Solving Permutation Problems with Estimation of Distribution Algorithms and Extensions Thereof

Josu Ceberio

Advisors: Alexander Mendiburu and Jose A. Lozano

Intelligent Systems Group

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Konputazio Zientziak eta Adimen Artifizialaren Saila
Departamento de Ciencias de la Computación e Inteligencia Artificial

Solving Permutation Problems with Estimation of Distribution Algorithms and Extensions Thereof

by

Josu Ceberio Uribe

Supervised by Alexander Mendiburu and Jose A. Lozano

Dissertation submitted to the Department of Computer Science and Artificial Intelligence of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, December 2014

To my family
and friends

Acknowledgments

First of all, I would like to thank my advisors Alexander Mendiburu and Jose Antonio Lozano. Without their wise guidance, this dissertation would never have been possible. Your patience and encouragement, as well as the scientific discussions in the *running* sessions, have been essential in the development of this dissertation. I hope to continue working together in the future. Thank you very much.

I thank Ekhine Irurozki for her help in the implementation of the mathematical procedures throughout the dissertation. She has been an essential part of this work as well.

I also owe a debt of gratitude to Prof. Ke Tang for allowing me to complete a stay as a visitor in his research group at the University of Science and Technology of China. And also to Dr. Thomas Weise for the occasional discussions that have certainly enriched this work.

Special thanks go to to my friends in the Intelligent Systems Group with whom I shared these four years, and also to the members of other research groups at the Faculty of Computer Science. They are part of this work.

I would like to acknowledge the great work of John Kennedy, whose comments on English writing have improved the readability of the published contributions.

I am grateful to the Basque Government for the financial support during the last four years. In addition, this work has been partially supported by the Saiotek and Research Groups 2007-2012 (IT-242-07) programs (Basque Government), TIN2010-14931 and TIN2013-41272P (Ministry of Science and Technology), COMBIOMED-RD07/0067/0003 network in computational biomedicine (Carlos III Health Institute), and by the NICaiA Project PIRSES-GA-2009-247619 (European Commission).

Finally, I would like to thank the support of my parents, brother and friends in Ondarru and Gernika. This thesis is also for them.

Contents

1

Preliminaries

In computer science, optimisation is the topic that consists of searching for the best solution(s) from a set of available solutions with regard to some criteria. Due to the importance that this discipline has for social, financial or industrial activities, optimisation has been a prominent research topic for several decades. Particularly, scientific fields such as artificial intelligence, machine learning or operational research have addressed optimisation problems generating a vast amount of knowledge about how to efficiently solve specific types of problems.

Optimization problems seem to divide naturally into two categories: those whose solutions are defined as *real-valued* variables, known as *continuous* problems, and those whose solutions are defined as *discrete* variables, also referred to as *combinatorial* problems. In continuous problems, we are generally looking for a vector of real numbers in an infinite and non-numerable subset of $\mathbb{R}^n$, where n denotes the number of variables involved in the problem. In contrast, in combinatorial problems, we are looking for an item from a finite or countably infinite set– typically a vector of integer values, a subset, a graph or a *permutation*. Due to the influence that the characterisation of a solution has in the development of algorithms, the proposed schemes for optimisation, either exact, heuristic or metaheuristic, have followed different research paths.

In this dissertation, we are especially interested in the study of a specific subset of combinatorial optimisation problems. We refer to those whose solutions are naturally codified as permutations. These problems, from now on *permutation-based problems* or simply *permutation problems*, will be the central theme of this dissertation.

In modern combinatorics, permutations are probably among the richest combinatorial structures [164]. Motivated principally by their versatility – ordered set 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, but particularly, in logistic problems such as routing [173], scheduling [71] or assignment [92].

In 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). Following the mathematical notation, the search space is usually denoted as $\mathbb{S}_n$, in reference to the *symmetric group* of size n. Motivated principally by the factorial cardinality of the search space, permutation problems are known as very hard problems when n goes above a relatively small number. For instance, when $n = 10$, the search space of solutions is compound of 3628800 permutations. And when, $n = 20$, it is compound of more than 2×10^{18} solutions. Indeed, the work of Garey and Johnson [63] 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 a reasonable time span. 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, particle swarm optimisation, ant colony optimisation, simulated annealing and genetic algorithms are some of the most cited metaheuristics.

In this dissertation, we are especially interested in a subset of metaheuristic algorithms that belongs to the field of evolutionary computation; we refer to Estimation of Distribution Algorithms (EDAs). Introduced for the first time by Muhlenbein et al. [128], and studied later by Larrañaga and Lozano [96, 106] and Pelikan [137], EDAs constitute a powerful evolutionary algorithm for solving combinatorial problems. Based on the same principles of natural selection and evolution of populations, EDAs use explicit probability distributions to lead the optimization towards promising areas of the search space. Permutation problems, however, represent a real challenge for EDAs, since classical probability distributions on the discrete or continuous domains can not be efficiently adapted to model permutation solutions. Classical probability notions such as variable *independence* are not naturally translated into the domain of permutations, since, in contrast to integer problems, in a permutation any two given positions can not have the same value by any chance. This simple constraint, known as the *mutual exclusivity constraint* [76], requires in general a more complex mathematical machinery in order to efficiently deal with the permutation nature of the solutions.

With the aim of taking a step forward in the application of EDAs on permutation problems, the first part of this dissertation is devoted to studying the existing approaches and to proposing novel EDAs by introducing suitable probability models for permutation domains that are able to efficiently capture the interrelations between the items in the solutions. Particularly, distance-based probability models [56] and Thurstone order statistic models [109, 142] have been considered in the framework of EDAs.

In addition to proposing new metaheuristic algorithms, the research community has also followed other research lines in order to study optimisation problems. One that has captured the attention of scientists the most is the fitness landscapes analysis. A *fitness landscape* is understood as a triple (Ω, N, f), where Ω is the search space of solutions, f stands for the fitness function to optimise, and the neighbourhood operator N assigns a set of neighbouring solutions $N(x) \in \Omega$ to every solution $x \in \Omega$.

