copulaedas: An **R** Package for Estimation of Distribution Algorithms Based on Copulas

Yasser Gonzalez-Fernandez
Institute of Cybernetics,
Mathematics and Physics

Marta Soto
Institute of Cybernetics,
Mathematics and Physics

Abstract

The use of copula-based models in EDAs (estimation of distribution algorithms) is currently an active area of research. In this context, the **copulaedas** package for R provides a platform where EDAs based on copulas can be implemented and studied. The package offers complete implementations of various EDAs based on copulas and vines, a group of well-known optimization problems, and utility functions to study the performance of the algorithms. Newly developed EDAs can be easily integrated into the package by extending an S4 class with generic functions for their main components. This paper presents **copulaedas** by providing an overview of EDAs based on copulas, a description of the implementation of the package, and an illustration of its use through examples. The examples include running the EDAs defined in the package, implementing new algorithms, and performing an empirical study to compare the behavior of different algorithms on benchmark functions and a real-world problem.

Keywords: black-box optimization, estimation of distribution algorithm, copula, vine, R.

1. Introduction

The field of numerical optimization (see e.g., Nocedal and Wright 1999) is a research area with a considerable number of applications in engineering, science, and business. Many mathematical problems involve finding the most favorable configuration of a set of parameters that achieve an objective quantified by a function. Numerical optimization entails the case where these parameters can take continuous values, in contrast with combinatorial optimization, which involves discrete variables. The mathematical formulation of a numerical optimization problem is given by $\min_{\boldsymbol{x}} f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^n$ is a real vector with $n \geq 1$ components and $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function (also known as the fitness, loss or cost function).

In particular, we consider within numerical optimization a black-box (or direct-search) scenario where the function values of evaluated search points are the only available information on f. The algorithms do not assume any knowledge of the function f regarding continuity, the existence of derivatives, etc. A black-box optimization procedure explores the search space by generating solutions, evaluating them, and processing the results of this evaluation in order to generate new promising solutions. In this context, the performance measure of the algorithms is generally the number of function evaluations needed to reach a certain value of f.

Algorithms that have been proposed to deal with this kind of optimization problems can be classified in two groups according to the approach followed for the generation of new solutions. On the one hand, deterministic direct search algorithms, such as the Hooke-Jeeves (Hooke and Jeeves 1961) and Nelder-Mead (Nelder and Mead 1965) methods, perform transformations to one or more candidate solutions at each iteration. Given their deterministic approach, these algorithms may have limited global search capabilities and can get stuck in local optima, depending on an appropriate selection of the initial solutions. On the other hand, randomized optimization algorithms offer an alternative to ensure a proper global exploration of the search space. Examples of these algorithms are simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983), evolution strategies (see e.g., Beyer and Schwefel 2002), particle swarm optimization (Kennedy and Eberhart 1995), and differential evolution (Storn and Price 1997).

In this paper, we focus on EDAs (estimation of distribution algorithms; Mühlenbein and Paaß 1996; Baluja 1994; Larrañaga and Lozano 2002; Pelikan, Goldberg, and Lobo 2002), which are stochastic black-box optimization algorithms characterized by the explicit use of probabilistic models to explore the search space. These algorithms combine ideas from genetic and evolutionary computation, machine learning, and statistics into an optimization procedure. The search space is explored by iteratively estimating and sampling from a probability distribution built from promising solutions, a characteristic that differentiates EDAs among other randomized optimization algorithms. One key advantage of EDAs is that the search distribution may encode probabilistic dependences between the problem variables that represent structural properties of the objective function, performing a more effective optimization by using this information.

Due to its tractable properties, the normal distribution has been commonly used to model the search distributions in EDAs for real-valued optimization problems (Bosman and Thierens 2006; Kern, Müller, Hansen, Büche, Ocenasek, and Koumoutsakos 2003). However, once a multivariate normal distribution is assumed, all the margins are modeled with the normal density and only linear correlation between the variables can be considered. These characteristics could lead to the construction of incorrect models of the search space. For instance, the multivariate normal distribution cannot represent properly the fitness landscape of multimodal objective functions. Also, the use of normal margins imposes limitations on the performance when the sample of the initial solutions is generated asymmetrically with respect to the optimum of the function (see Soto, Gonzalez-Fernandez, and Ochoa 2014 for an illustrative example of this situation).

Copula functions (see e.g., Joe 1997; Nelsen 2006) offer a valuable alternative to tackle these problems. By means of Sklar's Theorem (Sklar 1959), any multivariate distribution can be decomposed into the (possibly different) univariate marginal distributions and a multivariate copula that determines the dependence structure between the variables. EDAs based on copulas inherit these properties, and consequently, can build more flexible search distributions that may overcome the limitations of a multivariate normal probabilistic model.

The advantages of using copula-based search distributions in EDAs extend further with the possibility of factorizing the multivariate copula with the copula decomposition in terms of lower-dimensional copulas. Multivariate dependence models based on copula factorizations, such as nested Archimedean copulas (Joe 1997) and vines (Joe 1996; Bedford and Cooke 2001; Aas, Czado, Frigessi, and Bakken 2009), provide great advantages in high dimensions. Particularly in the case of vines, a more appropriate representation of multivariate distributions having pairs of variables with different types of dependence is possible.

Although various EDAs based on copulas have been proposed in the literature, as far as we know there are no publicly available implementations of these algorithms (see Santana 2011 for a comprehensive review of EDA software). Aiming to fill this gap, the **copulaedas** package (Gonzalez-Fernandez and Soto 2014a) for the R language and environment for statistical computing (R Core Team 2014) has been published on the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=copulaedas`. This package provides a modular platform where EDAs based on copulas can be implemented and studied. It contains various EDAs based on copulas, a group of well-known benchmark problems, and utility functions to study EDAs. One of the most remarkable features of the framework offered by **copulaedas** is that the components of the EDAs are decoupled into separated generic functions, which promotes code factorization and facilitates the implementation of new EDAs that can be easily integrated into the framework.

The remainder of this paper provides a presentation of the **copulaedas** package organized as follows. Section 2 continues with the necessary background on EDAs based on copulas. Next, the details of the implementation of **copulaedas** are described in Section 3, followed by an illustration of the use of the package through examples in Section 4. Finally, concluding remarks are given in Section 5.

2. Estimation of distribution algorithms based on copulas

This section begins by describing the general procedure of an EDA, according to the implementation in **copulaedas**. Then, we present an overview of the EDAs based on copulas proposed in the literature with emphasis on the algorithms implemented in the package.

2.1. General procedure of an EDA

The procedure of an EDA is built around the concept of performing the repeated refinement of a probabilistic model that represents the best solutions of the optimization problem. A typical EDA starts with the generation of a population of initial solutions sampled from the uniform distribution over the admissible search space of the problem. This population is ranked according to the value of the objective function and a subpopulation with the best solutions is selected. The algorithm then constructs a probabilistic model to represent the solutions in the selected population and new offspring are generated by sampling the distribution encoded in the model. This process is repeated until some termination criterion is satisfied (e.g., when a sufficiently good solution is found) and each iteration of this procedure is called a generation of the EDA. Therefore, the feedback for the refinement of the probabilistic model comes from the best solutions sampled from an earlier probabilistic model.

Let us illustrate the basic EDA procedure with a concrete example. Figure 1 shows the steps performed to minimize the two-dimensional objective function $f(x_1, x_2) = x_1^2 + x_2^2$
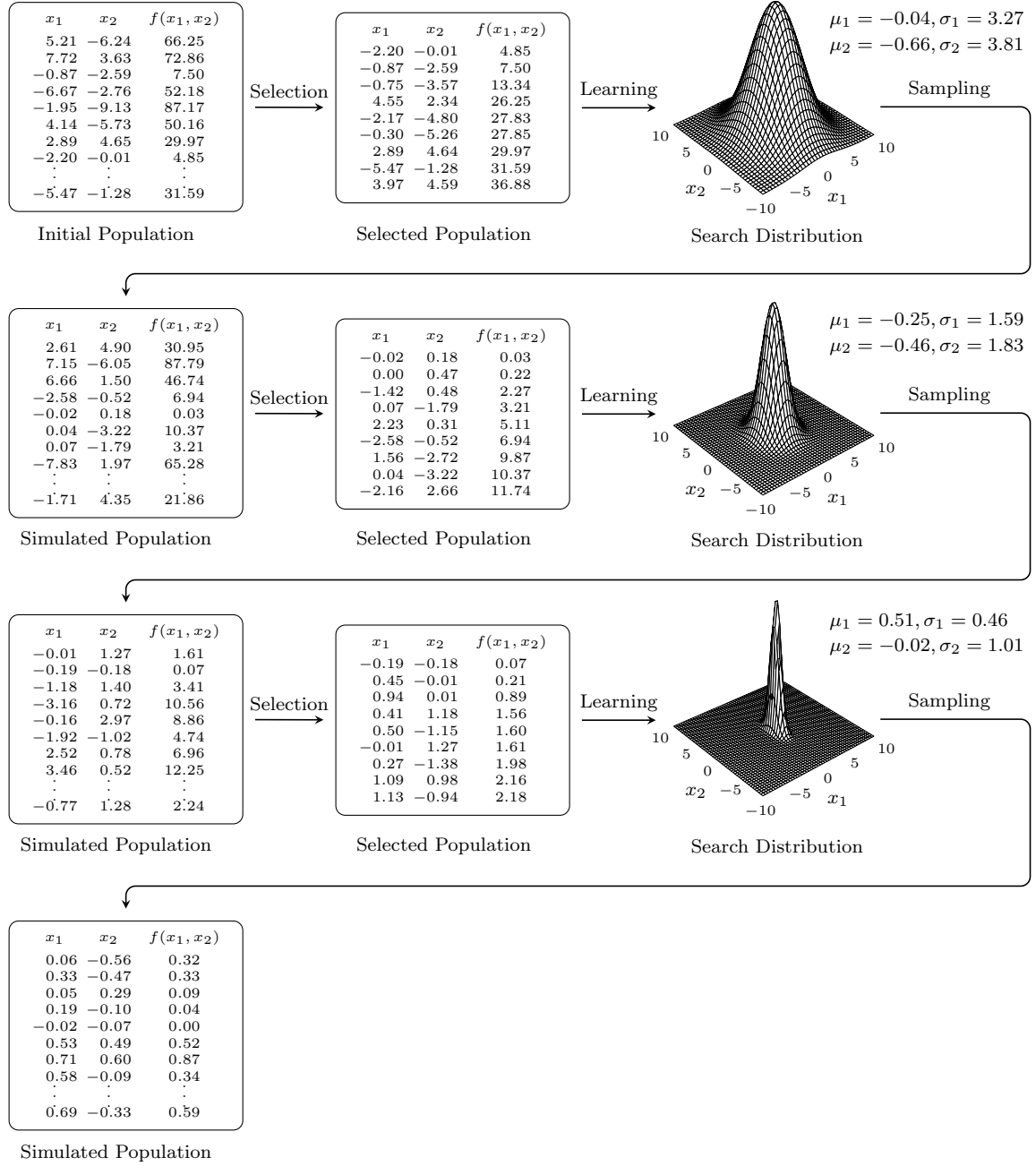
Figure 1: Steps performed to minimize the function $f(x_1, x_2) = x_1^2 + x_2^2$ using a continuous EDA that assumes independence between the variables. The search distribution models each variable with the normal distribution and, since mutual independence is assumed, the joint PDF is factorized as $\phi_{1,2}(x_1, x_2) = \phi_1(x_1, \mu_1, \sigma_1^2)\phi_2(x_2, \mu_2, \sigma_2^2)$. The simulation of the probabilistic model estimated at the third generation produces a solution $f(-0.02, -0.07) = 0.00$ that approximates the global optimum of the function with the required precision.

