perm_mateda: A matlab toolbox of estimation of distribution algorithms for permutation-based combinatorial optimization problems

EKHINE IRUROZKI, Intelligent Systems Group, School of Computer Science, University of the Basque Country UPV/EHU
JOSU CEBERIO, Intelligent Systems Group, School of Engineering, University of the Basque Country UPV/EHU
JOSEAN SANTAMARIA, Intelligent Systems Group, School of Computer Science, University of the Basque Country UPV/EHU
ROBERTO SANTANA, Intelligent Systems Group, School of Computer Science, University of the Basque Country UPV/EHU
ALEXANDER MENDIBURU, Intelligent Systems Group, School of Computer Science, University of the Basque Country UPV/EHU

Permutation problems are combinatorial optimization problems whose solutions are naturally codified as permutations. Due to their complexity, motivated principally by the factorial cardinality of the search space of solutions, they have been a recurrent topic for the artificial intelligence and operations research community. Recently, among the vast number of metaheuristic algorithms, new advances on estimation of distribution algorithms (EDAs) have shown outstanding performance when solving some permutation problems. These novel EDAs implement distance-based exponential probability models such as the Mallows and Generalized Mallows models. In this paper, we present a Matlab package, perm_mateda, for estimation of distribution algorithms on permutation problems, which has been implemented as an extension to the Mateda-2.0 toolbox of EDAs. Particularly, we provide implementations of the Mallows and Generalized Mallows EDAs under the Kendall's-τ, Cayley, and Ulam distances. In addition, four classical permutation problems have been also implemented: Traveling Salesman Problem, Permutation Flowshop Scheduling Problem, Linear Ordering Problem, and Quadratic Assignment Problem.

CCS Concepts: •**Mathematics of computing** → **Permutations and combinations; Combinatorial optimization; Probability and statistics;** •**Applied computing** → **Operations research;**

Additional Key Words and Phrases: estimation of distribution algorithms, Mallows and Generalized Mallows models, optimization, permutation-based problems, Matlab

ACM Reference Format:
Ekhine Irurozki, Josu Ceberio, Josean Santamaria, Roberto Santana, Alexander Mendiburu, 2016. perm_mateda: A matlab toolbox of estimation of distribution algorithms for permutation-based combinatorial optimization problems. *ACM Trans. Math. Soft.* 0, 0, Article 0 (0), 16 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

This work has been partially supported by the Research Groups 2013-2018 (IT-609-13) programs (Basque Government), TIN2013-41272P (Ministry of Science and Technology).
Author's addresses: E. Irurozki, J. Santamaria and R. Santana, Computer Science and Artificial Intelligence Department, Faculty of Computer Science, UPV/EHU, 20018 Paseo M. Lardizabal 1, Donostia - San Sebastian, Gipuzkoa, Spain; J. Ceberio, Computer Languages and Systems Department, Engineering School of Bilbao, UPV/EHU, 40013 Paseo Rafael Moreno "Pitxitxi" 3, Bilbao, Bizkaia; A. Mendiburu, Computer Architecture and Technology, Faculty of Computer Science, UPV/EHU, 20018 Paseo M. Lardizabal 1, Donostia / San Sebastian, Gipuzkoa, Spain.

DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

In combinatorics, many optimisation problems are defined as "the way of ordering n number of items" such that a specific function is maximized (or minimized). Referred to as *permutation-based problems*, or simply *permutation problems*, these combinatorial problems are characterized by the fact that their solutions are naturally codified as permutations. Motivated principally by their versatility - ordered sets of items, collection of disjoint cycles, transpositions, matrices or graphs- permutations appear in a vast range of domains, such as graph theory, mathematical psychology or bioinformatics, and, particularly, in logistic problems such as routing [Toth and Vigo 2001], scheduling [Gupta and Stafford 2006] or assignment [Burkard et al. 1998]. The DNA fragment assembly [Parsons et al. 1995], vehicle routing [Toth and Vigo 2001] or aircraft landing scheduling [Beasley et al. 2000] problems are examples of the variety of permutation problems.

If no constraint is assumed, the search space of solutions is defined as the set of all the permutations of n items ($n!$ solutions in total). Due to the factorial cardinality of the search space, permutation problems are known as very hard problems when n goes above a relatively small number. Indeed, the work of Garey and Johnson [Garey and Johnson 1979] on computational complexity demonstrated that many of these problems are *NP-hard*.

In view of their complexity, computing *optimal* solutions is intractable in general. For this reason, we are usually satisfied with *good* solutions. In this sense, the artificial intelligence community has proposed a large number of metaheuristic algorithms that provide *acceptable* solutions in reasonable computation times. Generally, these algorithms neither guarantee the optimality of solutions nor define how close the obtained solutions are from the optimal ones. Among the vast amount of metaheuristic algorithms, tabu search, scatter search, local search, variable neighbourhood search, ant colony optimisation, simulated annealing and genetic algorithms are some of the metaheuristics that have been applied on permutation problems.

Recently, Estimation of Distribution Algorithms (EDAs) [Larrañaga and Lozano 2002; Lozano et al. 2006; Pelikan et al. 2002, 2006] have also been applied successfully to permutation problems. EDAs are population based optimization algorithms that, at each generation, estimate a probability distribution from the selected solutions in order to represent the (in)dependencies between the variables. Then, the new solutions are obtained by sampling the probability distribution estimated in the previous step. This process (hopefully) leads the algorithm towards the optimal solution. Different EDAs have been proposed for discrete, continuous and mixed problems. Many works in the literature confirm the good performance of EDAs in artificial and real-world problems: Protein Folding [Armañanzas et al. 2008], Capacitated Vehicle Routing Problems [Tsutsui and Wilson 2004], Calibration of Chemical Applications [Mendiburu et al. 2006], Finding the Optimal Path in 3D Spaces [Yuan et al. 2007], Software Testing [Sagarna and Lozano 2006], Chemotherapy Treatment Optimization for Cancer [Brownlee et al. 2008], Nuclear Reactor Fuel Management Parameter Optimization [Jiang et al. 2006], Dynamic Pricing [McCall et al. 2012] or Molecular Docking [Soto et al. 2012].