Generally, when solving a combinatorial optimisation problem, in order to design efficient heuristic search algorithms, it is beneficial to have a deep understanding of the structure of the fitness landscapes generated under the different neighbourhood systems. In this sense, most of the studies have focused on analysing the *ruggedness* of the landscape, which provides a notion of the complexity of solving a given problem with local search type metaheuristics. Unfortunately, there is no consensus on which is the best way to measure the ruggedness of fitness landscapes. The autocorrelation coefficient [162], the fitness distance correlation [85] or the estimation of the number of local optima [75] are some of the most remarkable metrics. Although nowadays it is believed that it is difficult to use a small number of measures to describe the complexity of an instance of a combinatorial optimisation problem [171].

For that reason, identifying the characteristics of a problem that influence the ruggedness of the generated landscape is a fundamental research topic that still deserves further study nowadays. In this research line, the second part of the dissertation is devoted to studying the fitness landscape generated for the linear ordering problem under the *insert* neighbourhood (the most efficient one). Particularly, we analyse the manner in which the components of the problem, coupled with the neighbourhood operator, generate specific landscape shapes.

In addition to the fitness landscape analysis, there is a part of landscape theory that, despite its high complexity, can shed some light on the understanding of the structure of optimisation problems. We refer to the spectral landscape theory [187]. The last part of the thesis is devoted to this topic. The basic idea of this theory is to interpret the configuration space induced by a neighbourhood operator such as a Laplacian matrix, and to discuss the fitness function in terms of the regularities of this matrix. If the neighbourhood operator has certain symmetry and regularity properties, then it is possible to find a basis of elementary landscapes in which to decompose the fitness landscape [40]. Known as *elementary landscape decomposition*, this methodology permits us to decompose the fitness landscape, and based on this decomposition, optimise individually each of the elementary landscapes reformulating single-objective problems into multi-objective problems.

Before explaining in detail the contributions of this thesis, an introductory part will be devoted to explaining general background aspects that are related to the dissertation: permutation problems and EDAs. Further details of any topic or scientific discipline related with the aforementioned elements, but not directly used throughout the thesis, can be consulted in different works

that will be cited in the appropriate places. The rest of the current chapter is organised as follows: Section 1.1 defines permutation problems and introduces four examples that will be addressed throughout the thesis. In Section 1.2, a detailed background of EDAs is given. Finally, Section 1.3 presents the contents of the different chapters constituting this dissertation.

1.1 Permutation-based Problems

In combinatorics, a *permutation* is understood as a bijection σ of the set of natural numbers $\{1, \ldots, n\}$ onto the set $\{1, \ldots, n\}$. Permutations are usually represented in the classic one-line notation, i.e., as an ordered list of the set $\{1, \ldots, n\}$. We say that item j is in position i in σ when $\sigma(i) = j$. Alternatively, σ^{-1} denotes the inverse of the permutation σ, and $\sigma^{-1}(j)$ stands for the position of item j in σ (denoted also as $\sigma\langle j \rangle$). For instance, given the permutation $\sigma = 34215$, we say that the item in the first position is item 3, i.e., $\sigma(1) = 3$, and similarly, we say that the position of item 1 is 4, $\sigma^{-1}(1) = 4$. Note that, by definition, for any permutation σ in $\mathbb{S}_n$, $\sigma(i) \neq \sigma(j)$ for all $i \neq j$.

In the literature, permutations are often referred to as *rankings* or *orderings*, depending on the particular interpretation considered in each field. In order to avoid misunderstandings in this sense, we will avoid both terms, using the words *permutation*, *solution* or *individual* depending on the framework. Following the notation used in the literature, we will use the Greek letters σ, π, τ and γ to denote permutations.

A permutation-based problem consists of finding the best solution for a *fitness function* f:

$$\begin{aligned} f : \mathbb{S}_n &\longrightarrow \mathbb{R} \\ \sigma &\longmapsto f(\sigma) \end{aligned}$$

where $\mathbb{S}_n$ stands for the search space of solutions and is formed by the group of all the permutations of size n (*symmetric group*). In order to optimise f, one has to find a solution $\sigma^\star \in \mathbb{S}_n$ with minimum[1] *fitness value*, that is, $f(\sigma^\star) \leq f(\sigma) \ \ \forall \sigma \in \mathbb{S}_n$.

The literature contains a wide variety of permutation problems from different areas. Just to name a few, the traveling salesman problem [67], the quadratic assignment problem [92], the linear ordering problem [117, 155], the permutation flowshop scheduling problem [71], the vehicle routing problem [173], the DNA fragment assembly problem [135, 160] or the aircraft landing problem [13] are some examples of permutation based combinatorial optimisation problems.

[1] With the exception of the linear ordering problem, which is generally stated as a maximisation problem, during this dissertation we will deal with minimisation problems. Note that a maximisation problem can be easily transformed to a minimisation one by negating f, and vice-versa.

Obviously, behind the permutation nature of the solutions, each of the problems enumerated above introduces its particular challenges that need to be individually faced. Thus, it is dangerous to extrapolate the conclusions drawn on one problem to the whole class. In order to provide a solid ground to the contributions in this dissertation, we have designed a representative benchmark of problems that gathers the four most recurrent permutation problems in the literature: the traveling salesman problem, the linear ordering problem, the permutation flowshop scheduling problem and the quadratic assignment problem. These four problems have different characteristics and, therefore, they provide a suitable benchmark to validate the proposed advances. In what follows, we will introduce them one-by-one, highlighting in each case the most relevant characteristics to take into account. In addition, at the end of the section we present some insights about the particularities of each problem.

1.1.1 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) [67] is the problem of finding the shortest tour to go over n different cities visiting each city only once and returning to the city of departure. A solution is described as a permutation σ of cities, where $\sigma(i) = j$ represents that the city j is visited in the i-th stage.

Let us consider, with illustrative purposes, a TSP of 4 cities and the solution $\sigma = 3241$. The tour described in σ departs from city 3, then goes through 2, 4 and 1, and finally comes back to 3.

