, false\}$. Corresponding to each variable U_i are two literals, u_i and $\neg u_i$. A literal u_i (resp. $\neg u_i$) is *true* under T iff $T(u_i) = true$ (resp. $T(u_i) = false$). A set of literals is called a clause, and a set or sequence (tuple) of clauses a formula.

Let ϕ be a formula, $\mathbf{U} = vars(\phi)$, and C a clause in ϕ . We interpret ϕ as a formula of the propositional calculus in conjunctive normal form, so that a truth assignment T for \mathbf{u} satisfies C iff at least one literal $u_i \in C$ is true under T , and T satisfies ϕ iff it satisfies every clause in ϕ . The satisfiability problem (SAT) [118] is the problem of finding a satisfying assignment for a formula.

In the multi-objective version of the SAT problem, there are m formulas (ϕ_1, \dots, ϕ_m) , each representing a different objective. This type of problems have been employed to test EDAs for multi-objective problems [6]. MATEDA-2.0 includes the method `EvaluateSAT` which evaluates a solution on a set of 3-SAT formulas contained in the global variable `Formulas`. The output is a multi-objective solution, one component corresponds to the evaluation of one formula. The method `MakeRandomFormulas` can be used to generate random instances of the multi-objective SAT problem and a number of these instances can be found in the folder `../MATEDA/functions/SAT/instances`.

Example 9 shows the code that implements a Bayesian network based EDA for multi-objective 3-SAT. The global variable `Formulas` stores the formulas that are read from the file. In this example, there are 20 binary variables, 10 formulas and 20 clauses in each formula. `ParetoRank_ordering` is used as the ordering criterion and truncation selection is applied.

Example 9 (Bayesian network based EDA for multi-objective 3-SAT).

```

1 global Formulas; % Global variable for SAT function
2 load TypeForm_1.Form_1.mat; % Multimodal SAT instance
3 n = 20; % Number of variables
4 m = 10; % Number of objectives
5 c = 20; % Number of clauses
6 PopSize = 500;
7 NumbVar = n;
8 cache = [1,1,1,1,1];
9 Card = 2*ones(1,NumbVar);
10 maxgen = 50;
11 F = 'EvaluateSAT'; % 3-SAT function
12 selparams(1:2) = {0.5,'ParetoRank_ordering'};
13 sampling_params(1:3) = {PopSize,m,Obj_Card};
14 BN_params(1:7) = {'k2',5,0.05,'pearson','bayesian','no'};
15 edaparams{1} = {'learning_method','LearnBN',BN_params};
16 edaparams{2} = {'sampling_method','SampleBN',sampling_params};
17 edaparams{3} = {'selection_method','truncation_selection',selparams};
18 edaparams{4}={'replacement_method','best_elitism',{'ParetoRank_ordering'}};
19 edaparams{5} = {'stop_cond_method','max_gen',{maxgen}};
20 [AllStat,Cache]=RunEDA(PopSize,NumbVar,F,Card,cache,edaparams);

```

7.2.4 Mixture of multivariate Gaussian distributions for a spacecraft trajectory optimization problem

Spacecraft trajectory problems can be posed as global optimization problems with constraints. One class of these problems is the multiple gravity assist

missions with the possibility of using deep space manoeuvres (MSGDSM) [129]. It consists of finding an interplanetary trajectory of a spacecraft equipped with chemical propulsion, able to thrust its engine once at any time between each trajectory leg.

The generic form of the MGADSM problem can be written as [129]:

$$\begin{aligned} & \text{find} && x \in \mathfrak{R}^n \\ & \text{minimize} && J(x) \\ & \text{subject to} && g(x) \end{aligned}$$

where x is the decision vector, J is the objective function and g are non linear constraints that may come from operational considerations or from the spacecraft system design. These problems have many local optima and constitute a formidable benchmark for the evaluation of continuous EDAs and other evolutionary algorithms.

We choose an instance of this problem corresponding to the design of a deltaV-EGA manoeuvre required to reach Jupiter using an Earth Earth Jupiter fly-by sequence with deep space manoeuvres. An EDA based on the use of a mixture of multivariate Gaussian distributions is selected to address the problem.

Example 10 shows the code of the EDA. First, a global variable that will store the information about the problem instance is defined. Then, a file containing the problem instance is loaded³. The problem has 12 continuous variables with ranges defined in *Card*. The learning parameters of the EDA model specify that: Points are clustered according to the decision variables values, the clustering method used to learn the mixture is k-means, the number of mixture components is 10 and the measure used to cluster the points is the *sqEuclidean* metric as defined in Matlab. Once the mixture of multivariate Gaussian distributions have been learned and sampled, solutions are repaired to guarantee the variables values are within the prescribed bounds.

Example 10 (Mixture of Gaussian distributions for a trajectory problem).

```

1 global MGADSMproblem % Global variable for the space trajectory problem
2 load EdEdJ; % The instance is read
3 NumbVar = 12;
4 PopSize = 5000;
5 F = 'EvalSaga';
6 Card(1,:) = [7000,0,0,0,50,300,0.01,0.01,1.05,8,-1*pi,-1*pi];
7 Card(2,:) = [9100,7,1,1,2000,2000,0.90,0.90,7.00,500,pi,pi];
8 cache = [0,0,1,0,1];
9 learning_params(1:5) = {'vars','ClusterPointsKmeans',10,'sqEuclidean',1};
10 edaparams{1} = {'learning_method','LearnMixtureofFullGaussianModels',learning_params};
11 edaparams{2} = {'sampling_method','SampleMixtureofFullGaussianModels',{PopSize,3}};
12 edaparams{3} = {'replacement_method','best_elitism',{'fitness_ordering'}};
13 selparams(1:2) = {0.1,'fitness_ordering'};
14 edaparams{4} = {'selection_method','truncation_selection',selparams};
15 edaparams{5} = {'repairing_method','SetWithinBounds_repairing',{}};
16 edaparams{6} = {'stop_cond_method','max_gen',{5000}};
17 [AllStat,Cache]=RunEDA(PopSize,NumbVar,F,Card,cache,edaparams)

```

³This instance and the Matlab method that allows the evaluation of the function can be downloaded from www.esa.int/gsp/ACT/inf/op/globopt.htm

8 Analysis of data related to the points generated and the evaluation of the (possible multiple) objective functions

In this section, we present a number of methods used to extract, process, and visualize the information gathered during the EDA evolution. This information can reveal new knowledge about the problem structure and may serve to evaluate the efficacy of the EDA approach and the role played by its different components.

Visualization techniques are an important tool for interpretation and understanding of data. Classical approaches dealing with visualization of small, isolated problems have recently shifted to the visualization of massive scale, dynamic data, comprised of elements of varying levels of certainty and abstraction [14, 128].

It is a complex task to find a clear separation between the different classes of data generated during the EDA search. Nevertheless, we use the following classification to distinguish the way the different classes of data are produced:

- Data related to the points generated and the evaluation of the (possible multiple) objective functions (e.g. number and distribution of the points generated, shape of the Pareto front approximations, etc.).
- Probabilistic models (e.g. structure and parameters of the probabilistic models).

From the previous classification, it is possible to implement different methods to extract useful information from the EDA search and visualize this information. Methods related to the first class are analyzed in this section, methods to analyze and visualize the structures and parameters of the probabilistic models are presented in the next section.

The main classes of methods for the analysis of data related to the points generated and the evaluation of the objective functions are the following:

- Computation and visualization of fitness related measures.
- Analysis and visualization of dependencies between variables and correlations between the objectives.

8.1 Fitness related measures

Fitness related measures are obtained from the fitness values of the solutions visited by the algorithm at each generation.