$i \leftarrow 1$
repeat
    if $i = 1$ **then**
        Generate an initial population P_1 using a *seeding method*.
        Evaluate the solutions in the population P_1.
        If required, apply a *local optimization method* to the population P_1.
    else
        Select a population $P_i^{Selected}$ from P_{i-1} according to a *selection method*.
        Learn a probabilistic model M_i from $P_i^{Selected}$ using a *learning method*.
        Sample a new population $P_i^{Sampled}$ from M_i using a *sampling method*.
        Evaluate the solutions in the population $P_i^{Sampled}$.
        If required, apply a *local optimization method* to the population $P_i^{Sampled}$.
        Create the population P_i from P_{i-1} and $P_i^{Sampled}$ using a *replacement method*.
    end if
    If required, report progress information using a *reporting method*.
    $i \leftarrow i + 1$
until A criterion of the *termination method* is met.

Algorithm 1: Pseudocode of an EDA.

using a simple continuous EDA that assumes independence between the problem variables. Specifically, we aim to find the global optimum of the function $f(0,0) = 0$ with a precision of two decimal places.

The algorithm starts by generating an initial population of 30 candidate solutions from a continuous uniform distribution in $[-10, 10]^2$. Out of this initial sampling, the best solution found so far is $f(-2.20, -0.01) = 4.85$. Next, the initial population is ranked according to their evaluation in $f(x_1, x_2)$, and the best 30% of the solutions is selected to estimate the probabilistic model. This EDA factorizes the joint probability density function (PDF) of the best solutions as $\phi_{1,2}(x_1, x_2) = \phi_1(x_1, \mu_1, \sigma_1^2)\phi_2(x_2, \mu_2, \sigma_2^2)$, which describes mutual independence, and where ϕ_1 denotes the univariate normal PDF of x_1 with mean μ_1 and variance σ_1^2, and ϕ_1 denotes the univariate normal PDF of x_2 with mean μ_1 and variance σ_2^2. In the first generation, the parameters of the probabilistic model are $\mu_1 = -0.04$, $\sigma_1 = 3.27$, $\mu_2 = -0.66$ and $\sigma_2 = 3.81$. The second generation starts with the simulation of a new population from the estimated probabilistic model. Afterwards, the same selection procedure is repeated and the resulting selected population is used to learn a new probabilistic model. These steps are then repeated for a third generation.

Notice how in the first three generations the refinement of the probabilistic model that represents the best solutions is evidenced in the reduction of the variance of the marginal distributions towards a mean value around zero. Also, the convergence of the algorithm is reflected in the reduction of the value of the objective function from one generation to another. Ultimately, the simulation of the probabilistic model estimated at the third generation produces $f(-0.02, -0.07) = 0.00$, which satisfies our requirements and the algorithm terminates.

In practice, EDAs include other steps in addition to the ones illustrated in the previous basic example. The general procedure of an EDA implemented in **copulaedas** is outlined in Algorithm 1. In the following, we provide a description of the purpose of the main steps of

the algorithm, which are highlighted in italics in the pseudocode:

- The first step is the generation of an initial population of solutions following a *seeding method*, which is usually random, but it can use a particular heuristic when a priori information about the characteristics of the problem is available.

- The results of global optimization algorithms such as EDAs can often be improved if combined with *local optimization methods* that look for better solutions in the neighborhood of each candidate solution. Local optimization methods can also be used to implement repairing methods for constrained problems where the simulated solutions may be unfeasible and a strategy to repair these solutions is available.

- A *selection method* is used to determine the most promising solutions of the population. An example selection method is truncation selection, which creates the selected population with a percentage of the best solutions of the current population.

- The estimation and simulation of the search distribution are the essential steps of an EDA. These steps are implemented by *learning and sampling methods*, which are tightly related. Learning methods estimate the structure and parameters of the probabilistic model used by the algorithm from the selected population, while sampling methods are used to generate new solutions from the estimated probabilistic model.

- A *replacement method* is used to incorporate a new group of solutions into the current population. For example, a replacement strategy is to substitute the current population with the newly sampled population. Other replacement strategies retain the best solutions found so far or try to maintain the diversity of solutions.

- *Reporting methods* provide progress information during the execution of the EDA. Relevant progress information can be the number of evaluations of the objective function and the best solution found so far.

- A *termination method* determines when the algorithm stops according to certain criteria; for example, when a fixed number of function evaluations are realized or a certain value of the objective function is reached.

Although it was possible to locate with the required precision the optimum of the simple function presented in this section, it is not always possible to perform a successful search by considering only the marginal information. As we show later in this paper, the assumption of independence between the variables constitutes a strong limitation that may compromise the convergence of an EDA. The use of information about the relationships between the variables allows searching efficiently for the best solutions and it constitutes one of the main advantages of EDAs. Among the algorithms that consider dependences between the variables, we are particularly interested in EDAs whose learning and sampling steps involve probabilistic models based on copulas. The next section provides an overview of such algorithms.

2.2. Overview of EDAs based on copulas

To the best of our knowledge, the technical report (Soto, Ochoa, and Arderí 2007) and the theses (Arderí 2007; Barba-Moreno 2007) constitute the first attempts to incorporate copulas

into EDAs. Since then, a considerable number of EDAs based on copula theory have been proposed in the literature and, as evidence of its increasing popularity, the use of copulas in EDAs has been identified as an emerging approach for the solution of real-valued optimization problems (Hauschild and Pelikan 2011).

The learning step of copula-based EDAs consists of two tasks: the estimation of the marginal distributions and the estimation of the probabilistic dependence structure. In general, these tasks have been performed by following one of the two-step estimation procedures known in the copula literature as the IFM (inference functions for margins; Joe and Xu 1996; Joe 2005) and the semiparametric estimation method (Genest, Ghoudi, and Rivest 1995). Firstly, the marginal distributions are estimated and the selected population is transformed into uniformly distributed variables in $(0, 1)$ by means of the evaluation of each marginal cumulative distribution function. Secondly, the transformed population is used to estimate a copula-based model of the dependence structure among the variables. Usually, a particular parametric distribution (e.g., normal or beta) is assumed for each margin and its parameters are estimated by maximum likelihood (see e.g., Soto *et al.* 2007; Salinas-Gutiérrez, Hernández-Aguirre, and Villa-Diharce 2009). In other cases, empirical marginal distributions or kernel density estimation have been used (see e.g., Soto *et al.* 2007; Gao 2009; Cuesta-Infante, Santana, Hidalgo, Bielza, and naga 2010). The simulation step typically starts with the generation of a population of uniformly distributed variables in $(0, 1)$ with the dependence structure described by the copula-based model that was estimated in the learning step. Finally, this uniform population is transformed to the domain of the variables through the evaluation of the inverse of each marginal cumulative distribution function.

According to the copula model being used, EDAs based on copulas can be classified as EDAs based on either multivariate or factorized copulas. In the rest of this section we give an overall description of representative algorithms belonging to each group that have been proposed in the literature.

EDAs based on multivariate copulas

The research on EDAs based on multivariate copulas has focused on the use of multivariate elliptical copulas (Abdous, Genest, and Rémillard 2005; Fang, Fang, and Kotz 2002) and Archimedean copulas (Joe 1997; McNeil and Nešlehová 2009). The algorithms described in Soto *et al.* (2007); Arderí (2007) and Barba-Moreno (2007) are both based on the multivariate normal copula and theoretically similar, but they present differences in the estimation of the marginal distributions and the use of techniques such as variance scaling. Wang, Zeng, and Hong (2009b) present an EDA based on the bivariate normal copula and, since only normal marginal distributions are used, the proposed algorithm is equivalent to EMNA (estimation of multivariate normal algorithm; Larrañaga, Lozano, and Bengoetxea 2001). On the other hand, the algorithms presented in Wang, Zeng, and Hong (2009a) and Gao (2009) use exchangeable Archimedean copulas. Wang *et al.* (2009a) propose two algorithms that use Clayton and Ali-Mikhail-Haq copulas with fixed parameters, while Gao (2009) does not state which particular members of the family of Archimedean copulas are used.

Two EDAs based on multivariate copulas are implemented in **copulaedas**, one is based on the product or independence copula and the other on the normal copula. The first algorithm is UMDA (univariate marginal distribution algorithm) for continuous variables (Larrañaga, Etxeberria, Lozano, and Peña 1999, 2000), which can be integrated into the framework of

copula-based EDAs although originally it was not defined in terms of copulas. A consequence of Sklar's Theorem is that random variables are independent if and only if the underlying copula is the product copula. Thus, UMDA can be described as an EDA that models the dependence structure between the variables using a multivariate product copula.

The second EDA based on a multivariate copula implemented in **copulaedas** is GCEDA (Gaussian copula estimation of distribution algorithm; Soto *et al.* 2007; Arderí 2007). This algorithm is based on the multivariate normal copula, which allows the construction of multivariate distributions with normal dependence structure and non-normal margins. The dependence structure of the multivariate normal copula is determined by a positive-definite correlation matrix. If the marginal distributions are not normal, the correlation matrix is estimated through the inversion of the non-parametric estimator of Kendall's tau for each pair of variables (see e.g., Genest and Favre 2007; Hult and Lindskog 2002). If the resulting matrix is not positive-definite, the transformation proposed by Rousseeuw and Molenberghs (1993) is applied. GCEDA is equivalent to EMNA when all the marginal distributions are normal.

EDAs based on copula factorizations

The use of multivariate copulas to model the dependence structure between the variables offers a considerable number of advantages over the use of the multivariate normal distribution; nevertheless, it presents limitations. The number of tractable copulas available when more than two variables are involved is limited, most available copulas are just investigated in the bivariate case. In addition, the multivariate elliptical copulas might not be appropriate when all pairs of variables do not have the same dependence structure. Another limitation is that some multivariate extensions, such as exchangeable Archimedean copulas or the multivariate t copula, have only one parameter to describe certain aspects of the overall dependence. This characteristic can be a serious limitation when the type and strength of the dependence is not the same for all pairs of variables. One alternative to these limitations is to use copula factorizations that build high-dimensional probabilistic models by using lower-dimensional copulas as building blocks. Several EDAs based on copula factorizations, such as nested Archimedean copulas (Joe 1997) and vines (Joe 1996; Bedford and Cooke 2001; Aas *et al.* 2009), have been proposed in the literature.

The EDA introduced in Salinas-Gutiérrez *et al.* (2009) is an extension of MIMIC (mutual information maximization for input clustering) for continuous domains (Larrañaga *et al.* 1999, 2000) that uses bivariate copulas in a chain structure instead of bivariate normal distributions. Two instances of this algorithm were presented, one uses normal copulas and the other Frank copulas. In Section 4.2, we illustrate the implementation of this algorithm using **copulaedas**.

The exchangeable Archimedean copulas employed in Wang *et al.* (2009a) and Gao (2009) represent highly specialized dependence structures (Berg and Aas 2007; McNeil 2008). Within the domain of Archimedean copulas, nested Archimedean copulas provide a more flexible alternative to build multivariate copula distributions. In particular, hierarchically nested Archimedean copulas present one of the most flexible solutions among the different nesting structures that have been studied (see e.g., Berg and Aas 2007 for a review). Building from these models, Ye, Gao, Wang, and Zeng (2010) propose an EDA that uses a representation of hierarchically nested Archimedean copulas based on Lévy subordinators (Hering, Hofert, Mai, and Scherer 2010).