Formally, given a matrix $\mathbf{D} = [d_{i,j}]_{n\times n}$ with the distances between all the pairs of cities, the fitness function f is defined as the sum of the distances of going through all the cities in the order specified by σ:

$$f(\sigma) = \sum_{i=2}^{n} d_{\sigma(i-1),\sigma(i)} + d_{\sigma(n),\sigma(1)}$$

As we assume that the first city of the tour is not fixed, the TSP is a problem with symmetric solutions, since each tour can be represented by $2n$ different permutations for symmetric instances, and by n different permutations for asymmetric instances [2]. For example, solution $\sigma' = 1324$ represents the same city-tour that $\sigma = 3241$ does, therefore $f(\sigma) = f(\sigma')$, nonetheless, note that $\sigma \neq \sigma'$.

1.1.2 Linear Ordering Problem

In the Linear Ordering Problem (LOP) [117], we are given a matrix $\mathbf{B} = [b_{i,j}]_{n\times n}$ of integer values and the goal is to determine a simultaneous permutation of the rows and columns of $\mathbf{B}$ such that the sum of the entries above the

[2] When an instance is symmetric, $d_{i,j} = d_{j,i}$ for all $i, j \in \{1, \ldots, n\}$. On the contrary, when an instance is asymmetric, the previous equality is not guaranteed. In this dissertation we will assume symmetric TSP instances.

main diagonal is maximised (or equivalently, the sum of the entries below the main diagonal is minimised). A solution of the LOP is codified as a permutation of length n where each item $\sigma(i)$ means that the entries of the $\sigma(i)$-th row and column in matrix $\mathbf{B}$ are reallocated to the i-th row and column. The fitness function is defined as follows:

$$f(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{\sigma(i),\sigma(j)}$$

1.1.3 Permutation Flowshop Scheduling Problem

In the Permutation Flowshop Scheduling Problem (PFSP) [6, 9] n jobs ($i = 1, \ldots, n$) have to be scheduled on m machines ($j = 1, \ldots, m$) in such a way that a given criterion is minimized. A job consists of m operations and the j-th operation of each job must be processed on the j-th machine for a given specific processing time without interruption. The processing times are given by the matrix $\mathbf{P} = [p_{i,j}]_{n \times m}$ of non-negative numbers. Every job is available at time zero and at a given time, a job can start on the j-th machine when its $(j-1)$-th operation has finished on machine $(j-1)$, and machine j is idle.

A solution for the PFSP is codified as a permutation σ of n jobs, where $\sigma(i)$ is the job scheduled in the i-th position. For instance, for $n = 3$, the sequence $\sigma = 231$ denotes that job 2 is scheduled in the first position, i.e. $\sigma(1) = 2$, job 3 is scheduled second, and job 1 is scheduled third.

Inversely to the previous problems, different criteria have been proposed when optimising the PFSP. On the one hand, the total flow time (TFT) measures the sum of the times that each job remains in the flowshop, and its optimisation leads to the stable utilisation of the resources. Given a permutation σ of jobs, the TFT is defined as

$$f(\sigma) = \sum_{i=1}^{n} c_{\sigma(i),m} \tag{1.1}$$

where $c_{\sigma(i),m}$ stands for the completion time of job $\sigma(i)$ at machine m. An alternative criterion is to minimise the makespan, which measures the total time needed to process the n jobs in the m machines. Particularly, given a permutation σ of jobs, the makespan is calculated as

$$f(\sigma) = c_{\sigma(n),m} \tag{1.2}$$

For both criteria, the completion time of job $\sigma(i)$ on machine j, $c_{\sigma(i),j}$, is recursively calculated as

$$c_{\sigma(i),j} = \begin{cases} p_{\sigma(i),j} & i = j = 1 \\ p_{\sigma(i),j} + c_{\sigma(i-1),j} & i > 1, j = 1 \\ p_{\sigma(i),j} + c_{\sigma(i),j-1} & i = 1, j > 1 \\ p_{\sigma(i),j} + \max\{c_{\sigma(i-1),j}, c_{\sigma(i),j-1}\} & i > 1, j > 1 \end{cases} \tag{1.3}$$

where $p_{\sigma(i),j}$ stands for the processing time required by job $\sigma(i)$ in machine j.

1.1.4 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) [92] is the problem of allocating a set of facilities to a set of locations, with a cost function associated to the distance and flow between the facilities. The objective is to assign each facility to a location such that the total cost is minimized. Specifically, we are given two matrices with real values $\mathbf{D} = [d_{i,j}]_{n\times n}$ and $\mathbf{H} = [h_{k,l}]_{n\times n}$, where $d_{i,j}$ is the distance between location i and location j, and $h_{k,l}$ is the flow between facility k and facility l. Given n facilities, the solution of the QAP is codified as a permutation σ, where $\sigma(i) = j$ represents that the facility j is allocated in the i-th location. The fitness value of any solution σ is calculated by the following function:

$$f(\sigma) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j}h_{\sigma(i),\sigma(j)}$$

1.1.5 Some insights into the problems

Although the four problems above are codified naturally as permutations, the definition of the fitness functions has nothing in common. In order to provide some insights in this sense, in this section, we briefly analyse them in terms of contribution of the items in the permutation to the fitness function. These simple comments might be useful to understand the contents introduced in latter chapters.

As regards the TSP, the contribution of an item $\sigma(i)$ to f is clearly determined by the adjacent items $\sigma(i-1)$ (previous) and $\sigma(i+1)$ (posterior). Note that each entry in the matrix $\mathbf{D}$ is defined by two cities, and thus, the contribution of the item $\sigma(i)$ to the fitness function is defined as the sum of $d_{\sigma(i-1),\sigma(i)}$ and $d_{\sigma(i),\sigma(i+1)}$. Taking into account the symmetric nature of the TSP solutions, it can be seen that the meaningful information of a solution is given by the relative ordering of the items in the permutation.

In the LOP, in contrast, the contribution of the item $\sigma(i)$ depends on the distribution of items in previous $\{1,\ldots,i-1\}$ and posterior $\{i+1,\ldots,n\}$ positions. However, the ordering of the items in each set does not influence the contribution of item $\sigma(i)$.

With respect to the PFSP, for both criteria, the total flow time and the makespan, the completion time of the jobs are the central objects. Note that the contribution of the item $\sigma(i)$ to f is the cumulative sum of the completion times of the previously scheduled jobs $(\sigma(1)\ldots\sigma(i-1))$, and the processing times of the job/item $\sigma(i)$. As a consequence, the contribution of $\sigma(i)$ depends on the ordering of the previous $i-1$ items in σ.