A recent review by Ceberio et al. [2012] studied the performance of several classical EDAs when applied to permutation-based problems, confirming that, in general, they are not very competitive. The permutation codification of solutions represents a real challenge for EDAs, since classical probability distributions on the discrete or continuous domains can not be efficiently adapted to deal with permutation solutions. Notions, such as variable independence, are not naturally translated into the domain of permutations since, in contrast to integer problems, two given positions in a permutation can

not ever have the same value. This simple constraint, known as the *mutual exclusivity constraint* [Huang et al. 2009], requires in general a more complex mathematical machinery in order to deal efficiently with the permutation nature of the solutions.

In order to overcome this drawback, recently, several distance-based probability models, such as the Mallows and the Generalized Mallows models have been included in the context of EDAs [Ceberio et al. 2011b,a, 2014, 2013, 2015]. This step opens new research lines, both methodological and applied, in a topic (solving permutation problems) of growing interest in the literature. In addition to theoretical works, in Ceberio et al. [2014] authors present an EDA based on the Generalized Mallows model able to outperform state-of-the-art algorithms, obtaining new best-known results for the Permutation Flowshop Scheduling Problem. Taking into account its potential, and with the aim of providing a powerful tool for the community, we present a complete framework of EDAs for permutation problems in Matlab, which is implemented as an extension to the Mateda-2.0 toolbox of EDAs developed by Santana et al. [2010]. Particularly, we focus on two distance-based exponential probability models, called the Mallows model [Mallows 1957] and the Generalized Mallows model [Fligner and Verducci 1986]. In order to enhance the applicability and robustness of this framework, based on these two models, we present six different EDAs (two models combined with three distance-metrics): the Mallows EDA (MEDA) and the Generalized Mallows EDA (GMEDA), under the Kendall's-τ, Cayley and Ulam distance-metrics. In addition, four classical permutation problems have been also implemented for experimental purposes: Traveling Salesman Problem (TSP) [Goldberg and Jr. 1985], Permutation Flowshop Scheduling Problem (PFSP) [Gupta and Stafford 2006], Linear Ordering Problem (LOP) [Ceberio et al. 2014], and Quadratic Assignment Problem (QAP) [Koopmans and Beckmann 1955].

The remainder of the paper is structured as follows: in the next section a brief introduction on EDAs is given. In Section 3, the Mallows and Generalized Mallows models under the Kendall's-τ, Cayley and Ulam distances are introduced, together with an introduction to their implementation in Matlab. Section 4 describes the four permutation problems implemented in the package. Afterwards, Section 5 presents two complete example-codes for solving PFSP and LOP by means of Mallows with Kendall's-τ and Generalized Mallows models with Cayley. Finally, conclusions and ideas for future work are presented in Section 6.

2. A BRIEF REVIEW OF EDAS ON MATEDA-2.0

Algorithm 1 provides the general overview of EDAs. The methods the user can implement or adapt are highlighted in italics.

EDAs begin with the generation of an initial set of solutions (usually called a population). Although the first population is usually randomly generated, it can be done using a particular heuristic or *seeding method* in some situations, e.g., when previous information about the approximate location of the optimal solutions is available.

Selection methods serve to identify the subset of solutions that will be used to learn the probabilistic model. This subset usually gathers the solutions with the best value, according to the evaluation function defined for the optimization problem. From now on, we will refer to the evaluation function as fitness function. Accordingly, the value returned by this function will be referred to as fitness value or simply fitness.

The *learning method* is a characteristic and critical component of EDAs. Depending on the class of models used, this step involves parametric or structural learning, also known as model fitting and model selection, respectively.

Sampling methods are used to generate new solutions from the learned probabilistic models. They depend on the type of probabilistic model and the characteristics of the

ALGORITHM 1: Estimation of distribution algorithm

Generate an initial population D_0 using a *seeding method*;
If required, apply a *repairing method* to D_0;
Evaluate (all the objectives of) population D_0 using an *evaluation method*;
If required, apply a *local optimization method* to D_0;
$t = 1$;
while *The termination criterion is not met* **do**
  Select a set D_t^S of points from D_t according to a *selection method*;
  Compute a probabilistic model of D_t^S using a *learning method*;
  Sample a $D_{Sampled}$ population using a *sampling method*;
  If required, apply a *repairing method* to $D_{Sampled}$;
  Evaluate (all the objectives of) population $D_{Sampled}$ using an *evaluation method*;
  If required, apply a *local optimization method* to $D_{Sampled}$;
  Create D_t population from populations D_{t-1} and $D_{Sampled}$ using a *replacement method*;
  $t = t + 1$;
end

problem. In addition they can be conceived to deal with certain types of constraints in the solutions.

Repairing methods should be applied for constrained problems where sampled solutions may be unfeasible and some strategy to repair these solutions is available.

The *evaluation method* comprises the call to the fitness function (which measures the quality of the solutions). For multi-objective problems this may imply the evaluation of a set of functions. An advantage of EDAs and other evolutionary algorithms is that the fitness function does not have to be differentiable or even continuous.

EDAs are global optimization algorithms and their results can be improved when used together with *local optimization methods* that perform some local search departing from the current solution.

Replacement methods combine the solutions stored in the previous generation with the current set of sampled solutions. The mixing policies can help to retain the best solutions found so far, maintain the diversity in the population, etc.

Finally, the *termination criteria method* determines the stopping conditions for the EDA algorithm. These criteria can be as simple as a fixed number of generations or may imply a statistical analysis of the current population.

Mateda-2.0 provides a suitable modular framework to implement EDAs for permutation problems. The design of the new algorithms can be mainly focused on the implementation of new *learning* and *sampling methods*. In addition, for real-world permutation problems other modules could be also modified, such as local optimization or selection methods.

3. THE MALLOWS AND GENERALIZED MALLOWS MODELS

Before going into the details on probability models, some notation on permutations and distances is introduced. Throughout this section, the permutations will be in general denoted as σ or π. By π^{-1} the inverse permutation of π is denoted. The composition of σ and π is stated as $\sigma\pi$. The permutation e stands for the identity permutation, i.e., $e = 1234\dots n$. For every distance considered in this manuscript, it holds that $d(\sigma, \pi) = d(\sigma\pi^{-1}, e)$. For notational convenience, the distance to e, $d(\sigma\pi^{-1}, e)$, is also denoted as $d(\sigma\pi^{-1})$.