The average fitness (\bar{f}) of the population at each generation can be used as a source of information about the behavior of the EDA. If we are looking for a maximum, an increase in the average fitness means that the algorithm is able to generate better solutions. However, an increase of the average fitness can hide a loss of diversity in the population. In this case, the fitness variance ($\sigma(f)$) can support additional information about whether the fitness values of the population are really diverse. Similarly, the distribution of the fitness function represented using histograms gives a clearer perspective of the diversity in the population.

The response to selection ($R(t)$) is a general measure of the improvements obtained in the average fitness of the population by the application of evolutionary algorithm operators. The amount of selection allows to measure the effect of the selection operator in terms of the fitness ($S(t)$). The realized heritability ($b(t)$) can serve to evaluate the effect of the sampling and replacement methods. The mathematical framework that involves the use of $R(t)$, $S(t)$ and $b(t)$ was originally proposed in population genetics and has been applied to the analysis of the breeder genetic algorithm [81] and EDAs [75, 98, 107].

The following methods that compute fitness related measures are included in MATEDA-2.0:

- **Mean_Var_Objectives:** Computes the average fitness and variance of the objectives ($\bar{f}, \sigma(f)$).
- **ObjectiveDistribution:** Computes and visualizes the fitness distribution for a given range of generations and a subset of objectives using histograms.
- **Response_to_selection:** Computes the response to selection for every objective, $R(t) = \bar{f}(t+1) - \bar{f}(t)$.
- **Amount_of_selection:** Computes the amount of selection for every objective, $S(t) = \bar{f}_s(t) - \bar{f}(t+1)$.
- **Mean_Var_Objectives:** Computes the realized heritability for every objective, $b(t) = \frac{R(t)}{S(t)}$.

Example 11 shows the MATEDA-2.0 code for the computation of fitness related measures from the data generated by a Tree-EDA (the EDA that MATEDA-2.0 uses when no learning and sampling methods are specified).

Example 11 (Computation of fitness related measures).

```

1  PopSize = 300;
2  n = 45;
3  cache = [0,0,0,1,1];
4  Card = 2*ones(1,n);
5  F = 'sum'; % Onemax function;
6  ngen = 10;
7  edaparams{1} = {'stop_cond_method', 'max_gen', {ngen}};
8  [AllStat,Cache]=RunEDA(PopSize,n,F,Card,cache,edaparams)
9  for i=1:ngen,
10     auxr{1,i} = Cache{4,i};
11     auxs{1,i} = Cache{5,i};
12 end,
13 [mean_obj,var_obj] = Mean_Var_Objectives(auxr);
14 [RS] = Response_to_selection(auxr);
15 [S] = Amount_of_selection(auxr,auxs);
16 [b] = Realized_heritability(auxr,auxs);
17 ObjectiveDistribution(auxr,1,[1:ngen]);

```

The code presented in Example 11 starts by executing a UMDA with a maximum of 10 generations for the *Onemax* function. Since $cache = [0, 0, 0, 1, 1]$, the evaluation of the entire and selected populations for all generations are stored. From this information, auxiliary variables $auxr$ and $auxs$ are computed and from them the fitness related measures have been calculated.

8.2 *A posteriori* analysis of the dependencies

A posteriori modeling of the dependencies that arise in the selected sets (or the final Pareto sets approximations found by EDAs), can be useful to study the efficiency of EDAs that use different classes of models, to construct concise descriptions of the set of optimal solutions (e.g. graphical models of the obtained Pareto set approximations), to extract and visualize characteristic features of the selected sets at each generation, etc. We illustrate this claim in this section and explain the way in which MATEDA-2.0 allows the user to do *a posteriori* modeling of the dependencies

The most straightforward approach to modeling the dependencies between the variables is to construct a probabilistic graphical model, as usually done in *online* modeling employed by EDAs. However, a probabilistic graphical model is not the only choice available to model the dependencies in the data. For instance, a matrix of correlations between the objectives has been employed [22, 50] in multi-objective optimization to detect conflicting and redundant objectives.

Notice also, that in contrast to the *online* modeling case, in *a posteriori* modeling we do not require to model the selected sets learned at all the generations, and in general the main objective of modeling is not to sample new solutions.

The following methods have been implemented in MATEDA-2.0 to model (*a posteriori*) dependencies.

- **Corr**: This Matlab function computes the correlation matrix between a set of variables given a set of observations:
- **Methods for learning probabilistic models**: Currently, MATEDA-2.0 uses the capabilities of the Bayesian networks toolbox for representing dependencies for continuous and mixed variables. The following MATEDA-2.0 functions can be used to learn the models:
 - **LearnBN**.
 - **LearnGaussianNetwork**.
 - **LearnMixedModel**.
- **ShowParallelCoord**: Parallel coordinate visualization of the objectives.

The correlation matrix of the Pareto set approximation obtained with function **Corr** can provide an initial idea of which objectives are contradicting and redundant. However, extracting a set of non-redundant objectives from this matrix is not straightforward. Different techniques have been proposed in multi-objective optimization with this purpose [12, 22, 50, 96].

The methods used for learning the models have been analyzed in Section 5.7. Its application to *a posteriori* modeling is very similar but the user should take into account the difference between the problem objectives and the problem variables in the analysis of the model. Also, the type of representation used by objectives and variables have to be considered in the selection of the probabilistic model.

8.2.1 Parallel coordinate visualization of objectives

In parallel coordinates [48], every observation is plotted for each axis/variable, and a connecting line is drawn for each observation between all the axes. Parallel coordinates can be very effective to detect redundant and conflicting objectives. It can also be used to identify outlier instances. For example, those sets of objective values that do not follow the same trend than the rest.

An important issue in the effectiveness of the parallel coordinates approach to the visualization of multivariate data is the order and arrangement of dimensions (which in our case corresponds to the objectives). Ordering may help to reduce cluttering, improving visualization. One way to deal with this problem is to rearrange the data dimensions such that dimensions showing a similar behavior are positioned next to each other. However, the problem of finding an optimal one- or two-dimensional arrangement of the dimensions based on their similarity is NP-complete [2]. Heuristic algorithms have been proposed to find satisfying solutions [2].

In MATEDA-2.0, the method for displaying the parallel coordinates of the objectives is invoked as:

`ShowParallelCoord(fs,AllObjectives(:,ordering))`; where *fs* is the font size used in the parallel coordinates representation and *ordering* is an order of the objectives.

MATEDA-2.0 includes a function that allows the user to define a method to find an order of the objectives. The `ClusterUsingDist` method clusters together objectives with strong similarity in the Pareto set approximation (or selected population). Clustering is done according to a parameter *distance* which can take as values any of the distances used by Matlab command `pdist` (ex. 'correlation', 'euclidean', etc.). `ClusterUsingDist` uses affinity propagation [31] to cluster the objectives.

Example 12 shows how to find a Pareto set approximation using the best solutions found by an EDA. The code shown is the continuation of the code included in example 6 where the generation and optimization of a multi-objective decomposable function is presented.

Example 12 (Construction of the Pareto set approximation).

```
1 AllSols = [];  
2 AllVals = [];  
3 for i=1:MaxGen,  
4 AllSols = [AllSols;Cache{1,i}]; % All the populations  
5 AllVals = [AllVals;Cache{4,i}]; % All the evaluations  
6 end  
  
7 % The set of Pareto solutions found by the EDA is extracted  
8 Index = FindParetoSet(AllSols,AllVals);  
9 ParetoPop = AllSols(Index,:);  
10 ParetoVals = AllVals(Index,:);  
11 % The Pareto set approximation is shown using parallel coordinates  
12 parallelcoords(ParetoVals);  
13 % The correlations between the objectives are computed.  
14 ObjectivesCorr = corr(ParetoVals);  
15 % Affinity propagation is used to reduce the number of objectives  
16 % by selecting exemplars of the correlated variables  
17 [idx,netSim,dpsim,expref]=apcluster(ObjectivesCorr,mean(ObjectivesCorr));
```

The code shown above illustrates the way in which a Pareto set approximation can be obtained from the selected solutions found in every generation

of an EDA. In this example, the multi-objective function is based on the NK circular fitness landscape. Each objective corresponds to a subfunction where each variable depends on its $\frac{k}{2}$ previous and $\frac{k}{2}$ subsequent neighbors.