The QAP is probably, among the introduced problems, the one that generates the strongest dependencies among the items. If we check the fitness function associated to the problem, we see that all the pairwise combinations of items and their positions in the permutation are considered to calculate the fitness value of σ. As a result, the contribution of the item $\sigma(i)$ depends on the exact ordering of the rest of the items in the solution.

1.2 Estimation of Distribution Algorithms

In the last 30 years, metaheuristic algorithms [147] have attracted the attention of the artificial intelligence community, postulating as a solid alternative to classical operational research methods in the resolution of combinatorial optimization problems. These methods make few assumptions about the optimization problem to be solved, and thus, they can be applied to a wide variety of optimization problems, even in the cases where the function to be optimized has no mathematical expression or it can only be approximately calculated.

In this dissertation, we focus on a set of metaheuristics called Estimation of Distribution Algorithms (EDAs) [96, 106, 136, 138] that belong to the family of Evolutionary Algorithms (EAs). The main characteristic of EAs is the use of techniques inspired by the natural evolution of species. In nature, species change across time; individuals evolve, adapting to the characteristics of the environment. This evolution leads to individuals with better characteristics. The same idea is translated to the world of computation, where an individual represents a particular solution for the problem to be solved, a population comprises several individuals, and different operators such as crossover, mutation and selection techniques are used to make the individuals (solutions) evolve (improve). The most popular reference of these types of algorithms are Genetic Algorithms (GAs) [66]. As an extension to GAs, EDAs were introduced in the field of EAs in [128], although previous similar approaches can be found in [198].

Based on machine learning techniques, at each iteration, instead of using recombination and mutation operators, as GAs do, EDAs estimate a joint probability distribution associated with the set of most promising individuals, trying to explicitly express the interrelations between the different variables of the problem. Sampling the probabilistic model learnt in the previous generation, a new population of solutions is created. The algorithm stops iterating and returns the best solution found across the generations when a certain stopping criterion is met, such as a maximum number of generations/evaluations, homogeneous population, or lack of improvement in the solutions. Alg. 1.1 introduces a general pseudo-code of EDAs.

Based on this general framework, several EDA approaches have been developed in the last years [96, 106, 137, 138], where each approach learns a specific probabilistic model that characterizes the behavior of the EDA from

1 $D_0 \leftarrow$ Generate M individuals (the initial population) uniformly at random
2 **for** $t = 1, 2, ...$ until a stopping criterion is met
3 $D_{t-1}^{Se} \leftarrow$ Select $N \leq M$ individuals from D_{t-1} according to the selection method
4 $p_t(\boldsymbol{x}) = p(x|D_{t-1}^{Se}) \leftarrow$ Estimate the probability distribution from the set of selected individuals
5 $D_M \leftarrow$ Sample M individuals (the new population) from $p_t(\boldsymbol{x})$
6 $D_t \leftarrow$ Create the new population from D_{t-1} and D_M.
7 **end for**

Alg. 1.1: General pseudocode of Estimation of Distribution Algorithm.

the point of view of complexity and performance. Many works in the literature confirm the good performance of EDAs in the solution of problems from diverse fields: Protein Folding [153], Capacitated Vehicle Routing Problems [182], Calibration of Chemical Applications [125], Finding the Optimal Path in 3D Spaces [191], Software Testing [151], Chemotherapy Treatment Optimization for Cancer [24], Nuclear Reactor Fuel Management Parameter Optimization [83], Dynamic Pricing [121] or Molecular Docking [158] are some examples of real-world problems where EDA-based approaches were applied to find optimal solutions. Alternatively, due to the attention that EDAs received from the optimisation community, Santana et al. [152] developed a toolbox for EDAs, called MATEDA, for the mathematical computing environment Matlab. This package was conceived for solving single and multi-objective discrete and continuous optimisation problems using EDAs based on undirected and directed probabilistic graphical models.

Despite their success, little work has been done with regard to permutation-based combinatorial optimisation problems from the perspective of EDAs. In most cases, other metaheuristics have been preferred since these problems present a real challenge for EDAs, and particularly for the probability models implemented by these, which are not able to deal with the *mutual exclusivity constraints* associated with the permutations [76]. The probability distribution over permutations must guarantee that $P(\sigma : \sigma(i) = \sigma(j)) = 0$ for all $i \neq j$, which is not the case for the implemented probability models in the context of EDAs.

Due to the lack of suitable EDA proposals for solving permutation problems, and motivated by the growing interest seen with respect to probability models for modelling permutation spaces, this dissertation focuses on the introduction of these models to the framework of EDAs. In this sense, we study the implementation of distance-based exponential probability models, multi-stage probability models and Bradley-Terry type models, in the context of EDAs for solving permutation problems.

1.3 Outlook of the dissertation

This thesis is organised in three parts: contributions on EDAs, a comprehensive study of the LOP and the introduction of spectral landscape analysis for solving permutation problems.

The first part of the dissertation, divided into 5 chapters, introduces the contributions related to the use of estimation of distribution algorithms for solving permutation-based combinatorial optimisation problems and extensions thereof. Particularly, Chapter 2 presents a thorough review of the EDAs that have been applied to permutation-based problems. Classified into three groups, this review taxonomizes the existing EDAs as: adaptations of EDAs for integer problems, adaptations of EDAs for continuous problems, and permutation-oriented EDAs. The experiments conducted on the benchmark problems presented in Section 1.1 show that the most competitive EDAs are those designed originally to solve permutation problems. Additional tests, however, report that the evaluated approaches reach optimal solutions exceptionally, pointing out serious weaknesses of EDAs on permutation problems. In response to the lack of efficient proposals, some ideas on probabilistic modelling over permutation spaces are provided, that could be used as inspiration to design a new class of EDAs for permutation problems.

Experiments in Chapter 2 show that the most competitive EDAs implement models based on 1st and 2nd order marginals. As an extension to that idea, in Chapter 3 we propose an EDA that implements a model based on k-order marginal probabilities. Preliminary experiments on three instances of the TSP, QAP and PFSP show that learning models of higher order marginals may obtain better results. However, the computational effort required to obtain competitive results is so large that, in practice, this type of probability models are not feasible.