Cuesta-Infante *et al.* (2010) investigate the use of bivariate empirical copulas and a multi-

variate extension of Archimedean copulas. The EDA based on bivariate empirical copulas is completely nonparametric: it employs empirical marginal distributions and a construction based on bivariate empirical copulas to represent the dependence between the variables. The marginal distributions and the bivariate empirical copulas are defined through the linear interpolation of the sample in the selected population. The EDA based on Archimedean copulas uses a construction similar to a fully nested Archimedean copula and uses copulas from one of the families Frank, Clayton or HRT (i.e., heavy right tail copula or Clayton survival copula). The parameters of the copulas are fixed to a constant value, i.e., not estimated from the selected population. The marginal distributions are modeled as in the EDA based on bivariate empirical copulas.

The class of VEDAs (vine EDAs) is introduced in Soto and Gonzalez-Fernandez (2010) and Gonzalez-Fernandez (2011). Algorithms of this class model the search distributions using regular vines, which are graphical models that represent a multivariate distribution by decomposing the corresponding multivariate density into conditional bivariate copulas, unconditional bivariate copulas and univariate densities. In particular, VEDAs are based on the simplified pair-copula construction (Haff, Aas, and Frigessi 2010), which assumes that the bivariate copulas depend on the conditioning variables only through their arguments. Since all bivariate copulas do not have to belong to the same family, regular vines model a rich variety of dependences by combining bivariate copulas from different families.

A regular vine on n variables is a set of nested trees $T_1, \ldots, T_{n-1}$, where the edges of tree T_j are the nodes of the tree T_{j+1} with $j = 1, \ldots, n - 2$. The edges of the trees represent the bivariate copulas in the decomposition and the nodes their arguments. Moreover, the proximity condition requires that two nodes in tree T_{j+1} are joined by an edge only if the corresponding edges in T_j share a common node. C-vines (canonical vines) and D-vines (drawable vines) are two particular types of regular vines, each of which determines a specific decomposition of the multivariate density. In a C-vine, each tree T_j has a unique root node that is connected to $n - j$ edges. In a D-vine, no node is connected to more than two edges. Two EDAs based on regular vines are presented in Soto and Gonzalez-Fernandez (2010) and Gonzalez-Fernandez (2011): CVEDA (C-vine EDA) and DVEDA (D-vine EDA) based on C-vines and D-vines, respectively. Since both algorithms are implemented in **copulaedas**, we describe them in more detail in the rest of this section.

The general idea of the simulation and inference methods for C-vines and D-vines was developed by Aas *et al.* (2009). The simulation algorithm is based on the conditional distribution method (see e.g., Devroye 1986), while the inference method should consider two main aspects: the selection of the structure of the vines and the choice of the bivariate copulas. In the rest of this section we describe how these aspects are performed in the particular implementation of CVEDA and DVEDA.

The selection of the structure of C-vines and D-vines is restricted to the selection of the bivariate dependences explicitly modeled in the first tree. This is accomplished by using greedy heuristics, which use the empirical Kendall's tau assigned to the edges of the tree. In a C-vine, the node that maximizes the sum of the weights of its edges to the other nodes is chosen as the root of the first tree and a canonical root node is assumed for the rest of the trees. In a D-vine, the construction of the first tree consists of finding the maximum weighted sequence of the variables, which can be transformed into a TSP (traveling salesman problem) instance (Brechmann 2010). For efficiency reasons, in **copulaedas** we find an approximate solution of the TSP by using the cheapest insertion heuristic (Rosenkrantz, Stearns, and

Lewis, II 1977).

The selection of each bivariate copula in both CVEDA and DVEDA starts with an independence test (Genest and Rémillard 2004; Genest, Quessy, and Rémillard 2007). The product copula is selected when there is not enough evidence against the null hypothesis of independence at a given significance level. Otherwise, the parameters of a group of candidate copulas are estimated and the copula that minimizes a Cramér-von Mises statistic of the empirical copula is selected (Genest and Rémillard 2008).

The cost of the construction of C-vines and D-vines increases with the number of variables. To reduce this cost, we apply the truncation strategy presented in Brechmann (2010), for which the theoretical justification can be found in Joe, Li, and Nikoloulopoulos (2010). When a vine is truncated at a certain tree during the tree-wise estimation procedure, all the copulas in the subsequent trees are assumed to be product copulas. A model selection procedure based on either AIC (Akaike information criterion; Akaike 1974) or BIC (Bayesian information criterion; Schwarz 1978) is applied to detect the required number of trees. This procedure expands the tree T_{j+1} if the value of the information criterion calculated up to the tree T_{j+1} is smaller than the value obtained up to the previous tree; otherwise, the vine is truncated at the tree T_j. At this point, it is important to note that the algorithm presented in Salinas-Gutiérrez, Hernández-Aguirre, and Villa-Diharce (2010) also uses a D-vine. In this algorithm only normal copulas are fitted in the first two trees and conditional independence is assumed in the rest of the trees, i.e., the D-vine is always truncated at the second tree.

The implementation of CVEDA and DVEDA included in **copulaedas** uses by default the truncation procedure based on AIC and the candidate copulas normal, t, Clayton, Frank and Gumbel. The parameters of all copulas but the t copula are estimated using the method of moments. For the t copula, the correlation coefficient is computed as in the normal copula, and the degrees of freedom are estimated by maximum likelihood with the correlation parameter fixed (Demarta and McNeil 2005).

3. Implementation in R

According to the review presented by Santana (2011), the approach followed for the implementation of EDA software currently available through the Internet can be classified into three categories: (1) implementation of a single EDA, (2) independent implementation of multiple EDAs, and (3) common modular implementation of multiple EDAs. In our opinion, the third approach offers greater flexibility for the EDA community. In these modular implementations, the EDA components (e.g., learning and sampling methods) are independently programmed by taking advantage of the common schema shared by most EDAs. This modularity allows the creation and validation of new EDA proposals that combine different components, and promotes code factorization. Additionally, as the EDAs are grouped under the same framework, it facilitates performing empirical studies to compare the behavior of different algorithms. Existing members of this class are **ParadisEO** (Cahon, Melab, and Talbi 2004; DOLPHIN Project Team 2012), **LiO** (Mateo and de la Ossa 2006, 2007), **Mateda-2.0** (Santana *et al.* 2009, 2010) and now **copulaedas**.

The implementation of **copulaedas** follows an object-oriented design inspired by the **Mateda-2.0** toolbox for MATLAB (The MathWorks, Inc. 2014). EDAs implemented in the package are represented by S4 classes (Chambers 2008) with generic functions for their main steps.

Generic function	Description
edaSeed	*Seeding method.* The default method `edaSeedUniform` generates the values of each variable in the initial population from a continuous uniform distribution.
edaOptimize	*Local optimization method.* The use of a local optimization method is disabled by default.
edaSelect	*Selection method.* The default method `edaSelectTruncation` implements truncation selection.
edaLearn	*Learning method.* No default method.
edaSample	*Sampling method.* No default method.
edaReplace	*Replacement method.* The default method `edaReplaceComplete` completely replaces the current population with the new population.
edaReport	*Reporting method.* Reporting progress information is disabled by default.
edaTerminate	*Termination method.* The default method `edaTerminateMaxGen` ends the execution of the algorithm after a maximum number of generations.

Table 1: Description of the generic functions that implement the steps of the general procedure of an EDA outlined in Algorithm 1 and their default methods.

The base class of EDAs in the package is 'EDA', which has two slots: `name` and `parameters`. The `name` slot stores a character string with the name of the EDA and it is used by the `show` method to print the name of the algorithm when it is called with an 'EDA' instance as argument. The `parameters` slot stores all the EDA parameters in a list.

In **copulaedas**, each step of the general procedure of an EDA outlined in Algorithm 1 is represented by a generic function that expects an 'EDA' instance as its first argument. Table 1 shows a description of these functions and their default methods. The help page of these generic functions in the documentation of **copulaedas** contains information about their arguments, return value, and methods already implemented in the package.

The generic functions and their methods that implement the steps of an EDA look at the `parameters` slot of the 'EDA' instance received as first argument for the values of the parameters that affect their behavior. Only named members of the list must be used and reasonable default values should be assumed when a certain component is missing. The help page of each generic function describes the members of the list in the `parameters` slot interpreted by each function and their default values.

The `edaRun` function implements the Algorithm 1 by linking together the generic functions for each step. This function expects four arguments: the 'EDA' instance, the objective function and two vectors specifying the lower and upper bounds of the variables of the objective function. The length of the vectors with the lower and upper bounds should be the same, since it determines the number of variables of the objective function. When `edaRun` is called, it runs the main loop of the EDA until the call to the `edaTerminate` generic function returns `TRUE`. Then, the function returns an instance of the 'EDAResult' class that encapsulates the results of the algorithm. A description of the slots of this class is given in Table 2.

Two subclasses of 'EDA' are already defined in **copulaedas**: 'CEDA', that represents EDAs based on multivariate copulas; and 'VEDA', that represents vine-based EDAs. The implementation of

Slot	Description
eda	'EDA' instance.
f	Objective function.
lower	Lower bounds of the variables of the objective function.
upper	Upper bounds of the variables of the objective function.
numGens	Total number of generations.
fEvals	Total number of evaluations of the objective function.
bestSol	Best solution.
bestEval	Evaluation of the best solution.
cpuTime	Run time of the algorithm in seconds.

Table 2: Description of the slots of the 'EDAResult' class.

UMDA, GCEDA, CVEDA and DVEDA relies on the **copula** (Hofert, Kojadinovic, Maechler, and Yan 2014; Kojadinovic and Yan 2010), **vines** (Gonzalez-Fernandez and Soto 2014b), **mvtnorm** (Genz *et al.* 2014; Genz and Bretz 2009), and **truncnorm** (Trautmann, Steuer, Mersmann, and Bornkamp 2014) R packages. These packages implement the techniques for the estimation and simulation of the probabilistic models used in these EDAs.

4. Using copulaedas

In this section, we illustrate how to use **copulaedas** through several examples. To begin with, we show how to run the EDAs included in the package. Next, we continue with the implementation of a new EDA by using the functionalities provided by the package, and finally we show how to perform an empirical study to compare the behavior of a group of EDAs on benchmark functions and a real-world problem.

The two well-known test problems Sphere and Summation Cancellation are used as the benchmark functions. The functions `fSphere` and `fSummationCancellation` implement these problems in terms of a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ according to

$$f_{\text{Sphere}}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2,$$

$$f_{\text{Summation Cancellation}}(\boldsymbol{x}) = \frac{1}{10^{-5} + \sum_{i=1}^{n} |y_i|}, \quad y_1 = x_1, \, y_i = y_{i-1} + x_i.$$

Sphere is a minimization problem and Summation Cancellation is originally a maximization problem but it is defined in the package as a minimization problem. Sphere has its global optimum at $\boldsymbol{x} = (0, \ldots, 0)$ with evaluation zero and Summation Cancellation at $\boldsymbol{x} = (0, \ldots, 0)$ with evaluation -10^5. For a description of the characteristics of these functions see Bengoetxea, Miquélez, Lozano, and Larrañaga (2002) and Bosman and Thierens (2006).