The Mallows model [Fligner and Verducci 1986] is a distance-based exponential probability model over permutation spaces. Given a distance over permutations, the

Mallows model is defined by two parameters: the central permutation σ_0, and the spread parameter θ. Formally, the probability of every permutation σ under the Mallows model is defined as follows:

$$P(\sigma) = \psi(\theta)^{-1} \exp(-\theta d(\sigma, \sigma_0)) \tag{1}$$

where $\psi(\theta)$ denotes the normalization constant. When $\theta > 0$, the central permutation σ_0 is that with the highest probability value (the mode), and the probability of the other $n! - 1$ permutations decreases exponentially with the distance to the central permutation. Roughly speaking, the closer a permutation σ to σ_0, the larger its probability. Moreover, when $\theta = 0$, we obtain the uniform distribution and, as θ increases, the distributions become more concentrated around the mode.

As an extension to the Mallows model, the Generalized Mallows (GM) model was proposed in [Fligner and Verducci 1986]. Like the Mallows model, the GM is exponential and unimodal. However, instead of using a single spread parameter θ, the GM model makes use of a $(n-1)$-dimensional spread parameter $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_{n-1})$, where each θ_j affects a particular position in the permutation, and n is the size of the permutation. This allows modeling a distribution with more emphasis on the consensus of certain positions of the permutation while having more uncertainty in others.

Before going into details, it is worth stating that not every distance that can be considered for the Mallows model can also be considered for the GM model. In fact, the GM model requires the distance to be decomposed as the sum of $n - 1$ terms.

$$d(\sigma, \sigma_0) = \sum_{j=1}^{n-1} S_j(\sigma\sigma_0^{-1}) \tag{2}$$

The vector grouping the terms $\mathbf{S}(\sigma\sigma_0^{-1}) = (S_1(\sigma\sigma_0^{-1}), \ldots, S_{n-1}(\sigma\sigma_0^{-1}))$ is denoted as the distance decomposition vector. The GM model, whose definition relies on this vector, is expressed as:

$$P(\sigma) = \psi(\boldsymbol{\theta})^{-1} \exp\left(\sum_{j=1}^{n-1} -\theta_j S_j(\sigma\sigma_0^{-1})\right) \tag{3}$$

In what follows, we introduce in detail the Kendall's-τ, Cayley and Ulam distances, together with the methods defined to learn and sample the Mallows and GM model under each of these distances. In addition, references to Matlab code will be given together with additional explanations. Further discussion about Mallows and GM models with the three distances, as well as several sampling and learning algorithms, can be found in [Irurozki 2014] and [Irurozki et al. 2015].

3.1. Kendall's-tau distance

The Kendall's-τ distance $d_k(\sigma, \pi)$ counts the number of pairwise disagreements between σ and π.

$$d_k(\sigma, \pi) = |\{(i,j) : i < j,\ (\sigma(i) < \sigma(j) \wedge \pi(i) > \pi(j)) \\ \vee (\pi(i) < \pi(j) \wedge \sigma(i) > \sigma(j))\}|$$

It is equivalent to counting the number of adjacent swaps to convert σ^{-1} into π^{-1}. The Kendall's-τ distance $d_k(\sigma)$ can be broken down into a distance decomposition vector $\mathbf{V}(\sigma) = (V_1(\sigma), \ldots, V_{n-1}(\sigma))$, which is also referred to as *inversion vector*, and can be expressed as follows:

$$V_j(\sigma) = \sum_{i=j+1}^{n} \mathbf{1}_{[\sigma(i)<\sigma(j)]}. \tag{4}$$

It follows that $V_j(\sigma)$ is equal to the number of items smaller than $\sigma(j)$ and it ranges for $0 \leq V_j(\sigma) \leq n-j$ for $1 \leq j \leq n$.

As an example, if $\sigma = 213645$, then $V(\sigma) = (10020)$ and $d_k(\sigma) = \sum_{j=1}^{n-1} V_j(\sigma) = 3$. The conversion from $\mathbf{V}(\sigma)$ to σ and vice versa is carried out in time $O(n^2)$. Two auxiliary methods have been defined (1) for extracting the decomposition vector V associated to a given permutation (see Table II, `vVector`), and (2) for obtaining a permutation departing from V (see Table II, `GeneratePermuFromV`).

The spread parameter $\boldsymbol{\theta}$ of the GM model under the Kendall's-τ distance is an $(n-1)$-dimensional vector. Let σ be a permutation sampled from a GM model under the Kendall's-τ distance, with parameters $\boldsymbol{\theta}$ and σ_0, where $\sigma_0(j) = i$. The spread parameter θ_j is related to position j in σ in the sense that, the larger θ_j, the larger the probability of $\sigma(j) \leq i$. In the ranking domain, when permutations are interpreted as rankings, this means that item j is ranked in the first i positions with high probability. Further details of the Mallows and GM models under this distance can be found in [Fligner and Verducci 1986].

3.2. Cayley distance

The Cayley distance $d_c(\sigma, \pi)$ counts the minimum number of swaps (not necessarily adjacent) to transform σ into π. The Cayley distance is related to the concept of cycles in σ, defined as ordered sets $\{i_1, \ldots, i_s\} \subseteq \{1, \ldots, n\}$ such that $\sigma(i_1) = i_2$, $\sigma(i_2) = i_3$, $\ldots$, $\sigma(i_s) = i_1$. When the reference permutation is the identity, $d_c(\sigma)$ equals n minus the number of cycles in σ.

The distance decomposition vector $\mathbf{X}(\sigma) = (X_1(\sigma), \ldots, X_{n-1}(\sigma))$ of the Cayley distance has length $n-1$ and each term can be expressed as follows:

$$X_j(\sigma) = \begin{cases} 0 & \text{iff } j \text{ is the largest item in its cycle in } \sigma, \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

Given $\sigma = 213645$, its corresponding cycle notation is $(21)(3)(456)$, the distance decomposition vector is $X(\sigma) = 10011$ and $d_c(\sigma) = \sum_{j=1}^{n-1} X_j(\sigma) = 3$. This conversion from σ to $\mathbf{X}(\sigma)$ can be run in time $O(n)$. In this case, as done for Kendall's-τ, two methods are also defined in Table II: xVector (to obtain the corresponding decomposition) and GeneratePermuFromX (to obtain the permutation associated to a given X decomposition).