After the Pareto set approximation has been found, it is visualized using parallel coordinates and the correlation between its objectives are found. Finally, affinity propagation is applied to reduce 'redundant' objectives.

9 Analysis of the probabilistic models

Work on the analysis of the probabilistic models learned by EDAs and its relationship with the problem structure for single objective functions, have been accomplished for directed [4, 25, 27, 41, 43, 67, 76], undirected [101, 106, 109] and Gaussian [4] models. This work has been mainly focused on the number and frequency of the dependencies represented in the graphical structure of the model. Recent work explores other descriptors of the learned models such as the correlation between the model prediction and the fitness [13], the likelihood of the model [66], and the probabilities given by the model to some relevant points of the search [26]. MATEDA-2.0 allows the user to conduct the type of analysis achieved in previous work but also permits the detection of frequent substructures in the models learned, the visualization of interactions between edges in the models and other additional features.

9.1 Data structures to represent the models

The basic information about the interactions between the variables of a problem in a given model is represented in MATEDA-2.0 using an edge matrix E , where $E(i, j) = 1$ and $E(j, i) = 1$ if and only if variables X_i and X_j satisfy some previously defined structural relationship. Otherwise, $E(i, j) = 0$ and $E(j, i) = 0$.

The structural relationship depends on the type of probabilistic model. The following are some examples of relationships:

- Factorized model: Two variables are related if they appear together in any of the factors.
- Markov network model: Two variables are related if one of them belongs to the neighborhood of the other variable.
- Bayesian networks: Two variables are related if there is an arc between them.

Notice that, for Bayesian and Gaussian networks, we disregard the direction of the dependencies. This is a common approach in the analysis of this type of directed graphical models in EDAs [25, 27, 41, 43, 67].

Let us suppose, we have all the models learned in $nruns$ executions of an EDA with at most $maxgen$ iterations (when using MATEDA-2.0 with the option `cache(3) = 1`, all the models of a single execution are saved in the cell array `Cache{3,:}`). From this data, the data structure `run_structures` can be generated.

- $run_structures\{1\} = indexmatrix(n, n)$: Associates an index to each possible edge in the network. e.g. $indexmatrix(1, 2) = 1$, number of edges $m = \frac{n(n+1)}{2}$.
- $run_structures\{2\} = AllBigMatrices\{nruns\}(m, maxgen)$: For each run contains whether the edge e appeared in generation j .
- $run_structures\{3\} = AllSumMatrices(m, maxgen)$, i.e. the number of runs that each edge e appeared in generation j .
- $run_structures\{4\} = AllContactMatrix\{maxgen\}(n, n)$: The number of runs in which edge i, j was present in generation k .
- $run_structures\{5\} = SumAllContactMatrix(n, n)$, i.e. the total number of times edge (i, j) was present in all the structures learned in all generations of all runs.

The following programs allow to extract from the models learned by different type of EDAs the information which is stored in the *run_structures*.

- **ReadMNStructures**: Extracts the information from the Markov networks or factorization models learned during the execution of MATEDA-2.0 (i.e. Cliques saved in *Cache\{3, \}*).
- **ReadBNStructures**: Extracts the information from the Bayesian or Gaussian networks learned during the execution of MATEDA-2.0 (i.e. the models saved in *Cache\{3, \}*).
- **ReadStructures**: Extracts the information from a file with the edges corresponding to the structures learned by an arbitrary EDA.

The program **ReadStructures** is particularly useful when an implementation of EDAs not contained in MATEDA-2.0 has been applied and we want to use MATEDA-2.0 to analyze the structures. In this case, the structure must be provided in a file that contains two columns of numerical values, where a row (i, j) means that there is an edge between node $i+1$ and $j+1$ (note that vertices in the file are numbered from 0 to $NumberVar - 1$). A negative row $(-g, -r)$ indicates that the previous (positive) rows were learned in generation g of run r . Example 13 shows a file containing different structures.

Example 13 (File of some model structures learned by EDAs).

```

2 0
8 0
14 3
-1 -1
13 8
0 10
3 5
9 10
-1 -2
```

File that contains the structures learned by a Bayesian network at different generations and runs.

The structure learned at run 1, generation 1 contains edges (3, 1), (9, 1) and (15, 4). The structure learned at run 2, generation 1 contains edges (14, 9), (1, 11), (4, 6) and (10, 11).

9.2 Methods implemented for the analysis and visualization of the structures

The following methods have been implemented:

- **ViewSummStruct**: Shows one image where each edge has a color proportional to the times it has been present in the structures learned in all generations.
- **ViewInGenStruct**: Shows images where each edge has a color proportional (relative to *nruns*) to the times it has been present in the structures learned in those generations specified by the user. There is one figure for each generation.
- **ViewEdgDepStruct**: Searches for a given substructure in the set of all the structures learned, and shows the adjacency matrices corresponding to the structures that contain it. The substructure is specified by giving the edges that are present/absent in them. Different visualization options are implemented.
- **ViewPCStruct**: Searches for substructures in the set of all the structures learned and shows the parallel coordinates of the edges and the generations at which they are learned. The minimal number of times that an edge has to appear in (all) the structures learned and the method used to order the variables before displaying them using parallel coordinates are parameters of the algorithm.
- **ViewDenDroStruct**: Shows the dendrograms of the edges according to their co-occurrence in the structures learned by the EDAs. Allows to detect complex hierarchical relationships between the variables of the problem.
- **ViewGlyphStruct**: Shows the glyph representation of a subset of edges learned at a given set of runs and generations.

9.2.1 ViewSummStruct method

The type of structure shown by the **ViewSummStruct** method is usually called frequency matrix [25, 27, 106] or probabilistic coincidence matrix [42]. It serves to detect frequent patterns of interactions in the models learned during the search. Example 14 describes a case where the frequency matrix has been computed from the Bayesian network structures learned in several executions of the EBNA-Exact algorithm [25, 27].

Example 14 (Application of **ViewSummStruct** method).

```
1 NumberVar = 20;  
2 [run_structures,maxgen,nruns] = ReadStructures('ProteinStructsExR.txt',NumberVar );  
3 [results] = ViewStructures(run_structures,NumberVar ,maxgen,nruns,  
4 'viewmatrix_method','ViewSummStruct',[150,14]);
```

Figure 4 shows the image representation of the frequency matrix computed from a file containing the structures learned by EBNA-Exact in all runs, all generations. Bright colors means higher frequencies.

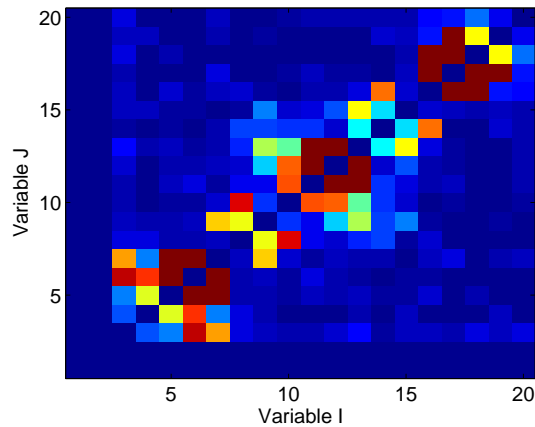


Figure 4: Frequency matrix computed from the structures learned by EBNA-Exact.

The frequency matrix representation has two main limitations. First, it is not possible to investigate the types of structures learned at each generation. It has been early observed that structures of the models learned by EDAs can change along the evolution [4]. The other limitation of this type of representation is that it is not possible to capture interactions between different substructures of the problem. For instance, if the frequency matrix shows that two edges have occurred ten times in the structures learned, we cannot determine how often they have appeared together in the same structure.