The results presented in this section were obtained using R version 3.1.0 with **copulaedas** version 1.4.1, **copula** version 0.999-8, **vines** version 1.1.0, **mvtnorm** version 0.9-99991, and **truncnorm** version 1.0-7. Computations were performed on a 64-bit Linux machine with an Intel(R) Core(TM)2 Duo 2.00 GHz processor. In the rest of this section, we assume **copulaedas** has been loaded. This can be attained by running the following command:

```
R> library("copulaedas")
```

4.1. Running the EDAs included in the package

We begin by illustrating how to run the EDAs based on copulas implemented in **copulaedas**. As an example, we execute GCEDA to optimize Sphere in five dimensions. Before creating a new instance of the 'CEDA' class for EDAs based on multivariate copulas, we set up the generic functions for the steps of the EDA according to the expected behavior of GCEDA. The termination criterion is either to find the optimum of the objective function or to reach a maximum number of generations. That is why we set the method for the `edaTerminate` generic function to a combination of the functions `edaTerminateEval` and `edaTerminateMaxGen` through the auxiliary function `edaTerminateCombined`.

```
R> setMethod("edaTerminate", "EDA",
+    edaTerminateCombined(edaTerminateEval, edaTerminateMaxGen))
```

The method for the `edaReport` generic function is set to `edaReportSimple` to make the algorithm print progress information at each generation. This function prints one line at each iteration of the EDA with the minimum, mean and standard deviation of the evaluation of the solutions in the current population.

```
R> setMethod("edaReport", "EDA", edaReportSimple)
```

Note that these methods were set for the base class 'EDA' and therefore they will be inherited by all subclasses. Generally, we find it convenient to define methods of the generic functions that implement the steps of the EDA for the base class, except when different subclasses should use different methods.

The auxiliary function 'CEDA' can be used to create instances of the class with the same name. All the arguments of the function are interpreted as parameters of the EDA to be added as members of the list in the `parameters` slot of the new instance. An instance of 'CEDA' corresponding to GCEDA using empirical marginal distributions smoothed with normal kernels can be created as follows:

```
R> gceda <- CEDA(copula = "normal", margin = "kernel", popSize = 200,
+    fEval = 0, fEvalTol = 1e-6, maxGen = 50)
R> gceda@name <- "Gaussian Copula Estimation of Distribution Algorithm"
```

The methods that implement the generic functions `edaLearn` and `edaSample` for 'CEDA' instances expect three parameters. The `copula` parameter specifies the multivariate copula and it should be set to `"normal"` for GCEDA. The marginal distributions are determined by the value of `margin` and all EDAs implemented in the package use this parameter for the same purpose. As `margin` is set to `"kernel"`, the algorithm will look for three functions named `fkernel`, `pkernel` and `qkernel` already defined in the package to fit the parameters of the margins and to evaluate the distribution and quantile functions, respectively. The `fkernel` function computes the bandwidth parameter of the normal kernel according to the rule-of-thumb of Silverman (1986) and `pkernel` implements the empirical cumulative distribution function. The quantile function is evaluated following the procedure described in Azzalini (1981). The `popSize` parameter determines the population size while the rest of the arguments of `CEDA` are parameters of the functions that implement the termination criterion.

Now, we can run GCEDA by calling `edaRun`. The lower and upper bounds of the variables are set so that the values of the variables in the optimum of the function are located at 25% of the interval. It was shown in Arderí (2007) and Soto *et al.* (2014) that the use of empirical marginal distributions smoothed with normal kernels improves the behavior of GCEDA when the initial population is generated asymmetrically with respect to the optimum of the function.

```
R> set.seed(12345)
R> result <- edaRun(gceda, fSphere, rep(-300, 5), rep(900, 5))

  Generation       Minimum          Mean    Std. Dev.
           1 1.522570e+05 1.083606e+06 5.341601e+05
           2 2.175992e+04 5.612769e+05 3.307403e+05
           3 8.728486e+03 2.492247e+05 1.496334e+05
           4 4.536507e+03 1.025119e+05 5.829982e+04
           5 5.827775e+03 5.126260e+04 2.983622e+04
           6 2.402107e+03 2.527349e+04 1.430142e+04
           7 9.170485e+02 1.312806e+04 6.815822e+03
           8 4.591915e+02 6.726731e+03 4.150888e+03
           9 2.448265e+02 3.308515e+03 1.947486e+03
          10 7.727107e+01 1.488859e+03 8.567864e+02
          11 4.601731e+01 6.030030e+02 3.529036e+02
          12 8.555769e+00 2.381415e+02 1.568382e+02
          13 1.865639e+00 1.000919e+02 6.078611e+01
          14 5.157326e+00 4.404530e+01 2.413589e+01
          15 1.788793e+00 2.195864e+01 1.136284e+01
          16 7.418832e-01 1.113184e+01 6.157461e+00
          17 6.223596e-01 4.880880e+00 2.723950e+00
          18 4.520045e-02 2.327805e+00 1.287697e+00
          19 6.981399e-02 1.123582e+00 6.956201e-01
          20 3.440069e-02 5.118243e-01 2.985175e-01
          21 1.370064e-02 1.960786e-01 1.329600e-01
          22 3.050774e-03 7.634156e-02 4.453917e-02
          23 1.367716e-03 3.400907e-02 2.056747e-02
          24 7.599946e-04 1.461478e-02 8.861180e-03
          25 4.009605e-04 6.488932e-03 4.043431e-03
          26 1.083879e-04 2.625759e-03 1.618058e-03
          27 8.441887e-05 1.079075e-03 5.759307e-04
          28 3.429462e-05 5.077934e-04 3.055568e-04
          29 1.999004e-05 2.232605e-04 1.198675e-04
          30 1.038719e-05 1.104123e-04 5.888948e-05
          31 6.297005e-06 5.516721e-05 2.945027e-05
          32 1.034002e-06 2.537823e-05 1.295004e-05
          33 8.483830e-07 1.332463e-05 7.399488e-06
```

The `result` variable contains an instance of the 'EDAResult' class. The `show` method prints the results of the execution of the algorithm.

```
R> show(result)
```

```
Results for Gaussian Copula Estimation of Distribution Algorithm
Best function evaluation    8.48383e-07
No. of generations          33
No. of function evaluations 6600
CPU time                    7.895 seconds
```

Due to the stochastic nature of EDAs, it is often useful to analyze a sequence of independent runs to ensure reliable results. The `edaIndepRuns` function supports performing this task. To avoid generating lot of unnecessary output, we first disable reporting progress information on each generation by setting `edaReport` to `edaReportDisabled` and then we invoke the `edaIndepRuns` function to perform 30 independent runs of GCEDA.

```
R> setMethod("edaReport", "EDA", edaReportDisabled)
R> set.seed(12345)
R> results <- edaIndepRuns(gceda, fSphere, rep(-300, 5), rep(900, 5), 30)
```

The return value of the `edaIndepRuns` function is an instance of the 'EDAResults' class. This class is simply a wrapper for a list with instances of 'EDAResult' as members that contain the results of an execution of the EDA. A `show` method for 'EDAResults' instances prints a table with all the results.

```
R> show(results)
```

	Generations	Evaluations	Best Evaluation	CPU Time
Run 1	33	6600	8.483830e-07	7.583
Run 2	38	7600	4.789448e-08	8.829
Run 3	36	7200	4.798364e-07	8.333
Run 4	37	7400	9.091651e-07	8.772
Run 5	34	6800	5.554465e-07	7.830
Run 6	35	7000	3.516341e-07	8.071
Run 7	35	7000	9.325531e-07	8.106
Run 8	35	7000	6.712550e-07	8.327
Run 9	36	7200	8.725061e-07	8.283
Run 10	37	7400	2.411458e-07	8.565
Run 11	36	7200	4.291725e-07	8.337
Run 12	35	7000	7.245520e-07	8.313
Run 13	37	7400	2.351322e-07	8.538
Run 14	36	7200	8.651248e-07	8.320
Run 15	34	6800	9.422646e-07	7.821
Run 16	35	7000	9.293726e-07	8.333
Run 17	36	7200	6.007390e-07	8.274
Run 18	38	7600	3.255231e-07	8.763
Run 19	36	7200	6.012969e-07	8.353
Run 20	35	7000	5.627017e-07	8.296
Run 21	36	7200	5.890752e-07	8.259
Run 22	35	7000	9.322505e-07	8.067
Run 23	35	7000	4.822349e-07	8.084

```
Run 24          34          6800    7.895408e-07    7.924
Run 25          36          7200    6.970180e-07    8.519
Run 26          34          6800    3.990247e-07    7.808
Run 27          35          7000    8.876874e-07    8.055
Run 28          33          6600    8.646387e-07    7.622
Run 29          36          7200    9.072113e-07    8.519
Run 30          35          7000    9.414666e-07    8.040
```

Also, the `summary` method can be used to generate a table with a statistical summary of the results of the 30 runs of the algorithm.

```
R> summary(results)
```

```
            Generations Evaluations Best Evaluation  CPU Time
Minimum        33.000000   6600.0000    4.789448e-08 7.5830000
Median         35.000000   7000.0000    6.841365e-07 8.2895000
Maximum        38.000000   7600.0000    9.422646e-07 8.8290000
Mean           35.433333   7086.6667    6.538616e-07 8.2314667
Std. Dev.       1.250747    250.1494    2.557100e-07 0.3181519
```

4.2. Implementation of a new EDA based on copulas

In this section we illustrate how to use **copulaedas** to implement a new EDA based on copulas. As an example, we consider the extension of MIMIC for continuous domains proposed in Salinas-Gutiérrez *et al.* (2009). Similarly to MIMIC, this extension learns a chain dependence structure, but it uses bivariate copulas instead of bivariate normal distributions. The chain dependence structure is similar to a D-vine truncated at the first tree, i.e., a D-vine where independence is assumed for all the trees but the first. Two instances of the extension of MIMIC based on copulas were presented in Salinas-Gutiérrez *et al.* (2009), one uses bivariate normal copulas while the other uses bivariate Frank copulas. In this article, the algorithm will be denoted as Copula MIMIC.

Since the algorithm in question matches the general schema of an EDA presented in Algorithm 1, only the functions corresponding to the learning and simulation steps have to be implemented. The first step in the implementation of a new EDA is to define a new S4 class that inherits from 'EDA' to represent the algorithm. For convenience, we also define an auxiliary function `CopulaMIMIC` that can be used to create new instances of this class.

```
R> setClass("CopulaMIMIC", contains = "EDA",
+    prototype = prototype(name = "Copula MIMIC"))
R> CopulaMIMIC <- function(...)
+    new("CopulaMIMIC", parameters = list(...))
```

Copula MIMIC models the marginal distributions with the beta distribution. A linear transformation is used to map the sample of the variables in the selected population into the $(0, 1)$ interval to match the domain of definition of the beta distribution. Note that, since the copula is scale-invariant, this transformation does not affect the dependence between the variables.

To be consistent with the margins already implemented in **copulaedas**, we define three functions with the common suffix `betamargin` and the prefixes `f`, `p` and `q` to fit the parameters of the margins and for the evaluation of the distribution and quantile functions, respectively. By following this convention, the algorithms already implemented in the package can use beta marginal distributions by setting the `margin` parameter to `"betamargin"`.

```
R> fbetamargin <- function(x, lower, upper) {
+     x <- (x - lower) / (upper - lower)
+     loglik <- function(s) sum(dbeta(x, s[1], s[2], log = TRUE))
+     s <- optim(c(1, 1), loglik, control = list(fnscale = -1))$par
+     list(lower = lower, upper = upper, a = s[1], b = s[2])
+ }
R> pbetamargin <- function(q, lower, upper, a, b) {
+     q <- (q - lower) / (upper - lower)
+     pbeta(q, a, b)
+ }
R> qbetamargin <- function(p, lower, upper, a, b) {
+     q <- qbeta(p, a, b)
+     lower + q * (upper - lower)
+ }
```

The 'CopulaMIMIC' class inherits methods for the generic functions that implement all the steps of the EDA except learning and sampling. To complete the implementation of the algorithm, we must define the estimation and simulation of the probabilistic model as methods for the generic functions `edaLearn` and `edaSample`, respectively.