Like the GM model under the Kendall's-τ, the GM model (Eq. 3) under the Cayley distance considers a $(n-1)$-dimensional vector spread parameter $\boldsymbol{\theta}$. So, again, the larger θ_j, the larger the probability that $\sigma(j) \leq i$. For further discussion, we refer the interested reader to [Irurozki et al. 2016].

3.3. Ulam distance

The Ulam distance $d_u(\sigma, \pi)$ counts the length of the complement of the longest common subsequence (LCS) in σ and π, i.e., the number of items which are not part of the LCS. If the reference permutation is the identity, $d_u(\sigma)$ equals n minus the length of the longest increasing subsequence (LIS).

The classical example to illustrate the Ulam distance, $d_u(\sigma, \pi)$, considers a shelf of books in the order specified by σ [Diaconis 1988]. The objective is to order the books as

specified by π with the minimum possible number of movements, where a movement consists of taking a book and inserting it into another position (delete-insert). The minimum number of movements is exactly $d_u(\sigma, \pi)$.

For example, given $\sigma = 2136457$, the length of the LIS[1] is 5 and, therefore, $d_u(\sigma) = 2$.

The computation of the Ulam distance between two given permutations has complexity $O(n\,log\,l)$ where l is the length of the longest common subsequence.

There is no distance decomposition vector for the Ulam distance. Consequently, the GM model can not be used with the Ulam distance. More details can be found in [Irurozki et al. 2014].

3.4. Learning

Learning the parameters of the Mallows or GM models given a sample of permutations is either known or conjectured to be NP-complete for all the distances considered in this manuscript. There exist several approaches to learn the models considered in this paper [Irurozki 2014]. On the one hand, EDAs do not require the best model to be learned in each generation, since approximate models can be beneficial to explore other areas of the search space. On the other hand, approximate algorithms are computationally more affordable than exact approaches, which is also desirable. The learning step is usually split into two stages as follows:

(1) Approximate the central permutation, σ_0.
(2) Estimate, by means of maximum-likelihood estimation (MLE), the spread parameter(s), θ, for the given distance.

We present three different methods for implementing step (1), which are:

— *Borda* [de Borda 1781]: Selects as central permutation the result of sorting the items in descending order according to their average position across all the input permutations. When the permutations come from a Mallows model under the Kendall's-τ distance, the Borda permutation is an asymptotically optimal estimator of the central permutation [Fligner and S.Verducci 1988]. No results exist in the literature relating the Borda permutation with Cayley or Ulam distance.
— *SetMedianPermutation*: Selects the individual in the sample that minimizes the sums of distances to the rest, given a particular distance.
— *BestPermutation*: Chooses the permutation with the best fitness.

Once the central permutation σ_0 has been obtained, the spread parameter(s) are estimated. In the case of MLE, the expression for these parameter(s) is obtained by equaling the derivative of the likelihood to zero. Although these expressions differ for each distance, in most of the cases numerical methods, such as Newton-Raphson, can be used to obtain an approximate value. Exact expressions can be found in [Irurozki et al. 2015, 2014; Irurozki 2014; Mandhani and Meila 2009].

The following example shows the call to the method declared for learning the GM model with the Kendall's-τ distance:

```
[model] = GMallows_kendall_learning(k,n,Card,SelPop,AuxFunVal,learning_params)
```

where *k* is the current generation of the EDA, and n is the number of variables (size) of the permutation. The *Card* parameter is a matrix with the dimensions of all the variables. The *SelPop* parameter is the population of permutations from which the model is learned, *AuxFunVal* is the evaluation (fitness values) of the population for

[1]Note that there can be more than one sequence with the same LIS value. In this case, there are two sequences: 13457 and 23457.

Table I. Learning methods and related auxiliary functions for the different model and distance combinations.

Main Methods	**Description**
Mallows_{kendall,cayley,ulam}_learning GMallows_{kendall,cayley}_learning	Main methods for the learning step.
Methods called by the main learning method	**Description**
{Borda,SetMedianPermutation,BestFitness}	Different methods to approximate the central permutation.
CalculateThetaParameter{K,GK,C,GC,U}	Learn θ parameter(s).
{Kendall,GKendall,Cayley,Ulam}ThetaFuncion {Kendall,GKendall,Cayley,Ulam}ThetaDevFuncion	Auxiliary methods used to define the functions to be solved by applying Newton-Raphson.
Auxiliary functions	**Description**
{Kendall,cayley,ulam}_distance	Calculate the distance between two permutations.
Mallows_{kendall,cayley,ulam}_Probability GMallows_{kendall,cayley}_Probability	Calculate the probability assigned by the model to a particular permutation.
Mallows_{kendall,cayley,ulam}_Probability_exp GMallows_{kendall,cayley}_Probability_exp	Calculate the value of the exponential part of the probability equation.
CalculatePsiConstants{K,GK,C,GC}	Calculate the normalization term.

the selected problem, and *learning_params* is the set of additional learning parameters (which can be different for each model). In all the cases –except for GM with Cayley– *learning_params* consists of the following: the first three parameters related to the Newton-Raphson method used for estimating the θ value: the *initialTheta* and *upperTheta* are the interval values for θ, and *maxit* is the maximum number of iterations allowed. The fourth parameter, *RankingFun*, indicates the function to be used for approximating the central permutation (σ_0). In the case of GM with Cayley, as Newton-Raphson is not needed, only one parameter is used (*RankingFun*).

In Table I, the different learning methods, together with related auxiliary functions are presented[2]. Different functions have been defined for each step (or even sub-step) of the learning phase, taking into account the modular design of the Mateda-2.0 toolbox. Particularly, the main learning method calls two secondary methods, one for approximating the central permutation and the other for estimating the spread parameter(s). Moreover, as this last method uses Newton-Raphson, it calls `ThetaFunction` and `ThetaDevFunction` methods (except for GM-Cayley). This way, it is straightforward to modify some parts of the code or write new code, it being enough to just change the particular method and the call to it. For example, replace Newton-Raphson by another numerical method, or include a new proposal to obtain the central permutation.