9.2.2 ViewInGenStruct method

The `ViewInGenStruct` method addresses the first limitation of the frequency matrix representation by displaying frequency matrices computed from structures learned at each generation. From a sequence of images it is then possible to distinguish the dynamics of the structures along the evolution. Example 15 shows an example of the application of the `ViewInGenStruct` method.

Example 15 (Application of `ViewInGenStruct` method).

```

1 [run_structures,maxgen,nruns] = ReadStructures('ProteinStructsExR.txt',20);
2 [results] = ViewStructures(run_structures,20,maxgen,nruns, 'viewmatrix_method',
3 'ViewInGenStruct',[150,14];[1,5,10]);

```

Figure 5 shows the output of the `ViewInGenStruct` method invoked in Example 15. The three frequency matrices shown are learned in generations 1, 5 and 10 respectively. It can be observed that the number of dependencies captured in the models diminish as the generations increase.

9.2.3 ViewEdgDepStruct method

The `ViewEdgDepStruct` serves to detect and display particular substructures learned in the models, allowing to identify complex patterns of interactions in the problem and addressing the second limitation of the `ViewSummStruct` method.

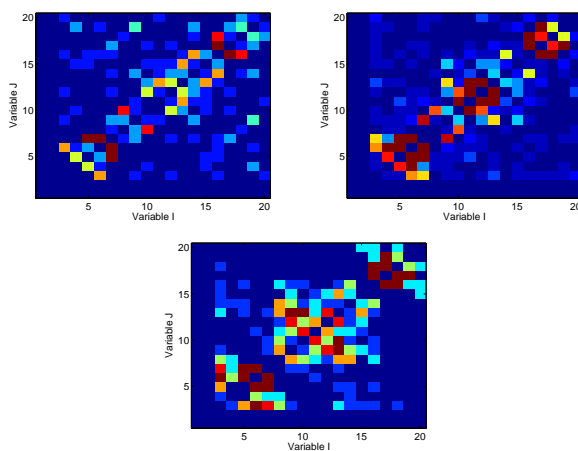


Figure 5: Frequency matrices showing the most frequent edges in the structures learned by EBNA-Exact at different generations.

For instance, the user might be interested in testing a particular hypothesis related to the mapping *problem structure* - *model structure*. By computing the frequency of appearance of the substructures in the models these hypothesis could be tested. Example 16 shows the application of the `ViewEdgDepStruct` method.

Example 16 (Application of `ViewEdgDepStruct` method).

```

1 viewparams{1} = [100,14];
2 viewparams{2} = [3 4 1; 4 5 1; 3 5 0]; % The substructure is described
3 viewparams{3} = [1:nruns]; % Selected runs (All)
4 viewparams{4} = [1:maxgen]; % Selected generations (All)
5 viewparams{5} = 'all_graphs'; % Graphs to be visualized (All)
6 [run_structures,maxgen,nruns] = ReadStructures('ProteinStructsExR.txt',20);
7 [results] = ViewStructures(run_structures,20,maxgen,nruns, 'viewmatrix.method',
8 'ViewEdgDepStruct',viewparams);

```

Figure 6 shows the four adjacency matrices of those structures learned in several runs of EBNA-Exact and that satisfy the condition that edges (3,4) and (4,5) appear together in the structure and edge (3,5) does not appear. The output variable *results* describes at which run and generation each of the visualized structures have been found.

9.2.4 ViewPCStruct method

The parallel coordinates, explained in Section 8.2.1, can also be used to visualize the structure of the models. Method `ViewPCStruct` implements a parallel coordinate visualization in which the vertical axis represents the generation at which edges (shown in the horizontal axis) have been learned. A line between two points means that both edges appear in the same structure learned at the same generation.

The method allows to select the most relevant subset of edges for representation. Parameter *const_{edg}* is the minimal number of times that an edge has to appear in (all) the structures learned to be selected for visualization. Since

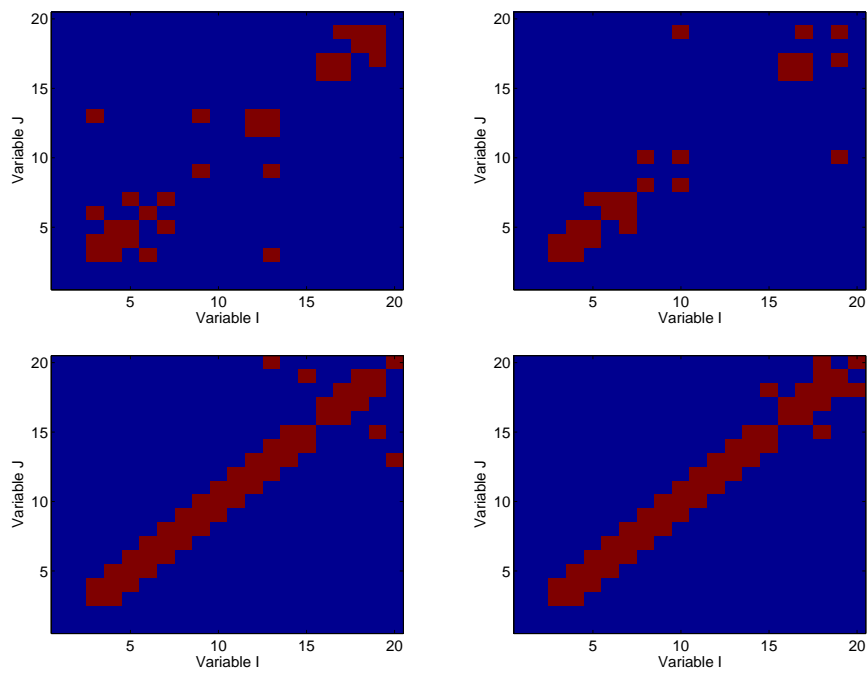


Figure 6: Edge matrices of those structures learned in several runs EBNA-Exact and such that edges $(3, 4)$ and $(4, 5)$ appear together in the structure and edge $(3, 5)$ does not appear.

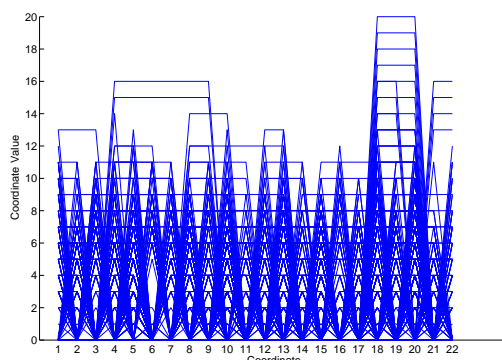


Figure 7: Parallel coordinate visualization of the most frequent edges in the structures learned by EBNA-Exact.

the clarity of the parallel coordinate visualization depends on the number of variables, this is an important parameter. Also the minimal number of edges ($min_{edg} > 0$) in the substructures selected is a parameter of the algorithm. Finally, the user can specify the method to order the variables before displaying them. `ViewPCStruct` outputs each of the represented edges together with the generation and run at which it has been learned.

Example 17 shows an example of the application of the `ViewPCStruct` method.

Example 17 (Application of `ViewPCStruct` method).

```

1 viewparams{1} = [14]; % Font size
2 viewparams{2} = []; % The edges will be found by the algorithm
3 viewparams{3} = 60; % Those edges that appear at least 60 times
4 viewparams{4} = 2; % Structures that have at least two edges
5 viewparams{5} = 'ClusterUsingDist'; % Variables are clustered using affinity prop.
6 viewparams{6} = 'correlation'; % Similarity measure used is the correlation
7 [run_structures,maxgen,nruns] = ReadStructures('ProteinStructsExR.txt',20);
8 [results] = ViewStructures(run_structures,20,maxgen,nruns,'viewmatrix_method',
9 'ViewPCStruct',viewparams);

```

Figure 7 shows the parallel coordinate visualization of the most frequent edges (they appear at least 20 times) in the structures learned in several runs of EBNA-Exact and that satisfy the condition that the structures have at least two edges.