In Chapter 4 we introduce, for the first time, an EDA that implements a probabilistic model for permutation spaces called Generalized Mallows model. Defined by two parameters and a distance on permutations, this model explicitly assigns to each permutation in the search space, a probability exponentially inverse to its distance to a central permutation. A hybrid algorithm consisting of the new EDA and a variable neighbourhood search is proposed for solving the PFSP. Conducted experiments demonstrate that the new algorithm outperforms state-of-the-art algorithms in 152 instances out of 220. A thorough analysis of the results suggests that the successful performance is, principally, due to the ability of the Generalized Mallows EDA to discover promising regions in the search space.

In Chapter 5, a review of three common distances on permutations, Kendall's-τ, Cayley and Ulam, for the Mallows and the Generalized Mallows EDAs is carried out. With the aim of predicting the most suitable distance for each permutation problem in the benchmark (Section 1.1), this chapter analyses the relation between the distances above and some common neighbourhood systems. The conducted experiments reveal that the performance of

the Mallows and Generalized Mallows EDAs is strongly correlated with that of local search algorithms when using related distances and neighbourhoods. Moreover, fitness landscape analysis techniques report that the suitability of a distance to solve a problem is clearly characterised by the generation of smooth shape fitness landscapes.

In Chapter 6, a generalization of the Bradley-Terry probability model called Plackett-Luce is introduced in the framework of EDAs. Experiments on the LOP and the PFSP demonstrate that Plackett-Luce outperforms previously studied models when applied to the LOP. However, according to the results, the Generalized Mallows EDA is the most stable proposal when considering both problems.

The second part of the thesis approaches permutation problems from the perspective of landscape theory. In Chapter 7, the LOP is revisited. This work proposes a procedure to extract static information from an instance by studying its characteristics in relation to the landscape generated under the *insert* neighbourhood. In addition, it proposes a method to incorporate the obtained knowledge in order to improve the performance of local search-based algorithms. Particularly, the procedure presented identifies positions of the permutation at which the items of the problem cannot generate local optimal solutions (under the *insert* neighbourhood), and thus, neither global optimal solutions. This information is then used to propose a new neighbourhood that discards the insert operations which move items to positions where optimal solutions are not produced. Applied to two state-of-the-art local search-based algorithms, conducted experiments confirm that the algorithms using the new neighbourhood are generally preferred in terms of performed evaluations and execution time.

In Chapter 8, an analysis of the complexity of the LOP instances is carried out, and two metrics of complexity, called *insert ratio* and *ubiquity ratio*, are introduced. These metrics measure the difficulty of solving the LOP with local search type algorithms under the insert neighbourhood system. Experiments demonstrate that both metrics correlate with the estimated number of local optima associated to the instances.

The third part of the thesis introduces a final contribution that uses spectral landscape theory in order to decompose the landscape generated for the QAP under the *interchange* neighbourhood. Particularly, Chapter 9 proposes the multi-objectivization of the QAP, reformulating the problem as a 2-objective problem. In order to validate this novel approach, we compare the results obtained by the NSGA-II on the multi-objective approach of the QAP with the results of a simple GA on the classical approach. Performed experiments confirm that solving the multi-objective formulation clearly outperforms the classical approach, being especially efficient when solving real life-like instances.

Finally, Chapter 10 draws the general conclusions of the dissertation and points out possible future works.

Part I

Contributions on the design of EDAs for permutation problems

2

A review of EDAs for permutation problems

2.1 Introduction

As mentioned in Chapter 1, many works in the literature have proposed EDA approaches for permutation-based problems. However, as we will see throughout this chapter, most of them are adaptations of EDAs designed originally to solve integer or continuous domain problems. As a consequence, these proposals show several drawbacks when applied to permutation problems. In order to set the basis for a development of EDAs in permutation-based problems similar to that given in other domains, in this chapter we carry out a review of EDAs that have been applied to this class of problems. In addition, we provide some ideas on probabilistic modelling over permutation spaces that could inspire researchers to design efficient EDAs in the future.

The remainder of the chapter is organized as follows. In Section 2.2 we review the EDAs that have been applied to permutation-based optimization problems. In Section 2.3 we carry out a thorough experimental analysis of the EDAs on the benchmark problems presented in Chapter 1. In Section 2.4 we present several models for the estimation of probability distributions over permutation spaces giving some advice on their use in EDAs. Finally, Section 2.5 sums up the main conclusions and raises some ideas for future work.

2.2 EDAs for permutation-based optimization problems

We classify the existing EDAs into three groups. A first group is composed of those EDAs designed originally for solving integer domain problems and adapted to simulate permutation individuals at the sampling step. In a second group, we place those approaches designed for solving continuous optimization problems that have been modified to deal with permutations. Beyond adaptations of existing approaches, the literature includes a few works of EDAs that were designed to solve permutation problems, or general designs that, in

order to illustrate their usefulness, have been applied for the first time on this class of problems. We place these EDA approaches in the third group.

In the following sections we explain each group in detail and elaborate on the weak and strong points of each proposal.

2.2.1 Adaptations of EDAs for integer domain problems

One path that EDA researchers have followed to deal with permutations is the use of EDAs designed for integer domain problems [43, 49, 94, 95]. These algorithms learn, departing from a dataset of permutations, a probability distribution over a set $\Omega = \{1, 2, \ldots, n\}^n$. As a result, the sampling of these models may not provide permutation individuals but an individual in Ω.

In order to overcome this deficiency, the authors simulate permutation individuals by modifying the sampling step. The most common method to sample a probabilistic model in EDAs is the Probabilistic Logic Sampling algorithm [74]. In this sampling strategy, variables are instantiated following an ancestral order. To sample the i-th ordered variable, the previous $(i-1)$-th variables have to be instantiated. In order to obtain a permutation, the following changes have to be made to the sampling strategy. A permutation can be obtained if the i-th variable is not allowed to take the values instantiated by the previous variables. To do that, when the i-th variable has to be sampled, the probability of the previously sampled values is set to 0 and the probabilities of the rest of the values are normalized to sum 1. Although this procedure leads to permutations, we note that every time that we modify the probabilities to enable sampling permutation individuals, the information kept by the probabilistic model is denaturalized somehow in the sampled solutions.