3.5. Sampling

The sampling step consists of generating permutations from the model obtained in the learning stage. The sampling algorithm depends on the distance considered in the model. Mallows and GM models under the Kendall's-τ and Cayley distances can be factorized, leading to efficient sampling algorithms. Roughly speaking, the sampling step uses the probability distributions learned for the V (Kendall's-τ) or X (Cayley) vectors , samples V (or X) vectors, and finally converts the vector to the corresponding permutation. Details of the sampling algorithms for both distances can be found in [Irurozki 2014] called Multistage Sampler. In the case of Ulam, the sampling algorithm relies on the generation of combinatorial structures, which makes the sampling algorithms

[2]In order to avoid repeating each model - distance combination, we represent the different options between keys. For example, the name of the learning method for GM with Cayley will be GMallows_cayley_learning.

Table II. Sampling methods and related auxiliary functions for the different model and distance combinations.

Main methods	Description
Mallows_{kendall,cayley,ulam}_sampling GMallows_{kendall,cayley}_sampling	Main methods for the sampling step.
Auxiliary functions	**Description**
{v,x}Vector	Decompose the permutation
GeneratePermuFrom{V,X}	Obtain a new permutation from the {v,x}vector.

Table III. Functions defined to read the instances and evaluate a solution for the four permutation-based problems implemented: TSP, PFSP, LOP, and QAP.

Methods	Description
Read{TSP,PFSP,LOP,QAP}Instance	Method to read an instance of a given problem
Eval{TSP,PFSP,LOP,QAP}	Method to evaluate a given solution (permutation)

more time consuming (details can be found in [Irurozki 2014] by the name of Distances Sampler).

The method for sampling the GM model with the Kendall's-τ distance is called:

```
[pop] = GMallows_kendall_sampling(n,model,Card,AuxPop,AuxFunVal,sampling_params)
```

where n is the number of variables (size) of the permutation, *model* is the model learned by the learning method, *Card* is a matrix with the dimensions of all the variables, *AuxPop* is the population from which the model was learned, and *AuxFunVal* is the evaluation (fitness values) of the *AuxPop* data set. The *sampling_params* are the additional sampling parameters (we follow the general scheme of the Mateda-2.0 toolbox). In this case, our methods only require one parameter, *N*, which is the number of new individuals to be generated.

In Table II the different sampling methods, together with related auxiliary functions are presented[3].

4. PERMUTATION PROBLEMS

Mateda-2.0 toolbox includes implementations of discrete and continuous optimization problems. Following the same idea, we have incorporated four problems defined on permutations: Traveling Salesman Problem (TSP), Permutation Flowshop Scheduling Problem (PFSP), Linear Ordering Problem (LOP), and Quadratic Assignment Problem (QAP). These problems are challenging and they appear frequently in the literature.

Table III describes the modules implemented for each optimization problem. Each problem is defined by two modules: one for reading the instance file and processing the parameters (`Read{TSP,PFSP,LOP,QAP}instance`), and the other for evaluating the solutions (permutations) (`Eval{TSP,PFSP,LOP,QAP}`).

The `Read{TSP,PFSP,LOP,QAP}instance` module takes as the only parameter the filename of the instance to load. The output is a global variable with the name of the problem that contains the parameters needed by EDA.

For example, to read a QAP problem, the method `ReadQAPInstance` is called:

```
ReadQAPInstance(InstanceName);
```

[3]As with the learning phase, we represent the different options between keys, V (or v) is for Kendall's-τ and X (or x) is for Cayley

where *InstanceName* is the path and filename of the instance to read. This module creates a global variable, in this case called *QAPInstance*, that holds the data of the problem. The global variable is used in order to avoid unnecessary traffic of arguments across the main `RunEDA` process.

For the evaluation of a solution in the QAP problem, the method `EvalQAP` is called:

```
[val] =  EvalQAP(permutation);
```

where *permutation* contains the permutation that will be evaluated. As previously mentioned, the instance data is taken from the global variable of the problem, in this case called *QAPInstance*. The output parameter *val* is the fitness of the permutation.

All problems described in Table III have the same input parameter. For the four problems, the methods to read and evaluate the problem are called in the same way. However, regarding the global variables, each problem has its own structure and, thus, the information stored for each problem is different.

— Traveling Salesman Problem (TSP): This problem is described by one matrix of size $n \times n$ containing the distances between the cities. The TSP implementation can work with both symmetric and asymmetric matrices. Once the data is read, the distance matrix of the problem and the number of cities are stored, in this order, in a global variable named *TSPInstance*.
— Permutation Flowshop Scheduling Problem (PFSP): The information about this problem is contained in one matrix of size $m \times n$ containing the processing times of executing job j, $j = 1, 2, \ldots, n$ in machine i, $i = 1, 2, \ldots, m$. The matrix that contains the processing times, the number of machines, and the number of jobs are stored, in this order, in a global variable named *PFSPInstance*.
— Linear Ordering Problem (LOP): It is described by one matrix of size $n \times n$ with arbitrary natural numbers. Once the data is read, the matrix and problem size are stored, in this order, in a global variable named *LOPInstance*.
— Quadratic Assignment Problem (QAP): The information about this problem is contained in two matrices of sizes $n \times n$. The first matrix contains the flow between the facilities and the second one the distances between the locations. Once the data is read, the distance matrix, flow matrix and problem size are stored, in this order, in a global variable named *QAPInstance*.

5. EXAMPLES

In this section, we will go through the whole process of optimizing an instance of the Linear Ordering Problem (LOP) [Ceberio et al. 2014]. Additional examples, instances, source code and program documentation can be downloaded from the web-site of the toolbox.[4]

First, in lines 1 to 3, the global variable of the problem is defined and the instance is loaded (in this example the `'LOP-example-instance-30'` file). This instance corresponds to a problem of size 30.

```
1 global LOPInstance
2 ReadLOPInstance('LOP-example-instance-30');
3 [matrix, size] = LOPInstance{:};
```

Then, we declare the general parameters of the EDA: size of the solution and cardinality of the variables (given by the instance), size of the population, the evaluation function to be used, and the `cache` (explained later).