9.2.5 ViewDendroStruct method

Dendrograms are graphs that serves to represent hierarchical trees. A dendrogram consists of many U-shaped lines connecting objects in the hierarchical tree [52]. The height of each U represents the distance between the two objects being connected.

Similarly to the `ViewPCStruct` method, `ViewDendroStruct` allows to select the most relevant subset of edges for representation. Parameter $const_{edg}$ is the minimal number of times that an edge has to appear in (all) the structures learned to be selected for visualization. Also the minimal number of edges ($min_{edg} > 0$) in the substructures selected is a parameter of the algorithm.

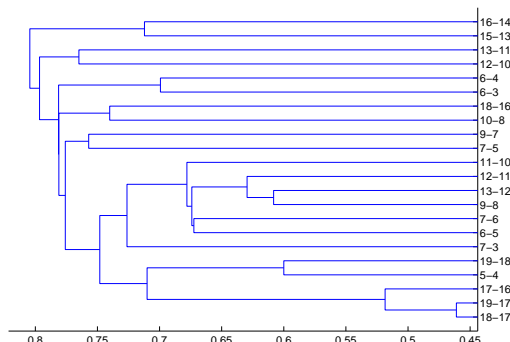


Figure 8: Dendrogram visualization of the most frequent edges in the structures learned by EBNA-Exact.

The `ViewDendroStruct` method computes first a hierarchical tree of the selected edges based on a distance defined by the user (usually the correlation of the edges in the structures learned). Then, the hierarchical tree is displayed using a dendrogram. This representation can be very effective to detect hierarchical relationships between substructures in the models. Example 18 shows an example of the application of the `ViewDendroStruct` method.

Example 18 (Application of `ViewDendroStruct` method).

```

1 viewparams{1} = [14]; % Font size
2 viewparams{2} = []; % The edges will be found by the algorithm
3 viewparams{3} = 60; % Those edges that appear at least 60 times
4 viewparams{4} = 2; % Structures that have at least two edges
5 viewparams{5} = 'correlation'; % Similarity measure used to group edges is correlation
6 [run_structures,maxgen,nruns] = ReadStructures('ProteinStructsExR.txt',20);
7 [results] = ViewStructures(run_structures,20,maxgen,nruns, 'viewmatrix_method',
8 'ViewDendroStruct',viewparams);

```

Figure 8 shows the dendrogram of the most frequent edges (they appear at least 20 times) in the structures learned in several runs of EBNA-Exact and that satisfy the condition that the structures have at least two edges. These are the same edges shown in Figure 7 using the parallel coordinate representation.

9.2.6 ViewGlyphStruct method

A Glyph is a visual representation of a piece of data where the attributes of a graphical entity (e.g. shape, size, color, and position) are dictated by one or more attributes of a data record. The placement or layout of glyphs on a display can communicate significant information regarding the data values themselves as well as relationships between data points [130]. Among the best known types of glyph graphs are star glyphs and Chernoff glyphs [16]. Glyphs have been extensively used as a visualization technique for many problems [64, 131]

Method `ViewGlyphStruct` displays the glyph representation of a subset of edges for a user defined set of runs and generations. If none of the selected edges is included (the structure learned at run i , generation j) a dot appears as the glyph representation. This representation is useful to detect common

Example 19 (Application of `ViewGlyphStruct` method).

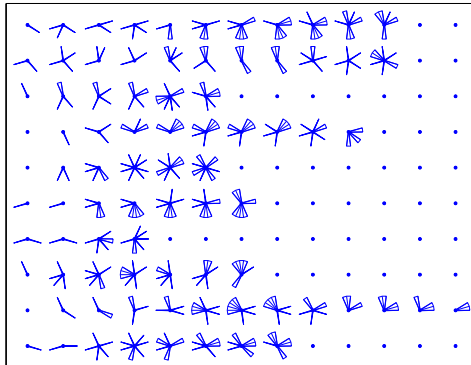


Figure 9: Glyph visualization of the most frequent edges in the structures learned by EBNA-Exact.

substructures in the models and to observe the complexity of the models learned along the generations. Example 19 shows an example of the application of the `ViewGlyphStruct` method.

```

1 viewparams{1} = [14];
2 viewparams{2} = results3; % List of edges. From results{3}, previous example.
3 viewparams{3} = [1:10]; % Selected runs that will be inspected
4 viewparams{4} = [1:13]; % Selected generations that will be inspected
5 [run_structures,maxgen,nruns] = ReadStructures('ProteinStructsExR.txt',20);
6 [newresults] = ViewStructures(run_structures,20,maxgen,nruns, 'viewmatrix_method',
7 'ViewGlyphStruct',viewparams);

```

Figure 9 shows the glyphs of the substructures formed by the most frequent edges found in the previous two examples. However, in this case only the first 10 runs and 13 generations are considered to reduce the number of structures on display.

10 Function approximation module

The goal of the function approximation module is to implement methods for using and validating the probabilistic models learned by EDAs for approximating functions. This research trend has received an increasing attention in the field of EDAs [13, 115, 116, 120, 122]. In this section, we discuss the question of function approximation with models learned by EDAs, and then we present the methods implemented in MATEDA-2.0 for this purpose.

10.1 Probabilistic models of (possibly multi-objective) fitness functions

Probabilistic models of the fitness functions can be useful in different situations:

- To create surrogated functions that help to diminish the number of evaluations for costly functions [68, 85, 115, 116, 120, 122].
- To obtain models of black box optimization problems for which an analytical expression of the fitness function is not available.
- To unveil and extract problem information that is hidden in the original formulation of the function or optimization problem [13].
- To design improved (local) optimization procedures based on the model structure [15, 94, 97, 116].

An important question is: Given two probabilistic models of a fitness function, which of the models is better? Notice that in this case, models are not intended to be evaluated in terms of its accuracy to represent the selected set (i.e. maximize the likelihood of the data). Instead, we would like to use them as sufficiently general *predictors* of the fitness function, or at least of the fitness function of sufficiently good solutions. For multi-objective problems, we can think of multi-models, each model representing a different objective.

We make a number of assumptions. First, the models have been already constructed. Actually, they are a byproduct of the EDA search. Otherwise, other machine learning techniques devised for function approximation could be used. The second assumption is that it is possible to generate a sample of solutions for which the objectives can be evaluated. We then separate the problem of finding good predictors in two subproblems:

1. Which measures should be used to evaluate the models using the solutions (with their respective objective evaluations)?
2. Which criteria should be taken into account to generate the solutions used to evaluate the models?

Different criteria can be employed to compare the models:

- Correlation between the probabilities assigned by the models to the solutions and their fitness values [13].
- Sum of the probabilities assigned to the solutions [99].
- Entropy of the model.
- Expected fitness value of the model, i.e. $\sum_x p(x)f(x)$.

These criteria will be closely related to the way in which the solutions are selected. Let us suppose the number of solutions to be generated is k . We identify as relevant the following procedures to generate them:

- The k solutions are randomly generated [13].
- The k solutions correspond to the best known values (for a single objective) of the function or are the selected solutions [99, 13].
- Solutions correspond to the k most probable configurations (MPC) of the model [26].

For random solutions, we can compare different models in terms of the correlations or the expected fitness values. In [13], the analysis of the correlation has been successfully employed to analyze the fitness modeling capabilities of EDAs based on Markov networks. The entropy of the model can also be used. A model that assigns the same or very similar probability to all the solutions is of scarce interest.

If the best known solutions are used to evaluate the models, the previous criteria can be used. However, care must be taken to detect the phenomenon of overfitting since it may have a harmful effect in EDAs [66, 99, 101]. This can be done by computing the total probability assigned to the best solutions [99]. If the sum of the probabilities given by the model to the k best solutions is very high (e.g. $\sum_x p(x) = 0.9$) then we can assume its capacity of generalization is limited.