Examples of adaptations of EDAs from this category are the Univariate Marginal Distribution Algorithm (UMDA) [95], the Mutual Information Maximization for Input Clustering (MIMIC) [15], the Dependency-Trees [139] or the Estimation of Bayesian Network Algorithm (EBNA) [15].

2.2.2 Adaptations of EDAs for continuous domain problems

Another way that the research community of EDAs has found to approach permutation-based problems is by means of EDAs designed for solving real-valued problems. These algorithms are based on a method that allows to decodify a real-valued vector as a permutation. Given a vector of n real values $(x(1), x(2), \ldots, x(n))$, a permutation individual can be obtained from it by ranking the positions using the values $x(i)$, $i = 1, \ldots, n$. Supposing we have the real vector:

$$(2.35, 3.42, 9.35, 0.32, 11.54, 10.42, 5.23, 4.2, 7.8),$$

the permutation obtained when decoding the vector, is $\sigma = 237198546$. Introduced first by Bean [12], this strategy is known as the *random keys* algorithm.

The main advantage of random keys is that they always provide feasible solutions, since each real-valued vector represents a permutation. However, as stated by Bosman and Thierens [21], random keys strategy is not effective and introduces large overheads since every time that an individual must be evaluated, an ordering algorithm has to be applied to get the corresponding permutation. In addition, part of the ineffectiveness of these strategy is related with the redundancy that the codification involves. One can easily notice that real-valued vectors with different values can lead to the same permutation. The real vector

$$(1.78, 3.90, 7.03, 1.24, 12.56, 9.87, 4.27, 4.10, 0.60)$$

would represent the same permutation $\sigma = 237198546$. In both cases, although the vector is different, the permutation is the same, and both solutions have the same fitness value. This creates plateaus of solutions in the corresponding continuous optimization problem that the EDA needs to solve.

This random keys strategy has been jointly used with different EDAs for real-valued problems [21, 149]. In [107] the Job Shop Scheduling Problem is approached with UMDA for the continuous domain (UMDA$_c$), MIMIC approach for the continuous domain (MIMIC$_c$) and Estimation of Gaussian Networks Algorithms (EGNAs).

Apart from the algorithms above, there exists a family of probability models for permutations closely related to the random keys strategy. Known as Thurstone order statistic models, these models, given $\{X_1, \ldots, X_n\}$ random variables with a continuous joint distribution $F(x_1, \ldots, x_n)$, can define a random permutation σ in such a way that $\sigma(i)$ is the position that X_i occupied between $X_1, \ldots, X_n$. In this way,

$$P(\sigma) = P(X_{\sigma^{-1}(1)} < X_{\sigma^{-1}(2)} < \ldots < X_{\sigma^{-1}(n)})$$

The computation of the maximum likelihood estimator parameters lead to very complex numerical problems, therefore, a number of papers have been devoted to this end [118, 119, 120, 190]. As regards the sampling process, it is as complex as sampling $F(x_1, \ldots, x_n)$ Examples of this type of models are F_i Gaussian or F_i Gumbel (Luce's model).

2.2.3 Permutation-oriented EDAs

In addition to the previously introduced EDAs, the research community has tried to go a step forward designing new algorithms that consider the real nature of permutations. In the following sections the outcome of that work is introduced and explained in detail. Although some of these algorithms could be considered in the previous two groups, we have introduced them in this group since they were devised originally to deal with permutations, or they have been applied to illustrate their usefulness for the first time over permutation-based problems.

2.2.3.1 IDEA Induced Chromosome Elements Exchanger

Bosman and Thierens [21] introduced a new algorithm called IDEA Induced Chromosome Elements Exchanger (ICE) to deal with permutation-based problems. They proposed a modification of the original IDEA approach introduced previously by the same authors in [20].

IDEA follows the general framework defined for EDAs for continuous problems, considering that the selected population follows a Gaussian distribution. A specific characteristic of IDEA is to factorize the Gaussian density function (pdf) as a product of marginal distributions. Particularly, the variables are partitioned into several subsets and a marginal pdf is estimated for each group. IDEA can be directly applied to permutation-based problems using the random keys representation. However, Bosman and Thierens [21] rejected this strategy since the joint use of random keys and real-value based EDAs, as previously reported by the authors, does not lead to very effective optimization algorithms. To overcome this problem, the ICE algorithm was proposed, in which probabilistic sampling of new solutions is replaced by a specialized crossover operator that takes into account the probabilistic model learnt by the algorithm. The crossover operator performs as follows: first, two parents (note that they are encoded as *random keys* strings) are randomly picked up from the selected samples. Second, based on the linkage information of the variables estimated in the probability model, blocks of variables are identified in the parents. Finally, the offspring is generated by choosing blocks from the parents (there exist multiple criteria) in a way that the linkage of the variables in each block is preserved. Note that, in ICE the probabilistic model is not explicitly used to sample new individuals, but only the information related with the partition of the variables.

2.2.3.2 Edge histogram models

In [176, 180] a new type of EDA for permutation-based problems called Edge Histogram Based Sampling Algorithm (EHBSA) is introduced. The algorithm estimates a probabilistic model that learns the adjacency of the items in the selected individuals at each generation. For an n-dimensional problem, the model is given by a matrix $\mathbf{E} = [e_{i,j}]_{n \times n}$ where $e_{i,j} = P(\sigma(k+1) = j | \sigma(k) = i)$ and $i, j \in \{1, 2, \ldots, n\}$ and $k \in \{1, 2, \ldots, n-1\}$. Each $e_{i,j}$ is added a ε value in order to control the pressure in sampling and avoid individuals with probability 0 or 1. ε is defined as

$$\varepsilon = \frac{2N}{n-1} B_{ratio},$$

where N denotes the size of the set of the selected individuals and B_{ratio} is a positive constant.