The method for `edaLearn` starts with the estimation of the parameters of the margins and the transformation of the selected population to uniform variables in $(0, 1)$. Then, the mutual information between all pairs of variables is calculated through the copula entropy (Davy and Doucet 2003). To accomplish this, the parameters of each possible bivariate copula are estimated by the method of maximum likelihood using the value obtained through the method of moments as an initial approximation. To determine the chain dependence structure, a permutation of the variables that maximizes the pairwise mutual information must be selected but, since this is a computationally intensive task, a greedy algorithm is used to compute an approximate solution (De Bonet, Isbell, and Viola 1997; Larrañaga *et al.* 1999). Finally, the method for `edaLearn` returns a list with three components that represents the estimated probabilistic model: the parameters of the marginal distributions, the permutation of the variables, and the copulas in the chain dependence structure.

```
R> edaLearnCopulaMIMIC <- function(eda, gen, previousModel,
+       selectedPop, selectedEval, lower, upper) {
+     margin <- eda@parameters$margin
+     copula <- eda@parameters$copula
+     if (is.null(margin)) margin <- "betamargin"
+     if (is.null(copula)) copula <- "normal"
+     fmargin <- get(paste("f", margin, sep = ""))
+     pmargin <- get(paste("p", margin, sep = ""))
+     copula <- switch(copula, normal = normalCopula(0),
```

```
+       frank = frankCopula(0))
+     n <- ncol(selectedPop)
+     # Estimate the parameters of the marginal distributions.
+     margins <- lapply(seq(length = n),
+       function(i) fmargin(selectedPop[, i], lower[i], upper[i]))
+     uniformPop <- sapply(seq(length = n), function(i) do.call(pmargin,
+       c(list(selectedPop[ , i]), margins[[i]])))
+     # Calculate pairwise mutual information by using copula entropy.
+     C <- matrix(list(NULL), nrow = n, ncol = n)
+     I <- matrix(0, nrow = n, ncol = n)
+     for (i in seq(from = 2, to = n)) {
+       for (j in seq(from = 1, to = i - 1)) {
+         # Estimate the parameters of the copula.
+         data <- cbind(uniformPop[, i], uniformPop[, j])
+         startCopula <- fitCopula(copula, data, method = "itau",
+           estimate.variance = FALSE)@copula
+         C[[i, j]] <- tryCatch(
+           fitCopula(startCopula, data, method = "ml",
+             start = startCopula@parameters,
+             estimate.variance = FALSE)@copula,
+             error = function(error) startCopula)
+         # Calculate mutual information.
+         if (is(C[[i, j]], "normalCopula")) {
+           I[i, j] <- -0.5 * log(1 - C[[i, j]]@parameters^2)
+         } else {
+           u <- rcopula(C[[i, j]], 100)
+           I[i, j] <- sum(log(dcopula(C[[i, j]], u))) / 100
+         }
+         C[[j, i]] <- C[[i, j]]
+         I[j, i] <- I[i, j]
+       }
+     }
+     # Select a permutation of the variables.
+     perm <- as.vector(arrayInd(which.max(I), dim(I)))
+     copulas <- C[perm[1], perm[2]]
+     I[perm, ] <- -Inf
+     for (k in seq(length = n - 2)) {
+       ik <- which.max(I[, perm[1]])
+       perm <- c(ik, perm)
+       copulas <- c(C[perm[1], perm[2]], copulas)
+       I[ik, ] <- -Inf
+     }
+     list(margins = margins, perm = perm, copulas = copulas)
+ }
R> setMethod("edaLearn", "CopulaMIMIC", edaLearnCopulaMIMIC)
```

The `edaSample` method receives the representation of the probabilistic model returned by

`edaLearn` as the `model` argument. The generation of a new solution with n variables starts with the simulation of an n-dimensional vector U having uniform marginal distributions in $(0, 1)$ and the dependence described by the copulas in the chain dependence structure. The first step is to simulate an independent uniform variable U_{π_n} in $(0, 1)$, where π_n denotes the variable in the position n of the permutation π selected by the `edaLearn` method. The rest of the uniform variables are simulated conditionally on the previously simulated variable by using the conditional copula $C(U_{\pi_k}|U_{\pi_{k+1}})$, with $k = n-1, n-2, \ldots, 1$. Finally, the new solution is determined through the evaluation of the beta quantile functions and the application of the inverse of the linear transformation.

```
R> edaSampleCopulaMIMIC <- function(eda, gen, model, lower, upper) {
+     popSize <- eda@parameters$popSize
+     margin <- eda@parameters$margin
+     if (is.null(popSize)) popSize <- 100
+     if (is.null(margin)) margin <- "betamargin"
+     qmargin <- get(paste("q", margin, sep = ""))
+     n <- length(model$margins)
+     perm <- model$perm
+     copulas <- model$copulas
+     # Simulate the chain structure with the copulas.
+     uniformPop <- matrix(0, nrow = popSize, ncol = n)
+     uniformPop[, perm[n]] <- runif(popSize)
+     for (k in seq(from = n - 1, to = 1)) {
+       u <- runif(popSize)
+       v <- uniformPop[, perm[k + 1]]
+       uniformPop[, perm[k]] <- hinverse(copulas[[k]], u, v)
+     }
+     # Evaluate the inverse of the marginal distributions.
+     pop <- sapply(seq(length = n), function(i) do.call(qmargin,
+         c(list(uniformPop[, i]), model$margins[[i]])))
+     pop
+ }
R> setMethod("edaSample", "CopulaMIMIC", edaSampleCopulaMIMIC)
```

The code fragments given above constitute the complete implementation of Copula MIMIC. As it was illustrated with GCEDA in the previous section, the algorithm can be executed by creating an instance of the 'CopulaMIMIC' class and calling the `edaRun` function.

4.3. Performing an empirical study on benchmark problems

We now show how to use **copulaedas** to perform an empirical study of the behavior of a group of EDAs based on copulas on benchmark problems. The algorithms to be compared are UMDA, GCEDA, CVEDA, DVEDA and Copula MIMIC. The first four algorithms are included in **copulaedas** and the fifth algorithm was implemented in Section 4.2. The two functions Sphere and Summation Cancellation described at the beginning of Section 4 are considered as benchmark problems in 10 dimensions.

The aim of this empirical study is to assess the behavior of these algorithms when only linear and independence relationships are considered. Thus, only normal and product copulas are

used in these EDAs. UMDA and GCEDA use multivariate product and normal copulas, respectively. CVEDA and DVEDA are configured to combine bivariate product and normal copulas in the vines. Copula MIMIC learns a chain dependence structure with normal copulas. All algorithms use normal marginal distributions. Note that in this case, GCEDA corresponds to EMNA and Copula MIMIC is similar to MIMIC for continuous domains. In the following code fragment, we create class instances corresponding to these algorithms.

```
R> umda <- CEDA(copula = "indep", margin = "norm")
R> umda@name <- "UMDA"
R> gceda <- CEDA(copula = "normal", margin = "norm")
R> gceda@name <- "GCEDA"
R> cveda <- VEDA(vine = "CVine", indepTestSigLevel = 0.01,
+    copulas = c("normal"), margin = "norm")
R> cveda@name <- "CVEDA"
R> dveda <- VEDA(vine = "DVine", indepTestSigLevel = 0.01,
+    copulas = c("normal"), margin = "norm")
R> dveda@name <- "DVEDA"
R> copulamimic <- CopulaMIMIC(copula = "normal", margin = "norm")
R> copulamimic@name <- "CopulaMIMIC"
```

The initial population is generated using the default `edaSeed` method, therefore, it is sampled uniformly in the real interval of each variable. The lower and upper bounds of the variables are set so that the values of the variables in the optimum of the function are located in the middle of the interval. We use the intervals $[-600, 600]$ in Sphere and $[-0.16, 0.16]$ in Summation Cancellation. All algorithms use the default truncation selection method with a truncation factor of 0.3. Three termination criteria are combined using the `edaTerminateCombined` function: to find the global optimum of the function with a precision greater than 10^{-6}, to reach 300000 function evaluations, or to loose diversity in the population (i.e., the standard deviation of the evaluation of the solutions in the population is less than 10^{-8}). These criteria are implemented in the functions `edaTerminateEval`, `edaTerminateMaxEvals` and `edaTerminateEvalStdDev`, respectively.

The population size of EDAs along with the truncation method determine the sample available for the estimation of the search distribution. An arbitrary selection of the population size could lead to misleading conclusions of the results of the experiments. When the population size is too small, the search distributions might not be accurately estimated. On the other hand, the use of an excessively large population size usually does not result in a better behavior of the algorithms but certainly in a greater number of function evaluations. Therefore, we advocate for the use of the critical population size when comparing the performance of EDAs. The critical population size is the minimum population size required by the algorithm to find the global optimum of the function with a high success rate, e.g., to find the optimum in 30 of 30 sequential independent runs.

An approximate value of the critical population size can be determined empirically using a bisection method (see e.g., Pelikan 2005 for a pseudocode of the algorithm). The bisection method begins with an initial interval where the critical population size should be located and discards one half of the interval at each step. This procedure is implemented in the `edaCriticalPopSize` function. In the experimental study carried out in this section, the initial interval is set to $[50, 2000]$. If the critical population size is not found in this interval,

the results of the algorithm with the population size given by the upper bound are presented.

The complete empirical study consists of performing 30 independent runs of every algorithm on every function using the critical population size. We proceed with the definition of a list containing all algorithm-function pairs.

```
R> edas <- list(umda, gceda, cveda, dveda, copulamimic)
R> fNames <- c("Sphere", "SummationCancellation")
R> experiments <- list()
R> for (eda in edas) {
+    for (fName in fNames) {
+       experiment <- list(eda = eda, fName = fName)
+       experiments <- c(experiments, list(experiment))
+    }
+ }
```

Now we define a function to process the elements of the `experiments` list. This function implements all the experimental setup described before. The output of `edaCriticalPopSize` and `edaIndepRuns` is redirected to a different plain text file for each algorithm-function pair.

```
R> runExperiment <- function(experiment) {
+    eda <- experiment$eda
+    fName <- experiment$fName
+    # Objective function parameters.
+    fInfo <- list(
+      Sphere = list(lower = -600, upper = 600, fEval = 0),
+      SummationCancellation = list(lower = -0.16, upper = 0.16,
+        fEval = -1e5))
+    lower <- rep(fInfo[[fName]]$lower, 10)
+    upper <- rep(fInfo[[fName]]$upper, 10)
+    f <- get(paste("f", fName, sep = ""))
+    # Configure termination criteria and disable reporting.
+    eda@parameters$fEval <- fInfo[[fName]]$fEval
+    eda@parameters$fEvalTol <- 1e-6
+    eda@parameters$fEvalStdDev <- 1e-8
+    eda@parameters$maxEvals <- 300000
+    setMethod("edaTerminate", "EDA",
+      edaTerminateCombined(edaTerminateEval, edaTerminateMaxEvals,
+        edaTerminateEvalStdDev))
+    setMethod("edaReport", "EDA", edaReportDisabled)
+    sink(paste(eda@name, "_", fName, ".txt", sep = ""))
+    # Determine the critical population size.
+    set.seed(12345)
+    results <- edaCriticalPopSize(eda, f, lower, upper,
+      eda@parameters$fEval, eda@parameters$fEvalTol, lowerPop = 50,
+      upperPop = 2000, totalRuns = 30, successRuns = 30,
+      stopPercent = 10, verbose = TRUE)
+    if (is.null(results)) {
```