[4] http://www.sc.ehu.es/ccwbayes/files/perm_mateda.zip

```
4 NumbVar = size;
5 Card = [ones(1,NumbVar); NumbVar*ones(1,NumbVar)];
6 PopSize = 10*NumbVar;
7 F = 'EvalLOP';
8 cache  = [0,0,1,0,0];
```

The toolbox allows storing different descriptors about the behavior of the EDA algorithm during the optimization process. These descriptors are stored in the output variable *Cache* (line 17). To specify the desired output information, a vector (*cache*) specifying which components of the algorithm will be stored is used (line 8). `cache(i) = 1` determines whether the i-th component of EDA $(i = 1, 2, \ldots, 5)$ will be saved in each generation (`cache(i) = 0` otherwise). The five components considered are the following:

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

Now, we tackle the design of our particular EDA. First, see line 9, the method used to initialize the population is chosen. In this case, a random initialization would be preferred. With respect to the selection operator, lines 10-11, the `NumbVar/PopSize` percentage of the solutions with the best fitness are selected by truncation. In relation to the replacement operator, line 12, the population is updated by adding the newly created solutions and preserving the `PopSize` solutions with the best fitness. As regards the stopping criterion, line 13, this is set to 500 generations of the algorithm.

```
9 edaparams{1} = {'seeding_pop_method','InitPermutations',{}};
10 selparams(1:2) = {NumbVar/PopSize,'fitness_ordering'};
11 edaparams{2} = {'selection_method','truncation_selection',selparams};
12 edaparams{3} = {'replacement_method','pop_aggregation',{'fitness_ordering'}};
13 edaparams{4} = {'stop_cond_method','max_gen',{500}};
```

Finally, we specify the probabilistic model that is going to be used. In this example, we propose using the Mallows model under the Kendall-τ distance. To this end, the learning and sampling methods must be detailed (see lines 14-15):

```
14 edaparams{5}= {'learning_method','Mallows_kendall_learning',{0.001,10,100,'Borda'}};
15 edaparams{6}= {'sampling_method','Mallows_kendall_sampling',{PopSize-1,1}};
16 edaparams{7} = {'local_opt_method','greedy_Insert',{100}};
```

With respect to the learning method, *Borda* is used to approximate the central permutation, and the spread parameters are estimated in the range $[0.001, 10]$ with a maximum of 100 iterations of the Newton-Raphson algorithm. According to the sampling step, `PopSize-1` new solutions are sampled from the model at each generation. A local optimization method is also added to the EDA. In the example shown in line 16, a greedy insert local optimizer with a maximum number of 100 movements is used.

Once the algorithm has been configured, it is executed by calling the function

```
17 [AllStat,Cache]=RunEDA(PopSize,NumbVar,F,Card,cache,edaparams)
```

In addition to the variable *Cache*, the output variable *AllStat* stores the following statistics of each EDA generation:

— Information about maximum, mean, median, minimum, and variance of fitness values in the current population.

— Best individual (according to an ordering criterion).
— Number of different individuals.
— Information about maximum, mean, median, minimum, and variance values of every variable in the current population.

From the information stored in `AllStat`, it is possible to illustrate the optimization process followed by the algorithm. For example, Fig 1 shows the best fitness and mean fitness of the individuals in the population across 500 generations. Fig 2 shows the best permutations obtained in each generation (for the sake of clarity, only a subset of the 500 permutations is shown). In addition, using the information about the probabilistic model stored in `Cache` (see line 8, where `cache(3)` was set to 1), a figure showing the evolution (convergence) of the θ variable has been plotted (see Fig 3).

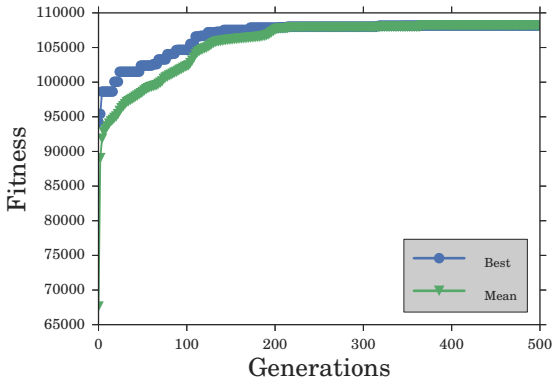

Fig. 1. Best and mean fitness obtained across 500 generations by the Mallows EDA proposed in the example when optimizing a LOP problem.

6. CONCLUSIONS

Permutation-based optimization problems are present in many real-world scenarios, such as those related to logistics, planning, etc. Recently, new estimation of distribution algorithms have been proposed to solve this kind of problems in an efficient way. In this paper, we propose an extension of the Mateda 2.0 toolbox, designed for solving permutation-based problems. Based on two distance-based exponential probability models, the Mallows and Generalized Mallows models, the toolbox implements the functions to run both models under the Kendall's-τ, Cayley, Ulam and Hamming distances. In order to provide a testbed of functions, four classical permutation problems have also been implemented: Traveling Salesman Problem, Permutation Flowshop Scheduling Problem, Linear Ordering Problem, and Quadratic Assignment Problem. For the sake of illustrating the different functionalities, the paper introduces a complete example of the definition of a problem, design and execution of the algorithm, and finally an analysis and visualization of the results. Additionally, the toolbox permits, without running EDAs, to make inference on permutation data by learning and sampling Mallows and Generalized Mallows models on the different metrics explained in this paper.

The algorithms have been implemented taking into account the modular nature of the original Matlab framework. In this sense, it will be easy for a future user to extend presented EDAs, including new probabilistic models, such as Plackett-Luce [Plackett