In order to sample the probabilistic model, the authors use an algorithm that samples the positions of the permutation in ascending order, starting

from position 1. Once position i-th has been sampled, position $(i+1)$-th is sampled using the row of matrix $\mathbf{E}$ corresponding to the item sampled at position i. This row is modified by setting to 0 those values which previously appeared, and normalizing the rest of the values to sum 1. A pseudocode for the sampling algorithm can be seen in Alg. 2.1.

```
1  Set the position counter k ⟵ 1.
2  Obtain first node σ(1) uniformly at random from {1, 2, ..., n}.
3  while (k < n)
4     Set to 0 previously sampled variables in row σ(k) of E.
5     Normalize non-sampled variables in row σ(k) of E.
6     Sample next variable σ(k + 1).
7     Update the position counter k ⟵ k + 1.
8  done
```

Alg. 2.1: Edge Histogram Based Sampling Algorithm.

In addition to this sampling, the authors propose another strategy which consists of the following steps. A parent individual is selected from the previous generation at random and $c > 2$ crossover points are selected uniformly at random, dividing the parent into c segments of variable length. Randomly selected $c-1$ segments of the parent are copied to the new individual and the remaining non-sampled segment in the individual is populated by sampling the probabilistic model with the previously introduced strategy. This sampling procedure leads to new individuals that differ from their parents on average in n/c items.

According to the authors, the introduced sampling strategies are called *sampling without template* EHBSA_{WO}, and *sampling with template* EHBSA_{WT} respectively.

In [179] the authors extend EHBSA to solve the PFSP, designing an asymmetrical edge histogram model. In [178] a revised EHBSA is proposed, referred to as enhanced EHBSA (eEHBSA). This approach presents a more flexible sampling procedure (cut-point selection) and modifies the way the new generation is created.

2.2.3.3 Node histogram models

In [181] the Node Histogram Based Sampling Algorithm (NHBSA) is introduced. The NHBSA builds a first order marginals matrix that represents the distribution of the items across the (absolute) positions in the permutation individuals. The probability model for an n-dimensional problem is given by a

matrix $\mathbf{Q} = [q_{i,j}]_{n \times n}$ where $q_{i,j} = P(\sigma(i) = j)$ and $i, j \in \{1, 2, \ldots, n\}$. Hence, $q_{i,j}$ represents the probability of the item j to be in the i-th position.

As in EHBSA, a ε is added to each $q_{i,j}$ in order to control the pressure in sampling, where N represents the size of the set of the selected individuals and B_{ratio} is a positive constant ratio set by the authors. ε is defined as

$$\varepsilon = \frac{N}{n} B_{ratio}$$

The design of the NHBSA focuses particularly on those problems where the main contribution of the items to the fitness function is given by their absolute position in the permutation.

As regards the sampling method, two strategies are proposed to simulate new individuals. The first proposal introduced by the authors uses a random sampling strategy. Similarly to EHBSA$_{WO}$, at each step, the sampling algorithm sets to 0 the probabilities in $\mathbf{Q}$ of the variables already sampled and normalizes the probabilities of the remaining variables to sum 1. A pseudocode for the sampling algorithm can be seen in Alg. 2.2.

The second sampling algorithm uses a parent individual from the previous generation to create the new individual. A random individual is picked-up from the previous generation and c random single positions are copied to the new individual. The remaining empty positions are filled by sampling the probabilistic model.

---

1 Generate a random permutation $pos[]$ of $1, \ldots, n$.
2 Generate a candidate item list $C = 1, \ldots, n$.
3 Set the position counter $k \longleftarrow 1$.
4 **while** $(k \leq n)$
5 Set to 0 those probabilities of variables already sampled variables in $pos[k]$ in $\mathbf{Q}$.
6 Normalize remaining probabilities to sum 1.
7 Sample item x.
8 Set $\sigma(pos[k]) \longleftarrow x$ and remove item x from C.
9 Update the position counter $k \longleftarrow k + 1$.
10 **done**

Alg. 2.2: Node Histogram Based Sampling Algorithm.

The authors denote as NHBSA$_{WO}$ and NHBSA$_{WT}$, the NHBSA that uses the *sampling without template* and *sampling with template* respectively.

In [177] several variations of the sampling methods for NHBSA are proposed. One proposes replacing the random sampling sequence with a sequential sampling sequence similar to EHBSA. Another variation considers choos-

ing randomly the number of nodes (variables) copied from the template string, instead of using a fixed number. Using probability density functions to determine the number c of crossover points is also introduced in [178].

2.2.3.4 Recursive EDA

Romero et al. [150] proposed a new class of EDAs called Recursive EDA (REDA). The REDA is an optimization strategy based on EDAs that consists of k optimization stages (see Fig. 2.1). In an initial stage, an EDA is applied to the problem and a solution is obtained. In a second stage, the variables of the problem are divided into two groups of equal size (when possible). Next, an EDA is executed over the variables that belong to the first group, while the variables of the second group remain fixed to the values given by the optimal solution in the previous stage. This process is repeated, fixing the variables in the second group and optimizing over the first group. This completes the second stage. The remaining stages follow the same procedure recursively. For instance, in the third stage, each group of the second stage is divided into two groups, and each group of variables is optimized separately. The algorithm stops when the number of variables in a group reaches a minimum threshold.

The motivation behind this proposal is to reduce the computational cost of learning the model (which in [124] is identified as the most expensive step of an EDA apart from the fitness function) by solving smaller problems at each stage.

Although every EDA approach could be used for optimization at each stage, due to the recursive nature of the strategy, the authors suggest using EDAs such as UMDA or MIMIC that permit keeping the computational cost feasible, since the EDA is executed repeatedly.

Even though this strategy is a general scheme and could be applied to any optimization problem, the authors proposed this algorithm for the optimization of the triangulation of Bayesian networks, and therefore we classify it as a specific EDA for permutation-based problems.