The k most probable configurations can be computed using algorithms that employ abductive inference and dynamic programming as those presented in [47, 83, 132]. Most probable configurations have been used in different contexts in EDAs [28, 47, 74, 124].

10.2 MATEDA-2.0 methods for fitness function approximation

The following methods are implemented in MATEDA-2.0:

- **BN_Pop_Prob**: Computes the probabilities given to a set of solutions by a Bayesian network.
- **BN_Fitness_Corr**: Computes the correlation between the probabilities given to the solutions by a network and the fitness values of the solutions.
- **Find_kMPes**: Given a Bayesian network, find the k most probable configurations of the network.
- **Find_Fitness_Approx**: Finds the probabilistic model with the highest correlation for each of the objectives in the given population (e.g. a Pareto set approximation).

The implementation of several of the methods discussed in the previous section (e.g. generation of a random population, computation of the sum of probabilities, computation of the entropy, etc.) is straightforward in Matlab.

The **Find_kMPes** method is essentially the algorithm introduced by Nilsson [83]. The algorithm computes the junction tree of the BN and apply max-propagation and dynamic programming for finding the k MPCs. In [83], two schedules for finding the subsequent maxima are proposed. This implementation corresponds to the first proposed schedule. Currently, it only works for binary variables.

Find_Fitness_Approx can be directly applied to select the set of models that best approximate (in terms of the correlation values) a Pareto set approximation.

11 Conclusions

It follows a summary of the main characteristics of the MATEDA-2.0 implementation.

- It can be used for the optimization of single and multi-objective problems.
- Highly modular implementation in which each EDA component (either added by the user or already included in MATEDA-2.0) is implemented as an independent program.
- Available implementations include learning and sampling methods of undirected graphical models and Bayesian networks for problems with discrete and continuous variables.
- Knowledge extraction and visualization module for extracting and displaying information from the probabilistic models learned and the populations generated by the EDA.
- A variety of seeding, local optimization, repairing, selection and replacement methods.
- Statistical analysis of different measures of the EDA evolution.
- Extended library of functions and testbed problems.

11.1 EDAs that can be implemented with MATEDA-2.0

By combining the different implementations of the EDA components already included in MATEDA-2.0, variants of the following EDAs can be implemented:

- Univariate marginal distribution algorithm (UMDA) [80].
- Factorized distribution algorithm (FDA) [79].
- EDAs based on Bayesian networks, similar to those presented in [29, 77, 88].
- Markov chain estimation of distribution algorithm (Mk-EDA) [103].
- EDAs based on univariate and multivariate Gaussian distributions [10, 9, 61, 60].
- EDAs based on mixtures of continuous distributions [11].
- Markov optimization algorithm (MOA) [123].
- Affinity propagation EDA (Aff-EDA) [109].

In this paper we have presented MATEDA-2.0, a suite of programs in Matlab for optimization using EDAs. We expect that these programs could help to find new applications of EDAs to practical problems. The knowledge-extraction and visualizations methods introduced should be useful in extending the uses of probabilistic modeling in optimization, particularly for revealing unknown information in black-box optimization problems. In the future, we also intend to incorporate new methods for the treatment of highly complicated, mixed, constrained, and other difficult problems [111].

References

- [1] D. H. Ackley. An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, pages 170–204, 1987.
- [2] M. Ankerst, S. Berchtold, and D. A. Keim. Similarity clustering of dimensions for an enhanced visualization of multidimensional data. In *Proceedings of the 1998 IEEE Symposium on Information Visualization*, page 52, Washington, DC, USA, 1998. IEEE Computer Society.
- [3] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21. Lawrence Erlbaum Associates (Hillsdale), 1987.
- [4] E. Bengoetxea. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications. Paris, France, 2003.
- [5] R. Blanco. *Learning Bayesian Networks from Data with Factorisation and Classification Purposes. Applications in Biomedicine*. PhD thesis, University of Basque Country, Donostia, Spain, 2005.
- [6] P. A. Bosman. *Design and Application of Iterated Density-Estimation Evolutionary Algorithms*. PhD thesis, Universiteit Utrecht, Utrecht, The Netherlands, 2003.
- [7] P. A. Bosman and J. Grahl. Matching inductive search bias and problem structure in continuous estimation of distribution algorithms. *European Journal of Operational Research*, 185:1246–1264, 2008.
- [8] P. A. Bosman, J. Grahl, and D. Thierens. Enhancing the performance of maximum-likelihood Gaussian edas using anticipated mean shift. In G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, editors, *Parallel Problem Solving from Nature - PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 133–143, Dortmund, Germany, 2008. Springer.
- [9] P. A. Bosman and D. Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. In *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, September 16-20 2000. Springer. Lecture Notes in Computer Science 1917.
- [10] P. A. Bosman and D. Thierens. IDEAs based on the normal kernels probability density function. Technical Report UU-CS-2000-11, Utrecht University, 2000.
- [11] P. A. Bosman and D. Thierens. Multi-objective optimization with diversity preserving mixture-based iterated density estimation evolutionary algorithms. *International Journal of Approximate Reasoning*, 31(3):259–289, 2002.
- [12] D. Brockhoff and E. Zitzler. Dimensionality reduction in multiobjective optimization: The minimum objective subset problem. In K.-H. Waldmann and U. M. Stocker, editors, *Operation Research Proceedings 2006*.

Selected Papers of the Annual International Conference of the German Operations Research Society., pages 423–429, 2006.

- [13] S. Brownlee, J. McCall, Q. Zhang, and D. Brown. Approaches to selection and their effect on fitness modelling in an estimation of distribution algorithm. In *Proceedings of the 2008 Congress on Evolutionary Computation CEC-2008*, pages 2621–2628, Hong Kong, 2008. IEEE Press.
- [14] R. Bürger and H. Hauser. Visualization of multi-variate scientific data. In *EuroGraphics 2007 State of the Art Reports (STARs)*, pages 117–134, 2007.
- [15] M. V. Butz, M. Pelikan, X. Llorá, and D. E. Goldberg. Automated global structure extraction for effective local building block processing in XCS. *Evolutionary Computation*, 14(3):345–380, 2006.
- [16] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68:361–368, 1973.
- [17] D. Cho and B. Zhang. Evolutionary optimization by distribution estimation with mixtures of factor analyzer. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC-2002*, volume 2, pages 1397–1401. IEEE press, 2002.
- [18] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
- [19] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.
- [20] C. Cotta. Protein structure prediction using evolutionary algorithms hybridized with backtracking. In J. Mira and J. R. Alvarez, editors, *Artificial Neural Nets Problem Solving Methods*, volume 2687 of *Lecture Notes in Computer Science*, pages 321–328. Springer, 2003.
- [21] A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, (2):25–36, 1992.
- [22] K. Deb and D. K. Saxena. On finding Pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems. KanGAL Report 2005011, Kanpur Genetic Algorithms Laboratory (KanGAL). Indian Institute of Technology Kanpur, 2005.
- [23] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.
- [24] W. Dong and X. Yao. Covariance matrix repairing in Gaussian based EDAs. In *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pages 415–422, Singapore, 2007. IEEE Press.