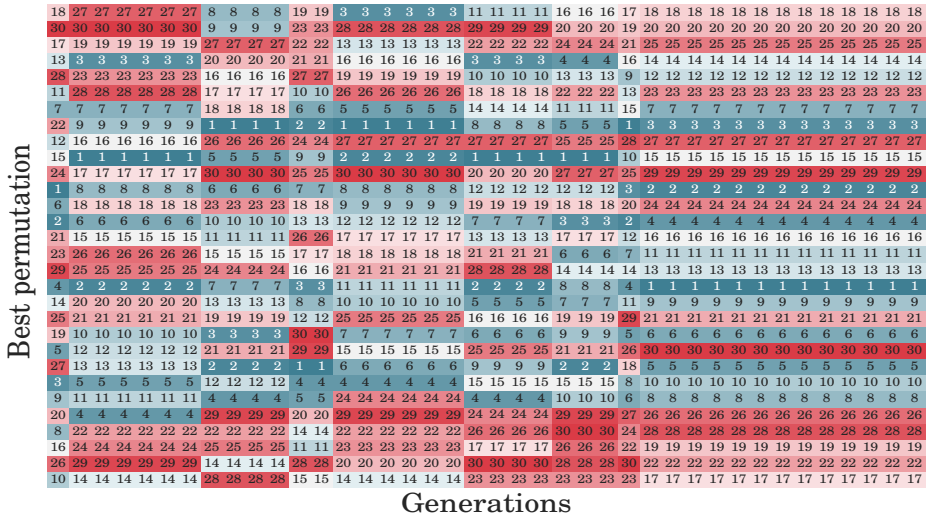

Fig. 2. Best permutations (in columns) obtained in each generation of the Mallows EDA proposed in the example when optimizing a LOP problem. For the sake of clarity, only a subset of the 500 permutations is shown.

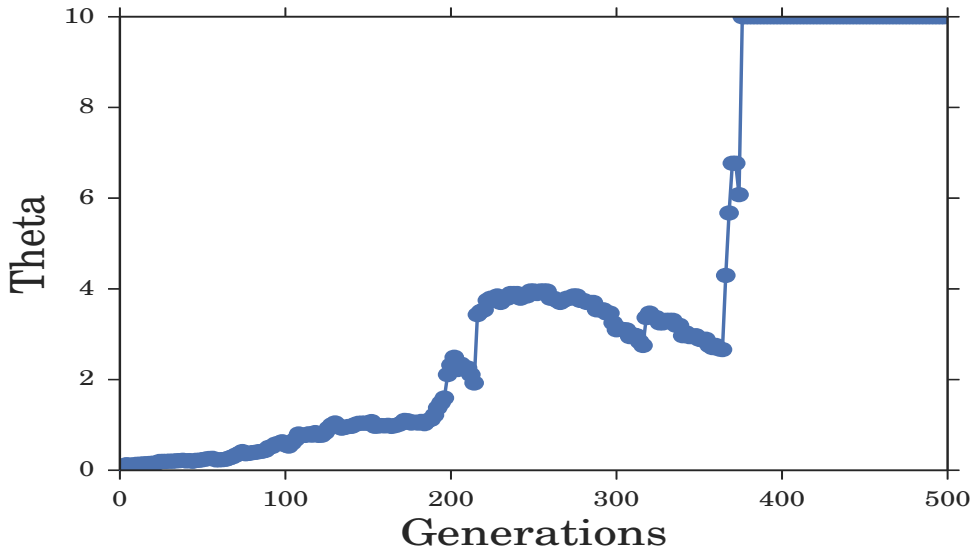

Fig. 3. Evolution of the θ value (a parameter of the probabilistic model) across generations.

1975; Luce R. 1959], add new distance metrics to existing models: for example, Hamming [Deza et al. 1998] or even implement mixtures of the probabilistic models [Santamaria et al. 2015].

Acknowledgments

ACKNOWLEDGMENTS

This work has been partially supported by the IT-609-13 program (Basque Government) and the TIN2013-41272P (Spanish Ministry of Science and Innovation).

REFERENCES

Ruben Armañanzas, Iñaki Inza, Roberto Santana, Yvan Saeys, Jose Flores, Jose Lozano, Yves Peer, Rosa Blanco, Victor Robles, Concha Bielza, and Pedro Larranaga. 2008. A review of estimation of distribution algorithms in bioinformatics. *BioData Mining* 1, 1 (2008), 6. DOI:http://dx.doi.org/10.1186/1756-0381-1-6

John E Beasley, Mohan Krishnamoorthy, Yazid M Sharaiha, and D Abramson. 2000. Scheduling aircraft landings—the static case. *Transportation Science* 34, 2 (2000), 180–197.

Alexander E. I. Brownlee, Martin Pelikan, John A. W. McCall, and Andrei Petrovski. 2008. An application of a multivariate estimation of distribution algorithm to cancer chemotherapy. In *Proceedings of the 10th Genetic and Evolutionary Computation Conference (GECCO '08)*, C. Ryan and M. Keijzer (Eds.). 463–464.

Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pitsoulis. 1998. The Quadratic Assignment Problem. (1998).

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. 2012. A review on Estimation of Distribution Algorithms in Permutation-based Combinatorial Optimization Problems. *Progress in Artificial Intelligence* 1, 1 (January 2012), 103–117.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. 2014. A Distance-based Ranking Model Estimation of Distribution Algorithm for the Flowshop Scheduling Problem. *IEEE Transactions on Evolutionary Computation* 18, 2 (April 2014), 286 – 300.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. 2015. A Review of Distances for the Mallows and Generalized Mallows Estimation of Distribution Algorithms. *Computational Optimization and Applications* (March 2015), 1–20.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. 2011a. Introducing the Mallows Model on Estimation of Distribution Algorithms. In *Proceedings of International Conference on Neural Information Processing (ICONIP) (Lecture Notes in Computer Science)*, Bao-Liang Lu, Liqing Zhang, and James T. Kwok (Eds.). Springer, 461–470.

Josu Ceberio, Alexander Mendiburu, and Jose Antonio Lozano. 2011b. A Preliminary Study on EDAs for Permutation Problems Based on Marginal-based Models. In *Proceeding of the 13th annual conference on Genetic and Evolutionary Computation*, Natalio Krasnogor and Pier Luca Lanzi (Eds.). ACM, 609–616.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. 2013. The Plackett-Luce Ranking Model on Permutation-based Optimization Problems. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*. 494 – 501.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. 2014. The Linear Ordering Problem Revisited. *European Journal of Operational Research* 241, 3 (2014), 686–696.

Jean C. de Borda. 1781. Memoire sur les elections au scrutin. *Histoire de l'Academie Royale des Science* (1781).

Michael Deza, Liens-ecole Normale Supérieure, and Tayuan Huang. 1998. Metrics on Permutations, a Survey. In *Journal of Combinatorics, Information and System Sciences*. Citeseer.

Persi Diaconis. 1988. *Group representations in probability and statistics*. Institute of Mathematical Statistics, Hayward, CA. vi+198 pages.