Algorithm	Success	Pop.	Evaluations	Best Evaluation	CPU Time
Sphere:					
UMDA	30/30	81	3788.1 ± 99.0	6.7e−07 ± 2.0e−07	0.2 ± 0.0
GCEDA	30/30	310	13102.6 ± 180.8	6.8e−07 ± 2.0e−07	0.4 ± 0.0
CVEDA	30/30	104	4804.8 ± 99.9	5.9e−07 ± 1.6e−07	4.5 ± 0.8
DVEDA	30/30	111	5080.1 ± 111.6	6.5e−07 ± 2.3e−07	4.4 ± 0.8
Copula MIMIC	30/30	172	7441.8 ± 127.2	6.8e−07 ± 1.9e−07	65.6 ± 1.2
Summation Cancellation:					
UMDA	0/30	2000	300000.0 ± 0.0	−7.2e+02 ± 4.5e+02	32.5 ± 0.3
GCEDA	30/30	325	38913.3 ± 268.9	−1.0e+05 ± 1.0e−07	4.2 ± 0.1
CVEDA	30/30	294	41228.6 ± 1082.7	−1.0e+05 ± 1.0e−07	279.1 ± 5.1
DVEDA	30/30	965	117022.3 ± 1186.8	−1.0e+05 ± 1.1e−07	1617.5 ± 39.1
Copula MIMIC	0/30	2000	300000.0 ± 0.0	−4.2e+04 ± 3.5e+04	1510.6 ± 54.9

Table 3: Summary of the results obtained in 30 independent runs of UMDA, GCEDA, CVEDA, DVEDA and Copula MIMIC in the 10-dimensional Sphere problem (top) and Summation Cancellation problem (bottom). Pop. denotes Population.

```
+        # Run the experiment with the largest population size, if the
+        # critical population size was not found.
+        eda@parameters$popSize <- 2000
+        set.seed(12345)
+        edaIndepRuns(eda, f, lower, upper, runs = 30, verbose = TRUE)
+      }
+    sink(NULL)
+  }
```

We can run all the experiments by calling `runExperiment` for each element of the list.

```
R> for (experiment in experiments) runExperiment(experiment)
```

Running the complete empirical study sequentially is a computationally demanding operation. If various processing units are available, it can be speeded up significantly by running the experiments in parallel. The **snow** package (Tierney, Rossini, Li, and Sevcikova 2013) offers a great platform to achieve this purpose, since it provides a high-level interface for using a cluster of workstations for parallel computations in R. The functions `clusterApply` or `clusterApplyLB` can be used to call `runExperiment` for each element of the `experiments` list in parallel, with minimal modifications to the code presented here.

A summary of the results of the algorithms with the critical population size is shown in Table 3. Overall, the five algorithms are able to find the global optimum of Sphere in all the 30 independent runs with similar function values but only GCEDA, CVEDA and DVEDA optimize Summation Cancellation. In the rest of this section we provide some comments about results of the algorithms on each function.

UMDA exhibits the best behavior in terms of the number of function evaluations in Sphere. There are no strong dependences between the variables of this function and the results suggest that considering the marginal information is enough to find the global optimum efficiently. The rest of the algorithms being tested require the calculation of a greater number of parameters

to represent the relationships between the variables and hence larger populations are needed to compute them reliably (Soto *et al.* 2014 illustrate this issue in more detail with GCEDA). CVEDA and DVEDA do not assume a normal dependence structure between the variables and for this reason are less affected by this issue. The estimation procedure used by the vine-based algorithms selects the product copula if there is not enough evidence of dependence.

Both UMDA and Copula MIMIC fail to optimize Summation Cancellation. A correct representation of the strong linear interactions between the variables of this function seems to be essential to find the global optimum. UMDA completely ignores this information by assuming independence between the variables and it exhibits the worst behavior. Copula MIMIC reaches better fitness values than UMDA but neither can find the optimum of the function. The probabilistic model estimated by Copula MIMIC cannot represent important dependences necessary for the success of the optimization. The algorithms GCEDA, CVEDA and DVEDA do find the global optimum of Summation Cancellation. The results of GCEDA are slightly better than the ones of CVEDA and these two algorithms achieve much better results than DVEDA in terms of the number of function evaluations. The correlation matrix estimated by GCEDA can properly represent the multivariate linear interactions between the variables. The C-vine structure used in CVEDA, on the other hand, provides a very good fit for the dependence structure between the variables of Summation Cancellation, given that it is possible to find a variable that governs the interactions in the sample (see Gonzalez-Fernandez 2011 for more details).

The results of CVEDA and DVEDA illustrate that the method of moments for the estimation of the copula parameters is a viable alternative to the maximum likelihood method in the context of EDAs, where copulas are fitted at every generation. The empirical investigation confirms the robustness of CVEDA and DVEDA in problems with both weak and strong interactions between the variables. Nonetheless, the flexibility afforded by these algorithms comes with an increased running time when compared to UMDA or GCEDA, since the interactions between the variables have to be discovered during the learning step.

A general result of this empirical study is that copula-based EDAs should use copulas other than the product only when there is evidence of dependence. Otherwise, the EDA will require larger populations and hence a greater number of function evaluations to accurately determine the parameters of the copulas that correspond to independence.

4.4. Solving the molecular docking problem

Finally, we illustrate the use of **copulaedas** for solving a so-called real-world problem. In particular, we use CVEDA and DVEDA to solve an instance of the molecular docking problem, which is an important component of protein-structure-based drug design. From the point of view of the computational procedure, it entails predicting the geometry of a small ligand that binds to the active site of a large macromolecular protein receptor. Protein-ligand docking remains being a highly active area of research, since the algorithms for exploring the conformational space and the scoring functions that have been implemented so far have significant limitations (Warren *et al.* 2006).

In our docking simulations, the protein is treated as a rigid body while the ligand is fully flexible. Thus, a candidate solution represents only the geometry of the ligand and it is encoded as a vector of real values that represent its position, orientation and flexible torsion angles. The first three variables of this vector represent the ligand position in the three-

dimensional space constrained to a box enclosing the receptor binding site. The construction
of this box is based on the minimum and maximum values of the ligand coordinates in its
crystal conformation plus a padding distance of 5Å added to each main direction of the space.
The remainder vector variables are three Euler angles that represent the ligand orientation
as a rigid body and take values in the intervals $[0, 2\pi]$, $[-\pi/2, \pi/2]$ and $[-\pi, \pi]$, respectively;
and one additional variable restricted to $[-\pi, \pi]$ for each flexible torsion angle of the ligand.

The semiempirical free-energy scoring function implemented as part of the suite of automated
docking tools **AutoDock** 4.2 (Morris *et al.* 2009) is used to evaluate each candidate ligand con-
formation. The overall binding energy of a given ligand molecule is expressed as the sum of the
pairwise interactions between the receptor and ligand atoms (intermolecular interaction en-
ergy), and the pairwise interactions between the ligand atoms (ligand intramolecular energy).
The terms of the function consider dispersion/repulsion, hydrogen bonding, electrostatics, and
desolvation effects, all scaled empirically by constants determined through a linear regression
analysis. The aim of an optimization algorithm performing the protein-ligand docking is to
minimize the overall energy value. Further details of the energy terms and how the function
is derived can be found in Huey, Morris, Olson, and Goodsell (2007).

Specifically, we consider as an example here the docking of the 2z5u test system, solved
by X-ray crystallography and available as part of the Protein Data Bank (Berman *et al.*
2000). The protein receptor is lysine-specific histone demethylase 1 and the ligand is a
73-atom molecule (non-polar hydrogens are not counted) with 20 ligand torsions in a box
of 28Å×32Å×24Å. In order to make it easier for the readers of the paper to reproduce this
example, the implementation of the **AutoDock** 4.2 scoring function in C was extracted from
the original program and it is included (with unnecessary dependences removed) in the supple-
mentary material as `docking.c`. During the evaluation of the scoring function, precalculated
grid maps (one for each atom type present in the ligand being docked) are used to make the
docking calculations fast. The result of these precalculations and related metadata for the
2z5u test system are contained in the attached ASCII file `2z5u.dat`.

We make use of the system command `R CMD SHLIB` to build a shared object for dynamic
loading from the file `docking.c`. Next, we integrate the created shared object into R using the
function `dyn.load` and load the precalculated grids using the utility C function `docking_load`
as follows.

```
R> system("R CMD SHLIB docking.c")
R> dyn.load(paste("docking", .Platform$dynlib.ext, sep = ""))
R> .C("docking_load", as.character("2z5u.dat"))
```

The docking of the 2z5u test system results in a minimization problem with a total of 26
variables. Two vectors with the lower and upper bounds of these variables are defined using
utility functions provided in `docking.c` to compute the bounds of the variables that determine
the position of the ligand and the total number of torsions. For convenience, we also define
an R wrapper function `fDocking` for the C scoring function `docking_score` provided in the
compiled code that was loaded into R.

```
R> lower <- c(.C("docking_xlo", out = as.double(0))$out,
+    .C("docking_ylo", out = as.double(0))$out,
+    .C("docking_zlo", out = as.double(0))$out, 0, -pi/2, -pi,
+    rep(-pi, .C("docking_ntor", out = as.integer(0))$out))
```

```
R> upper <- c(.C("docking_xhi", out = as.double(0))$out,
+    .C("docking_yhi", out = as.double(0))$out,
+    .C("docking_zhi", out = as.double(0))$out, 2 * pi, pi/2, pi,
+    rep(pi, .C("docking_ntor", out = as.integer(0))$out))
R> fDocking <- function(sol)
+    .C("docking_score", sol = as.double(sol), out = as.double(0))$out
```

CVEDA and DVEDA are used to solve the minimization problem, since they are the most robust algorithms among the EDAs based on copulas implemented in **copulaedas**. The parameters of these algorithms are set to the values reported by Soto *et al.* (2012) in the solution of the 2z5u test system. The population size of CVEDA and DVEDA is set to 1400 and 1200, respectively. Both algorithms use the implementation of the truncated normal marginal distributions (Johnson, Kotz, and Balakrishnan 1994) provided by Trautmann *et al.* (2014) to satisfy the box constraints of the variables. The termination criterion of both CVEDA and DVEDA is to reach a maximum of 100 generations, since an optimum value of the scoring function is not known. Instances of the 'VEDA' class that follow the description given above are created with the following code.

```
R> setMethod("edaTerminate", "EDA", edaTerminateMaxGen)
R> cveda <- VEDA(vine = "CVine", indepTestSigLevel = 0.01,
+    copulas = "normal", margin = "truncnorm", popSize = 1400, maxGen = 100)
R> dveda <- VEDA(vine = "DVine", indepTestSigLevel = 0.01,
+    copulas = "normal", margin = "truncnorm", popSize = 1200, maxGen = 100)
```

Now we proceed to perform 30 independent runs of each algorithm using the `edaIndepRuns` function. The arguments of this function are the instances of the 'VEDA' class corresponding to CVEDA and DVEDA, the scoring function `fDocking`, and the vectors `lower` and `upper` that determine the bounds of the variables.

```
R> set.seed(12345)
R> cvedaResults <- edaIndepRuns(cveda, fDocking, lower, upper, runs = 30)
R> summary(cvedaResults)
```

	Generations	Evaluations	Best Evaluation	CPU Time
Minimum	100	140000	-30.560360	1421.9570
Median	100	140000	-29.387815	2409.0680
Maximum	100	140000	-23.076769	3301.5960
Mean	100	140000	-29.139028	2400.4166
Std. Dev.	0	0	1.486381	462.9442

```
R> set.seed(12345)
R> dvedaResults <- edaIndepRuns(dveda, fDocking, lower, upper, runs = 30)
R> summary(dvedaResults)
```