- [25] 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.
- [26] C. Echegoyen, A. Mendiburu, R. Santana, and J. A. Lozano. Analyzing the probability of the optimum in EDAs based on Bayesian networks. In *Proceedings of the 2009 Congress on Evolutionary Computation CEC-2009*, Norway, 2009. IEEE Press.
- [27] C. Echegoyen, R. Santana, J. A. Lozano, and P. Larrañaga. The impact of probabilistic learning algorithms in EDAs based on Bayesian networks. In *Linkage in Evolutionary Computation*, Studies in Computational Intelligence, pages 109–139. Springer, 2008.
- [28] C. Echegoyen, R. Santana, A. Mendiburu, and J. A. Lozano. Estudio de la probabilidad del óptimo en edas basados en redes Bayesianas. In *Proceedings of the X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB-2009)*. Thomson, 2009.
- [29] 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.
- [30] B. Everitt and D. Hand. *Mixture Models: Inference and Applications to Clustering*. Chapman and Hall, London, 1981.
- [31] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.
- [32] N. Friedman and M. Goldszmidt. Building classifiers using Bayesian networks. In *AAAI/IAAI, Vol. 2*, pages 1277–1284, 1996.
- [33] M. R. Gallagher. *Multi-Layer Perceptron Error Surfaces: Visualization, Structure and Modelling Models for Iterative Global Optimization*. PhD thesis, University of Queensland, Queensland, Australia, 2000.
- [34] J. A. Gámez, J. L. Mateo, and J. M. Puerta. EDNA: Estimation of dependency networks algorithm. In J. Mira and J. R. Álvarez, editors, *Bio-inspired Modeling of Cognitive Tasks, Second International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2007*, volume 4527 of *Lecture Notes in Computer Science*, pages 427–436. Springer, 2007.
- [35] D. Geiger and D. Heckerman. Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence*, (82):45–74, 1996.
- [36] 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.

- [37] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [38] J. Grahl, P. A. Bosman, and F. Rothlauf. The correlation-triggered adaptive variance scaling idea. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation GECCO-2006*, pages 397–404, New York, NY, USA, 2006. ACM Press.
- [39] J. M. Hammersley and P. Clifford. Markov fields of finite graphs and lattice. Technical report, University of California-Berkeley, 1968.
- [40] G. Harik. Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.
- [41] M. Hauschild and M. Pelikan. Enhancing efficiency of hierarchical BOA via distance-based model restrictions. MEDAL Report No. 2008007, Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), April 2008.
- [42] M. Hauschild, M. Pelikan, C. Lima, and K. Sastry. Analyzing probabilistic models in hierarchical BOA on traps and spin glasses. In D. Thierens et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2007*, volume I, pages 523–530, London, UK, 2007. ACM Press.
- [43] M. Hauschild, M. Pelikan, K. Sastry, and D. E. Goldberg. Using previous models to bias structural learning in the hierarchical BOA. MEDAL Report No. 2008003, Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), 2008.
- [44] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In J. F. Lemmer and L. N. Kanal, editors, *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence*, pages 149–164. Elsevier, 1988.
- [45] J. D. Hirst. The evolutionary landscape of functional model proteins. *Protein Engineering*, 12:721–726, 1999.
- [46] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.
- [47] 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.
- [48] A. Inselberg. *Parallel Coordinates: Visual Multidimensional Geometry and its Applications*. Springer, 2007.
- [49] E. Ising. The theory of ferromagnetism. *Zeitschrift fuer Physik*, 31:253–258, 1925.

- [50] A. L. Jaimes, C. A. C. Coello, and D. Chakraborty. Objective reduction using a feature selection technique. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation GECCO-2008*, pages 673–680, New York, NY, USA, 2008. ACM.
- [51] A. Johnson and J. L. Shapiro. The importance of selection mechanisms in distribution estimation algorithms. In P. Collet, editor, *Proceedings of EA 2001*, volume 2310 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 2002.
- [52] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.
- [53] S. Kauffman. *Origins of Order*. Oxford University Press, 1993.
- [54] N. Khan. Bayesian optimization algorithms for multi-objective and hierarchically difficult problems. Master’s thesis, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2003.
- [55] N. Khan, D. E. Goldberg, and M. Pelikan. Multi-objective Bayesian optimization algorithm. IlliGAL Report No. 2002009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2002.
- [56] J. Knowles, R. Watson, and D. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer, 2001.
- [57] 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 Computation Conference*, volume 2, pages 1596–1601, Orlando, Florida, USA, 1999. Morgan Kaufmann.
- [58] F. R. Kschischang and B. J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, 16(2):219–230, 1998.
- [59] 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.
- [60] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 343–352, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

- [61] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report EHU-KZAA-IK-4/99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.
- [62] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.
- [63] S. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.
- [64] M. D. Lee, R. E. Reilly, and M. E. B. ME. An empirical evaluation of Chernoff faces, star glyphs, and spatial visualizations for binary data. In *Proceedings of the Asia-Pacific Symposium on Information Visualisation*, volume 142, pages 1–10. ACM Press, 2003.
- [65] P. Leray and O. Francois. BNT structure learning package: Documentation and experiments. Technical report, Laboratoire PSI - INSA Rouen - FRE CNRS 2645, November 2004.
- [66] C. F. Lima, F. G. Lobo, and M. Pelikan. From mating pool distributions to model overfitting. In M. Keijzer, editor, *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 431–438, New York, NY, USA, 2008. ACM.
- [67] C. F. Lima, M. Pelikan, D. E. Goldberg, F. G. Lobo, K. Sastry, and M. Hauschild. Influence of selection and replacement strategies on linkage learning in BOA. In *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pages 1083–1090. IEEE Press, 2007.
- [68] C. F. Lima, M. Pelikan, K. Sastry, M. V. Butz, D. E. Goldberg, and F. G. Lobo. Substructural neighborhoods for local search in the Bayesian optimization algorithm. In T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, and X. Yao, editors, *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN IX)*, volume 4193 of *Lecture Notes in Computer Science*, pages 232–241. Springer, 2006.
- [69] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. Springer, 2006.
- [70] S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, May 1995. Also IlliGAL Report No. 95001.
- [71] T. Mahnig and H. Mühlenbein. Comparing the adaptive Boltzmann selection schedule SDS to truncation selection. In *Evolutionary Computation and Probabilistic Graphical Models. Proceedings of the Third Symposium on Adaptive Systems (ISAS-2001)*, pages 121–128, Havana, Cuba, March 2001.

- [72] G. McLachlan and K. Basford. *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker Inc, Monticello, New York, 1988.
- [73] M. Meila and M. I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.
- [74] 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.
- [75] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.
- [76] H. Mühlenbein and R. Höns. The estimation of distributions and the minimum relative entropy principle. *Evolutionary Computation*, 13(1):1–27, 2005.
- [77] 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.
- [78] H. Mühlenbein and T. Mahnig. Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal on Approximate Reasoning*, 31(3):157–192, 2002.
- [79] H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.
- [80] 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.
- [81] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, 1(4):335–360, 1993.
- [82] K. Murphy. The BayesNet toolbox for Matlab. *Computer science and Statistics: Proceedings of Interface*, 33, 2001.
- [83] D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 2:159–173, 1998.
- [84] A. Ochoa, M. R. Soto, R. Santana, J. Madera, and N. Jorge. The factorized distribution algorithm and the junction tree: A learning perspective. In A. Ochoa, M. R. Soto, and R. Santana, editors, *Proceedings of the Second Symposium on Artificial Intelligence (CIMAF-99)*, pages 368–377, Havana, Cuba, March 1999.

- [85] Oriols-Puis, E. Bernardó-Manilla, K. Pastry, and D. E. Goldberg. Substructures surrogates for learning decomposable classification problems: Implementation and first results. pages 2875–2882, London, UK, 2007. ACE Press.
- [86] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms*. Studies in Fuzziness and Soft Computing. Springer, 2005.
- [87] M. Pelikan and D. E. Goldberg. Genetic algorithms, clustering, and the breaking of symmetry. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, pages 385–394, Paris, France, September 16-20 2000. Springer. Lecture Notes in Computer Science 1917.
- [88] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. 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 Computation Conference GECCO-1999*, volume I, pages 525–532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA.
- [89] 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.
- [90] M. Pelikan and A. K. Hartmann. Searching for ground states of Ising spin glasses with hierarchical BOA and cluster exact approximation. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pages 333–349. Springer, 2006.
- [91] M. Pelikan, K. Sastry, and E. Cantú-Paz, editors. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*. Studies in Computational Intelligence. Springer, 2006.
- [92] M. Pelikan, K. Sastry, and D. E. Goldberg. Multiobjective hBOA, clustering and scalability. IlliGAL Report No. 2005005, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, February 2005.
- [93] J. Peña, J. A. Lozano, and P. Larrañaga. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, chapter Benefits of data clustering in multimodal function optimization via EDAs, pages 99–124. Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.
- [94] F. B. Pereira, P. Machado, E. Costa, A. Cardoso, A. Ochoa, R. Santana, and M. R. Soto. Too busy to learn. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC-2000*, pages 720–727, La Jolla Marriott Hotel La Jolla, California, USA, July 2000. IEEE Press.