Michael A. Fligner and Joseph S.Verducci. 1988. Multistage ranking models. *J. Amer. Statist. Assoc.* 83, 403 (1988), 892–901.

Michael A. Fligner and Joseph S. Verducci. 1986. Distance based ranking Models. *Journal of the Royal Statistical Society* 48, 3 (1986), 359–369.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

David E. Goldberg and Robert Lingle Jr. 1985. Alleles Loci and the Traveling Salesman Problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 154–159.

Jatinder N. D. Gupta and Edward F. Stafford. 2006. Flow shop scheduling research after five decades. *European Journal of Operational Research* 169, 3 (2006), 699–711.

Jonathan Huang, Carlos Guestrin, and Leonidas Guibas. 2009. Fourier Theoretic Probabilistic Inference over Permutations. *Journal of Machine Learning Research (JMLR)* 10 (May 2009), 997–1070.

Ekhine Irurozki. 2014. *Sampling and Learning Distance-based Probability Models for Permutation Spaces*. Ph.D. Dissertation. Faculty of Computer Science, University of the Basque Country.

Ekhine Irurozki, Borja Calvo, and Jose A Lozano. 2014. Mallows Model under the Ulam Distance: a Feasible Combinatorial Approach. In *Neural Information Processing System (NIPS), Analysis on Rank Data workshop*. University of the Basque Country. DOI:http://dx.doi.org/EHU-KZAA-TR;2014-04

Ekhine Irurozki, Borja Calvo, and Jose A. Lozano. 2015. PerMallows: An R Package for Mallows and Generalized Mallows Models. *Journal of Statistical Software* (2015).

Ekhine Irurozki, Borja Calvo, and Jose A. Lozano. 2016. Sampling and Learning Mallows and Generalized Mallows Models Under the Cayley Distance. *Methodology and Computing in Applied Probability* (2016), 1–35. DOI:http://dx.doi.org/10.1007/s11009-016-9506-7

Shan Jiang, Ahmet K. Ziver, Jonathan N. Carter, Christopher C. Pain, Antony J.H. Goddard, Simon Franklin, and H.J. Phillips. 2006. Estimation of Distribution Algorithms for nuclear reactor fuel management optimisation. *Annals of Nuclear Energy* 33, 11-12 (2006), 1039–1057. DOI:http://dx.doi.org/DOI:10.1016/j.anucene.2006.03.012

Tjalling C. Koopmans and Martin J. Beckmann. 1955. *Assignment Problems and the Location of Economic Activities*. Cowles Foundation Discussion Papers 4. Cowles Foundation for Research in Economics, Yale University.

Pedro Larrañaga and Jose A. Lozano. 2002. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

Jose A. Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea. 2006. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Duncan Luce R. 1959. *Individual Choice Behavior*. Wiley, New York.

Colin L. Mallows. 1957. Non-null ranking models. *Biometrika* 44, 1-2 (1957), 114–130.

Bhushan Mandhani and Marina Meila. 2009. Tractable Search for Learning Exponen-

tial Models of Rankings. *Journal of Machine Learning Research (JMLR)* 5 (2009), 392–399.

John McCall, Alexander E. I. Brownlee, and Siddhartha Shakya. 2012. Applications of distribution estimation using Markov network modelling (DEUM). In *Markov Networks in Evolutionary Computation*, S. Shakya and R. Santana (Eds.). Springer, 193–207.

Alexander Mendiburu, Jose Miguel-Alonso, Jose A. Lozano, Miren Ostra, and Carlos Ubide. 2006. Parallel EDAs to create multivariate calibration models for quantitative chemical applications. *J. Parallel and Distrib. Comput.* 66, 8 (2006), 1002–1013.

Rebecca J. Parsons, Stephanie Forrest, and Christian Burks. 1995. Genetic algorithms, operators, and DNA fragment assembly. *Machine Learning* 21, 1-2 (1995), 11–33.

Martin Pelikan, David E. Goldberg, and Fernando G. Lobo. 2002. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications* 21, 1 (2002), 5–20.

Martin Pelikan, Kumara Sastry, and Erick Cantú-Paz. 2006. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications (Studies in Computational Intelligence)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Robin L. Plackett. 1975. The Analysis of Permutations. *Journal of the Royal Statistical Society* 24, 10 (1975), 193–202.

Ramon Sagarna and Jose A. Lozano. 2006. Scatter Search in software testing, comparison and collaboration with Estimation of Distribution Algorithms. *European Journal of Operational Research* 169, 2 (2006), 392–412.

Josian Santamaria, Josu Ceberio, Roberto Santana, Alexander Mendiburu, and Jose A. Lozano. 2015. Introducing Mixtures of Generalized Mallows in Estimation of Distribution Algorithms. In *X Congreso Español de Metaheuristicas, Algoritmos Evolutivos y Bioinspirados - MAEB 2015*. Mérida-Almendralejo, Spain, 19–25.

Roberto Santana, Concha Bielza, Pedro Larranaga, Jose A Lozano, Carlos Echegoyen, Alexander Mendiburu, Ruben Armananzas, and Siddartha Shakya. 2010. Mateda-2.0: Estimation of distribution algorithms in MATLAB. *Journal of Statistical Software* 35, 7 (2010), 1–30.

Marta Soto, Alberto Ochoa, Yasser González-Fernández, Yanely Milanés, Adriel Álvarez, Diana Carrera, and Ernesto Moreno. 2012. Vine Estimation of Distribution Algorithms with Application to Molecular Docking. In *Markov Networks in Evolutionary Computation*, S. Shakya and R. Santana (Eds.). Springer, 175–190.

Paolo Toth and Daniele Vigo. 2001. *The vehicle routing problem*. Society for Industrial and Applied Mathematics.

Shigeyoshi Tsutsui and Gordon Wilson. 2004. Solving Capacitated Vehicle Routing Problems Using Edge Histogram Based Sampling Algorithms. In *Proceedings of the IEEE Conference on Evolutionary Computation, Portland, Oregon (USA)*. 1150–1157.

Bo Yuan, Maria E. Orlowska, and Shazia Wasim Sadiq. 2007. Finding the Optimal Path in 3D Spaces Using EDAs - The Wireless Sensor Networks Scenario. In *Adaptive and Natural Computing Algorithms*. Springer, 536–545.