	Generations	Evaluations	Best Evaluation	CPU Time
Minimum	100	120000	-30.93501	1928.2030
Median	100	120000	-30.70019	3075.7960

Algorithm	Pop.	Evaluations	Lowest Energy	RMSD	CPU Time
CVEDA	1400	140000.0 ± 0.0	-29.13 ± 1.48	0.58 ± 0.13	2400.4 ± 462.9
DVEDA	1200	120000.0 ± 0.0	-30.01 ± 1.68	0.56 ± 0.14	3053.9 ± 695.0

Table 4: Summary of the results obtained in 30 independent runs of CVEDA and DVEDA for the docking of the 2z5u test system. Pop. denotes Population.
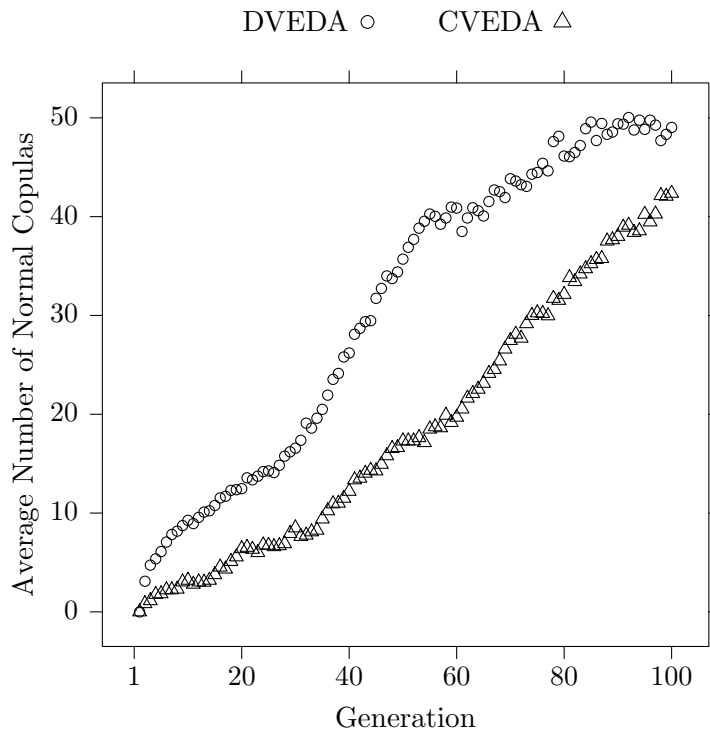
DVEDA ○   CVEDA △

Figure 2: Average number of normal copulas selected at each generation of CVEDA and DVEDA in 30 independent runs for the docking of the 2z5u test system. Since this is an optimization problem with 26 variables, the C-vines and D-vines have a total of 325 copulas.

```
Maximum          100      120000      -24.01502 4872.1820
Mean             100      120000      -30.01427 3053.9579
Std. Dev.          0           0        1.68583  695.0071
```

The results obtained in the docking of the 2z5u test system are summarized in Table 4. In addition to the information provided by `edaIndepRuns`, we present a column for the RMSD (root mean square deviation) between the coordinates of the atoms in the experimental crystal structure and the predicted ligand coordinates of the best solution found at each run. These values can be computed for a particular solution using the `docking_rmsd` function included in `docking.c`. The RMSD values serve as a measure of the quality of the predicted ligand conformations when an experimentally determined solution is known.

Generally, a structure with RMSD below 2Å can be considered as successfully docked. Therefore, both CVEDA and DVEDA achieve good results when solving the 2z5u test system. DVEDA exhibits slightly lower energy and RMSD values, and it requires a smaller popula-

tion size than CVEDA. On the other hand, DVEDA uses more CPU time than CVEDA, a fact that might be related with more dependences being encoded in the D-vines estimated by DVEDA than in the C-vines used by CVEDA. This situation is illustrated in Figure 2, which presents the average number of normal copulas selected by CVEDA and DVEDA at each generation in the 30 independent runs. In both algorithms, the number of normal copulas increases during the evolution, but DVEDA consistently selects more normal copulas than CVEDA. Although the estimation procedure of the C-vines in CVEDA intends to represent the strongest dependences in the first tree, the constraint that only one variable can be connected to all the others prevent strong correlations to be explicitly encoded in the C-vines.

It is also worth noting that because of the use of the truncation procedure based on AIC in both CVEDA and DVEDA, the number of statistical tests required to estimate the vines was dramatically reduced. The median number of vine trees fitted in the 30 independent runs was 4 in CVEDA and 5 in DVEDA out of a total of 25 trees. The interested reader is referred to Soto *et al.* (2012) for a deeper study of the use of EDAs based on copulas for solving the molecular docking problem.

5. Concluding remarks

We have developed **copulaedas** aiming at providing in a single package an open-source implementation of EDAs based on copulas and utility functions to study these algorithms. In this paper, we illustrate how to run the copula-based EDAs implemented in the package, how to implement new algorithms, and how to perform an empirical study to compare copula-based EDAs on benchmark functions and practical problems. We hope that these functionalities help the research community to improve EDAs based on copulas by getting a better insight of their strengths and weaknesses, and also help practitioners to find new applications of these algorithms to real-world problems.

References

Aas K, Czado C, Frigessi A, Bakken H (2009). "Pair-Copula Constructions of Multiple Dependence." *Insurance: Mathematics and Economics*, **44**(2), 182–198.

Abdous B, Genest C, Rémillard B (2005). "Dependence Properties of Meta-Elliptical Distributions." In P Duchesne, B Rémillard (eds.), *Statistical Modelling and Analysis for Complex Data Problems*, pp. 1–15. Kluwer Academic Publisher.

Akaike H (1974). "A New Look at Statistical Model Identification." *IEEE Transactions on Automatic Control*, **19**(6), 716–723.

Arderí RJ (2007). *Algoritmo con estimación de distribuciones con cópula Gaussiana*. Bachelor's thesis, University of Havana, Cuba.

Azzalini A (1981). "A Note on the Estimation of a Distribution Function and Quantiles by a Kernel Method." *Biometrika*, **68**(1), 326–328.

Baluja S (1994). "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning." *Technical Report*

CMU-CS-94-163, Carnegie Mellon University, USA. URL http://www.esprockets.com/papers/CMU-CS-94-163.ps.gz.

Barba-Moreno S (2007). *Una propuesta para algoritmos de estimación de distribución no paramétricos.* Master's thesis, Center for Research in Mathematics, Mexico.

Bedford T, Cooke RM (2001). "Probability Density Decomposition for Conditionally Dependent Random Variables Modeled by Vines." *The Annals of Mathematics and Artificial Intelligence*, **32**(1), 245–268.

Bengoetxea E, Miquélez T, Lozano JA, Larrañaga P (2002). "Experimental Results in Function Optimization with EDAs in Continuous Domain." In P Larrañaga, JA Lozano (eds.), *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pp. 181–194. Kluwer Academic Publisher.

Berg D, Aas K (2007). "Models for Construction of Multivariate Dependence." *Note SAMBA/23/07*, Norwegian Computing Center, NR, Norway. URL http://www.nr.no/files/samba/bff/SAMBA2307.pdf.

Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE (2000). "The Protein Data Bank." *Nucleic Acids Research*, **28**(1), 235–242.

Beyer HG, Schwefel HP (2002). "Evolution Strategies: A Comprehensive Introduction." *Natural Computing*, **1**(1), 3–52.

Bosman PAN, Thierens D (2006). "Numerical Optimization with Real-Valued Estimation of Distribution Algorithms." In M Pelikan, K Sastry, E Cantú-Paz (eds.), *Scalable Optimization via Probabilistic Modeling. From Algorithms to Applications*, pp. 91–120. Springer-Verlag.

Brechmann EC (2010). *Truncated and Simplified Regular Vines and Their Applications.* Diploma thesis, University of Technology, Munich, Germany. URL http://mediatum.ub.tum.de/doc/1079285/1079285.pdf.

Cahon S, Melab N, Talbi E (2004). "**ParadisEO**: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics." *Journal of Heuristics*, **10**(3), 357–380.

Chambers J (2008). *Software for Data Analysis: Programming with R.* Springer-Verlag.

Cuesta-Infante A, Santana R, Hidalgo JI, Bielza C, naga PL (2010). "Bivariate Empirical and n-Variate Archimedean Copulas in Estimation of Distribution Algorithms." In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010)*, pp. 1355–1362.

Davy M, Doucet A (2003). "Copulas: A New Insight into Positive Time-Frequency Distributions." *IEEE Signal Processing Letters*, **10**(7), 215–218.

De Bonet JS, Isbell CL, Viola P (1997). "MIMIC: Finding Optima by Estimating Probability Densities." In M Mozer, M Jordan, T Petsche (eds.), *Advances in Neural Information Processing Systems*, volume 9, pp. 424–430. MIT Press.

Demarta S, McNeil AJ (2005). "The t Copula and Related Copulas." *International Statistical Review*, **73**(1), 111–129.

Devroye L (1986). *Non-Uniform Random Variate Generation.* Springer-Verlag.

DOLPHIN Project Team (2012). **ParadisEO** *2.0.1.* Inria Lille – Nord Europe, Villeneuve d'Ascq, France. URL http://paradiseo.gforge.inria.fr/.

Fang HB, Fang KT, Kotz S (2002). "The Meta-Elliptical Distributions with Given Marginals." *Journal of Multivariate Analysis*, **82**(1), 1–16.

Gao Y (2009). "Multivariate Estimation of Distribution Algorithm with Laplace Transform Archimedean Copula." In W Hu, X Li (eds.), *Proceedings of the International Conference on Information Engineering and Computer Science (ICIECS 2009)*, pp. 1–5.

Genest C, Favre AC (2007). "Everything You Always Wanted to Know about Copula Modeling But Were Afraid to Ask." *Journal of Hydrologic Engineering*, **12**(4), 347–368.

Genest C, Ghoudi K, Rivest LP (1995). "A Semiparametric Estimation Procedure of Dependence Parameters in Multivariate Families of Distributions." *Biometrika*, **82**(3), 543–552.

Genest C, Quessy JF, Rémillard B (2007). "Asymptotic Local Efficiency of Cramér-von Mises Tests for Multivariate Independence." *The Annals of Statistics*, **35**(1), 166–191.

Genest C, Rémillard B (2004). "Tests of Independence or Randomness Based on the Empirical Copula Process." *Test*, **13**(2), 335–369.

Genest C, Rémillard B (2008). "Validity of the Parametric Bootstrap for Goodness-of-Fit Testing in Semiparametric Models." *Annales de l'Institut Henri Poincaré: Probabilités et Statistiques*, **44**(6), 1096–1127.

Genz A, Bretz F (2009). *Computation of Multivariate Normal and t Probabilities.* Springer-Verlag.

Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2014). **mvtnorm***: Multivariate Normal and t Distributions.* R package version 0.9-99991, URL http://CRAN.R-project.org/package=mvtnorm.

Gonzalez-Fernandez Y (2011). *Algoritmos con estimación de distribuciones basados en cópulas y vines.* Bachelor's thesis, University of Havana, Cuba.

Gonzalez-Fernandez Y, Soto M (2014a). **copulaedas***: Estimation of Distribution Algorithms Based on Copulas.* R package version 1.4.1, URL http://CRAN.R-project.org/package=copulaedas.