- [95] P. Pósiík. Preventing premature convergence in a simple EDA via global step size setting. In G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, editors, *Parallel Problem Solving from Nature - PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 549–558, Dortmund, Germany, 2008. Springer.
- [96] R. C. Purshouse and P. J. Fleming. Conflict, harmony and independence: Relationships in evolutionary multicriterion optimisation. In *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003*, volume 2632 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin-Heidelberg, 2003.
- [97] J. P. Rivera and R. Santana. Design of an algorithm based on the estimation of distributions to generate new rules in the XCS classifier system. Technical Report ICIMAF 2000-100, CEMAFIT 2000-78, Institute of Cybernetics, Mathematics and Physics, Havana, Cuba, June 2000.
- [98] R. Santana. Towards an intelligent genetic search: Defining measures of convergence. In *Proceedings of the students sessions, ACAI'99*, pages 41–42, Chania, Greece, 1999.
- [99] R. Santana. An analysis of the performance of the mixture of trees factorized distribution algorithm when priors and adaptive learning are used. Technical Report ICIMAF 2002-180, Institute of Cybernetics, Mathematics and Physics, Havana, Cuba, March 2002.
- [100] R. Santana. A Markov network based factorized distribution algorithm for optimization. In *Proceedings of the 14th European Conference on Machine Learning (ECML-PKDD 2003)*, volume 2837 of *Lecture Notes in Artificial Intelligence*, pages 337–348, Dubrovnik, Croatia, 2003. Springer.
- [101] R. Santana. Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97, 2005.
- [102] R. Santana. *Advances in Probabilistic Graphical Models for Optimization and Learning. Applications in Protein Modelling*. PhD thesis, University of the Basque Country, 2006.
- [103] R. Santana, P. Larrañaga, and J. A. Lozano. Protein folding in 2-dimensional lattices with estimation of distribution algorithms. In *Proceedings of the First International Symposium on Biological and Medical Data Analysis*, volume 3337 of *Lecture Notes in Computer Science*, pages 388–398, Barcelona, 2004. Springer.
- [104] R. Santana, P. Larrañaga, and J. A. Lozano. Interactions and dependencies in estimation of distribution algorithms. In *Proceedings of the 2005 Congress on Evolutionary Computation CEC-2005*, pages 1418–1425, Edinburgh, U.K., 2005. IEEE Press.
- [105] 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, 2006.

- [106] R. Santana, P. Larrañaga, and J. A. Lozano. The role of a priori information in the minimization of contact potentials by means of estimation of distribution algorithms. In E. Marchiori, J. H. Moore, and J. C. Rajapakse, editors, *Proceedings of the Fifth European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 4447 of *Lecture Notes in Computer Science*, pages 247–257. Springer, 2007.
- [107] R. Santana, P. Larrañaga, and J. A. Lozano. Adaptive estimation of distribution algorithms. In C. Cotta, M. Sevaux, and K. Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 177–197. Springer, 2008.
- [108] R. Santana, P. Larrañaga, and J. A. Lozano. Combining variable neighborhood search and estimation of distribution algorithms in the protein side chain placement problem. *Journal of Heuristics*, 14:519–547, 2008.
- [109] R. Santana, P. Larrañaga, and J. A. Lozano. Estimation of distribution algorithms with affinity propagation methods. Technical Report EHU-KZAA-IK-1/08, Department of Computer Science and Artificial Intelligence, University of the Basque Country, January 2008.
- [110] R. Santana, P. Larrañaga, and J. A. Lozano. Protein folding in simplified models with estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 12(4):418–438, 2008.
- [111] R. Santana, P. Larrañaga, and J. A. Lozano. Research topics on discrete estimation of distribution algorithms. *Memetic Computing*, 1(1):35–54, 2009.
- [112] R. Santana and A. Ochoa. Dealing with constraints with estimation of distribution algorithms: The univariate case. In A. Ochoa, M. R. Soto, and R. Santana, editors, *Proceedings of the Second Symposium on Artificial Intelligence (CIMA-F-99)*, pages 378–384, Havana, Cuba, March 1999.
- [113] R. Santana, A. Ochoa, and M. R. Soto. Factorized Distribution Algorithms for functions with unitation constraints. In *Evolutionary Computation and Probabilistic Graphical Models. Proceedings of the Third Symposium on Adaptive Systems (ISAS-2001)*, pages 158–165, Havana, Cuba, March 2001.
- [114] R. Santana, A. Ochoa, and M. R. Soto. The mixture of trees factorized distribution algorithm. 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 543–550, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
- [115] K. Sastry, M. Pelikan, and D. Goldberg. Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation. In *Proceedings of the 2004 Congress on Evolutionary Computation CEC-2004*, pages 720–727, Portland, Oregon, 2004. IEEE Press.

- [116] K. Sastry, M. Pelikan, and D. E. Goldberg. Efficiency enhancement of estimation of distribution algorithms. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pages 161–186. Springer, 2006.
- [117] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 7(2):461–464, 1978.
- [118] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, USA, 1992.
- [119] R. Shachter and C. Kenley. Gaussian influence diagrams. *Management Science*, 35:527–550, 1989.
- [120] S. Shakya. *DEUM: A framework for an Estimation of Distribution Algorithm based on Markov Random Fields*. PhD thesis, The Robert Gordon University. School of Computing, Aberdeen, UK, 2006.
- [121] S. Shakya and J. McCall. Optimization by estimation of distribution with DEUM framework based on Markov random fields. *International Journal of Automation and Computing*, 4(3):262–272, 2007.
- [122] 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.
- [123] S. Shakya and R. Santana. An EDA based on local Markov property and Gibbs sampling. In M. Keijzer, editor, *Proceedings of the 2008 Genetic and evolutionary computation conference (GECCO)*, pages 475–476, New York, NY, USA, 2008. ACM.
- [124] 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.
- [125] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*, volume 81 of *Lecture Notes in Statistics*. Springer, New York, 1993.
- [126] F. Stillinger, T. Head-Gordon, and C. Hirshfeld. Toy model for protein folding. *Physical Review E*, 48:1469–1477, 1993.
- [127] D. Thierens and P. A. Bosman. Multi-objective mixture-based iterated density estimation evolutionary algorithms. 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 663–670, San Francisco, CA, 2001. Morgan Kaufmann Publishers.

- [128] J. J. Thomas and K. A. Cook. A visual analytics agenda. *IEEE Computer Graphics and Applications*, 26(1):10–13, 2006.
- [129] T. Vinko, D. Izzo, and C. Bombardelli. Benchmarking different global optimisation techniques for preliminary space trajectory design. In *Proceedings of 58th International Astronautical Congress*, 2007.
- [130] M. O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1(3/4):194–210, 2002.
- [131] C. M. Wittenbrink, A. T. Pang, and S. K. Lodha. Glyphs for visualizing uncertainty in vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):266–279, 1996.
- [132] C. Yanover and Y. Weiss. Finding the M most probable configurations using loopy belief propagation. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.
- [133] Q. Zhang, J. Sun, and E. P. K. Tsang. Evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2):192–200, 2005.