Gonzalez-Fernandez Y, Soto M (2014b). **vines***: Multivariate Dependence Modeling with Vines.* R package version 1.1.0, URL http://CRAN.R-project.org/package=vines.

Haff IH, Aas K, Frigessi A (2010). "On the Simplified Pair-Copula Construction – Simply Useful or Too Simplistic?" *Journal of Multivariate Analysis*, **101**(5), 1296–1310.

Hauschild M, Pelikan M (2011). "An Introduction and Survey of Estimation of Distribution Algorithms." *Swarm and Evolutionary Computation*, **1**(3), 111–128.

Hering C, Hofert M, Mai JF, Scherer M (2010). "Constructing Hierarchical Archimedean Copulas with Lévy Subordinators." *Journal of Multivariate Analysis*, **101**(6), 1428–1433.

Hofert M, Kojadinovic I, Maechler M, Yan J (2014). **copula**: *Multivariate Dependence with Copulas*. R package version 0.999-8, URL http://CRAN.R-project.org/package=copula.

Hooke R, Jeeves TA (1961). "Direct Search Solution of Numerical and Statistical Problems." *Journal of the Association for Computing Machinery*, **8**(2), 212–229.

Huey R, Morris GM, Olson AJ, Goodsell DS (2007). "A Semiempirical Free Energy Force Field with Charge-Based Desolvation." *Journal of Computational Chemistry*, **28**(6), 1145–1152.

Hult H, Lindskog F (2002). "Multivariate Extremes, Aggregation and Dependence in Elliptical Distributions." *Advances in Applied Probability*, **34**(3), 587–608.

Joe H (1996). "Families of m-Variate Distributions with Given Margins and $m(m-1)/2$ Bivariate Dependence Parameters." In L Rüschendorf, B Schweizer, MD Taylor (eds.), *Distributions with Fixed Marginals and Related Topics*, pp. 120–141.

Joe H (1997). *Multivariate Models and Dependence Concepts*. Chapman & Hall.

Joe H (2005). "Asymptotic Efficiency of the Two-Stage Estimation Method for the Copula-Based Models." *Journal of Multivariate Analysis*, **94**(2), 401–419.

Joe H, Li H, Nikoloulopoulos AK (2010). "Tail Dependence Functions and Vine Copulas." *Journal of Multivariate Analysis*, **101**(1), 252–270.

Joe H, Xu JJ (1996). "The Estimation Method of Inference Functions for Margins for Multivariate Models." *Technical Report 166*, University of British Columbia. URL http://www.stat.ubc.ca/~harry/ifm.pdf.

Johnson NL, Kotz S, Balakrishnan N (1994). *Continuous Univariate Distributions*, volume 1. 2nd edition. John Wiley & Sons.

Kennedy J, Eberhart RC (1995). "Particle Swarm Optimization." In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pp. 1942–1948.

Kern S, Müller SD, Hansen N, Büche D, Ocenasek J, Koumoutsakos P (2003). "Learning Probability Distributions in Continuous Evolutionary Algorithms – A Comparative Review." *Natural Computing*, **3**(1), 77–112.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). "Optimization by Simulated Annealing." *Science*, **220**(4598), 671–680.

Kojadinovic I, Yan J (2010). "Modeling Multivariate Distributions with Continuous Margins Using the **copula** R Package." *Journal of Statistical Software*, **34**(9), 1–20. URL http://www.jstatsoft.org/v34/i09/.

Larrañaga P, Etxeberria R, Lozano JA, Peña JM (1999). "Optimization by Learning and Simulation of Bayesian and Gaussian Networks." *Technical Report EHU-KZAA-IK-4/99*, University of the Basque Country. URL http://www.sc.ehu.es/ccwbayes/publications/postscript/kzaa-ik-04-99.ps.

Larrañaga P, Etxeberria R, Lozano JA, Peña JM (2000). "Optimization in Continuous Domains by Learning and Simulation of Gaussian Networks." In *Proceedings of the Workshop in Optimization by Building and Using Probabilistic Models in the Genetic and Evolutionary Computation Conference (GECCO 2000)*, pp. 201–204.

Larrañaga P, Lozano JA (eds.) (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publisher.

Larrañaga P, Lozano JA, Bengoetxea E (2001). "Estimation of Distribution Algorithms Based on Multivariate Normal and Gaussian Networks." *Technical Report EHU-KZAA-IK-1/01*, University of the Basque Country.

Mateo JL, de la Ossa L (2006). ***LiO***. Intelligent Systems Group and Data Mining (SIMD), Universidad de Castilla-La Mancha, Albacete, Spain. URL http://www.dsi.uclm.es/simd/SOFTWARE/LIO/.

Mateo JL, de la Ossa L (2007). "**LiO**: An Easy and Flexible Library of Metaheuristics." *Technical Report DIAB-06-04-1*, Department of Computing Systems, Escuela Politecnica Superior de Castilla La Mancha, Albacete, Spain. URL http://www.info-ab.uclm.es/descargas/thecnicalreports/DIAB-06-04-1/LiO.pdf.

McNeil AJ (2008). "Sampling Nested Archimedean Copulas." *Journal of Statistical Computation and Simulation*, **78**(6), 567–581.

McNeil AJ, Nešlehová (2009). "Multivariate Archimedean Copulas, d-Monotone Functions and l_1-Norm Symmetric Distributions." *The Annals of Statistics*, **37**(5b), 3059–3097.

Morris GM, Huey R, Lindstrom W, Sanner MF, Belew RK, Goodsell DS, Olson, J A (2009). "**AutoDock4** and **AutoDockTools4**: Automated Docking with Selective Receptor Flexiblity." *Journal of Computational Chemistry*, **30**(16), 2785–2791.

Mühlenbein H, Paaß G (1996). "From Recombination of Genes to the Estimation of Distributions I. Binary Parameters." In *Parallel Problem Solving from Nature — PPSN IV*, pp. 178–187. Springer-Verlag.

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *Computer Journal*, **7**(4), 308–313.

Nelsen RB (2006). *An Introduction to Copulas*. 2nd edition. Springer-Verlag.

Nocedal J, Wright SJ (1999). *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag.

Pelikan M (2005). *Hierarchical Bayesian Optimization Algorithm. Toward a New Generation of Evolutionary Algorithms*. Springer-Verlag.

Pelikan M, Goldberg DE, Lobo F (2002). "A Survey of Optimization by Building and Using Probabilistic Models." *Computational Optimization and Applications*, **21**(1), 5–20.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rosenkrantz DJ, Stearns RE, Lewis, II PM (1977). "An Analysis of Several Heuristics for the Traveling Salesman Problem." *SIAM Journal on Computing*, **6**(3), 563–581.

Rousseeuw P, Molenberghs G (1993). "Transformation of Nonpositive Semidefinite Correlation Matrices." *Communications in Statistics – Theory and Methods*, **22**(4), 965–984.

Salinas-Gutiérrez R, Hernández-Aguirre A, Villa-Diharce E (2009). "Using Copulas in Estimation of Distribution Algorithms." In *Proceedings of the Eight Mexican International Conference on Artificial Intelligence (MICAI 2009)*, pp. 658–668.

Salinas-Gutiérrez R, Hernández-Aguirre A, Villa-Diharce E (2010). "D-Vine EDA: A New Estimation of Distribution Algorithm Based on Regular Vines." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, pp. 359–365.

Santana R (2011). "Estimation of Distribution Algorithms: From Available Implementations to Potential Developments." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pp. 679–686.

Santana R, Bielza C, Larrañaga P, Lozano JA, Echegoyen C, Mendiburu A, Armañanzas R, Shakya S (2010). "**Mateda-2.0**: A MATLAB Package For the Implementation and Analysis of Estimation of Distribution Algorithms." *Journal of Statistical Software*, **35**(7), 1–30. URL http://www.jstatsoft.org/v35/i07/.

Santana R, Echegoyen C, Mendiburu A, Bielza C, Lozano JA, Larrañaga P, Armañanzas R, Shakya S (2009). *Mateda-2.0*. University of the Basque Country. URL http://www.sc.ehu.es/ccwbayes/members/rsantana/software/matlab/MATEDA.html.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464.

Silverman BW (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

Sklar A (1959). "Fonctions de répartition à n dimensions et leurs marges." *Publications de l'Institut de Statistique de l'Université de Paris*, **8**, 229–231.

Soto M, Gonzalez-Fernandez Y (2010). "Vine Estimation of Distribution Algorithms." *Technical Report ICIMAF 2010-561*, Institute of Cybernetics, Mathematics and Physics, Cuba.

Soto M, Gonzalez-Fernandez Y, Ochoa A (2014). "Modeling with Copulas and Vines in Estimation of Distribution Algorithms." *Investigación Operacional*. Forthcoming, preprint available at http://arxiv.org/abs/1210.5500.

Soto M, Ochoa A, Arderí RJ (2007). "Gaussian Copula Estimation of Distribution Algorithm." *Technical Report ICIMAF 2007-406*, Institute of Cybernetics, Mathematics and Physics, Cuba.

Soto M, Ochoa A, Gonzalez-Fernandez Y, Milanés Y, Álvarez A, Carrera D, Moreno E (2012). "Vine Estimation of Distribution Algorithms with Application to Molecular Docking." In S Shakya, R Santana (eds.), *Markov Networks in Evolutionary Computation*, volume 14 of *Adaptation, Learning, and Optimization*, pp. 209–225. Springer-Verlag.

Storn R, Price K (1997). "Differential Evolution — A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces." *Journal of Global Optimization*, **11**(4), 341–359.

The MathWorks, Inc (2014). *MATLAB – The Language of Technical Computing, Version R2014a*. The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

Tierney L, Rossini AJ, Li N, Sevcikova H (2013). **snow**: *Simple Network of Workstations*. R package version 0.3-13, URL http://CRAN.R-project.org/package=snow.

Trautmann H, Steuer D, Mersmann O, Bornkamp B (2014). **truncnorm**: *Truncated Normal Distribution*. R package version 1.0-7, URL http://CRAN.R-project.org/package=truncnorm.

Wang L, Zeng J, Hong Y (2009a). "Estimation of Distribution Algorithm Based on Archimedean Copulas." In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC 2009)*, pp. 993–996.

Wang L, Zeng J, Hong Y (2009b). "Estimation of Distribution Algorithm Based on Copula Theory." In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 1057–1063.

Warren GL, Andrews CW, Capelli AM, Clarke B, LaLonde J, Lambert MH, Lindvall M, Nevins N, Semus SF, Senger S, Tedesco G, Wall ID, Woolven JM, Peishoff CE, Head MS (2006). "A Critical Assessment of Docking Programs and Scoring Functions." *Journal of Medicinal Chemistry*, **49**(20), 5912–5931.

Ye B, Gao H, Wang X, Zeng J (2010). "Estimation of Distribution Algorithm Based on Nested Archimedean Copulas Constructed with Lévy Subordinators." In *Proceedings of the Eleventh International Conference on Computer-Aided Industrial Design & Conceptual Design (CAIDCD 2010)*, pp. 1586–1590.

Affiliation:

Yasser Gonzalez-Fernandez
Department of Interdisciplinary Mathematics
Institute of Cybernetics, Mathematics and Physics
Calle 15 No. 551 e/ C y D, Vedado, La Habana, Cuba
Current affiliation:
School of Information Technology
York University
4700 Keele St, Toronto, Ontario, Canada
E-mail: ygf@yorku.ca

Marta Soto
Department of Interdisciplinary Mathematics
Institute of Cybernetics, Mathematics and Physics
Calle 15 No. 551 e/ C y D, Vedado, La Habana, Cuba
E-mail: mrosa@icimaf.cu