Regarding the codification scheme, REDAs use the previously introduced random keys encoding in the continuous approaches. However, for discrete domains, they refuse to use straightforward individual codifications as do the approaches introduced in Section 2.2.1. Instead of that, they propose a new codification that allows to learn probability distributions over permutations. In order to do that, they set a bijection between the numbers $\{1, \ldots, n!\}$ to the set of permutations of order n. This bijection is based on the decomposition in prime factors of $n!$ that is given such that $n! = p_1^{n_1} \cdot \ldots \cdot p_r^{n_r}$. An individual is then represented as a vector of length r, corresponding to the number of prime factors. Position i-th in the individual can take $n_i + 1$ values $\{0, 1, \ldots, n_i\}$, representing the possible exponent of the i-th prime factor. Therefore, given an individual, we obtain an integer, and from it the permutation is obtained. In Alg 2.3, the pseudocode of the transformation algorithm is introduced. In

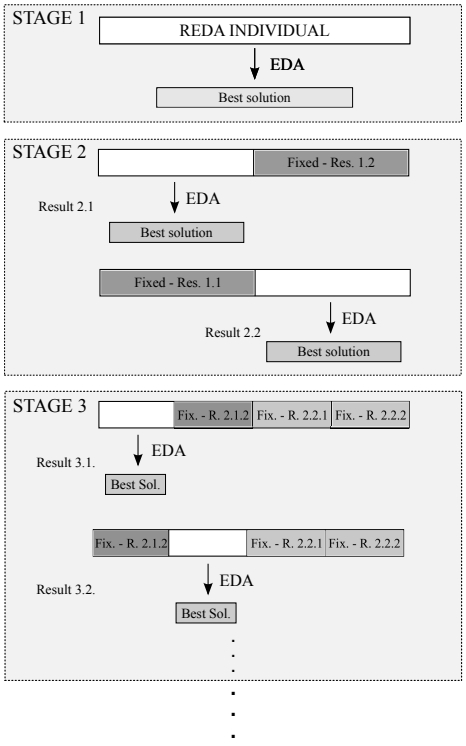


Fig. 2.1: Recursive EDA Strategy

addition, Table 2.1 shows an example for this procedure, supposing a problem of size 4, and given the individual (2,2,1).

---

1 **input:** *individual*
2 Initialize a vector $l = (1, 2, \ldots, n)$.
3 **for** $i = 1$ to $n - 1$
4 $\quad \sigma(i) = l(individual(i))$
5 $\quad$ update l, removing the $individual(i)$ position
6 **end for**
7 $\sigma(n) = l(1)$
8 **output:** σ

Alg. 2.3: Pseudocode of the algorithm that generates a permutation from an individual.

Table 2.1: Example of the algorithm that generates a permutation from an individual.

Line Alg. 2.3	*individual*	Vector l	σ	Updated vector l
2	(2,2,1)	(1,2,3,4)		-
For, $i = 1$	(**2**,2,1)	(1,**2**,3,4)	2	(1,3,4)
For, $i = 2$	(2,**2**,1)	(1,**3**,4)	23	(1,4)
For, $i = 3$	(2,2,**1**)	(**1**,4)	231	(4)
7	(2,2,1)	(**4**)	2314	()

Although this codification allows to learn a probability distribution over permutation spaces, the decodification process denaturalizes the relation between the variables and permutations.

2.2.4 Hybrid EDAs

In addition to the previous algorithms, several hybrid EDAs have also been proposed for permutation-based problems. These algorithms generally combine standard EDA approaches with other techniques such as local search [81, 193, 194] or particle swarm optimization [104]. In [193, 194] an operator called Guided Mutation is introduced, which combines a conventional mutation operator with a probabilistic model learnt at each step. In [36] authors propose a hybrid EDA for solving single machine scheduling problems that combines classic univariate and bivariate probabilistic models with crossover and mutation genetic operators.

Due to the difficulty of measuring the contribution of the EDA part to the performance of the hybrid approaches, we decided not to include these algorithms in the experiments.

2.3 Experiments

In order to compare the performance of the most representative EDAs reviewed, in this section we introduce an extensive experimental study.

2.3.1 Experimental setup

We considered the benchmark problems presented in Chapter 1. Particularly, we selected the following sets of 24 instances for each problem type:

- TSP: *bays29*, *berlin52*, *burma14*, *ch130*, *dantzig42*, *eil51*, *eil76*, *eil101*, *fri26*, *gr17*, *gr24*, *gr48*, *gr96*, *gr137*, *hk48*, *pr76*, *pr107*, *pr124*, *pr136*, *rat99*, *st70*, *swiss42*, *ulysses16* and *ulysses22*.
- QAP: *bur26a*, *bur26b*, *bur26c*, *bur26d*, *nug17*, *nug18*, *nug20*, *nug21*, *tai10a*, *tai10b*, *tai12a*, *tai12b*, *tai15a*, *tai15b*, *tai20a*, *tai20b*, *tai25a*, *tai25b*, *tai30a*, *tai30b*, *tai35a*, *tai35b*, *tai40a* and *tai40b*.

- LOP: *t75i11xx*, *t65f11xx*, *t65b11xx*, *t65d11xx*, *t65i11xx*, *t65l11xx*, *t65n11xx*, *t65w11xx*, *t69r11xx*, *t70b11xx*, *t70d11xx*, *t70d11xxb*, *be75eec*, *be75np*, *be75oi*, *be75tot*, *tiw56n54*, *tiw56n58*, *tiw56n62*, *tiw56n66*, *tiw56n67*, *stabu70*, *stabu74*, *stabu75* and *usa70*.
- PFSP: *tai20×5, tai20×10, tai50×10* and *tai100×20* (The first six instances from each type).

The TSP instances were obtained from TSPLIB [148]. As regards the LOP instances, LOLIB benchmark was addressed[1]. Finally, QAP and PFSP instances were downloaded from the Eric Taillard's benchmark of instances [168].

Regarding the set of the selected algorithms, the choice has been made according to the classification of EDAs presented in Section 2.2. From the set of EDAs designed originally for solving integer domain problems, we have chosen UMDA [95], MIMIC [15], EBNA$_{BIC}$ [15] and TREE [139]. From the group of EDAs forthe continuous domain problems, we have chosen UMDA$_c$ and EGNA$_{ee}$ [107]. In addition, all the EDAs specifically designed for solving permutation optimization problems have been selected for the comparison: IDEA-ICE, EHBSA$_{WO}$, EHBSA$_{WT}$, NHBSA$_{WO}$, NHBSA$_{WT}$ and REDA. Alternatively, for comparison purposes, we have included a very well known GA, the Ordering Messy Genetic Algorithm (OmeGA) [90] designed for permutation problems.

As previously mentioned, there are hybrid EDAs that have been applied to several permutation problems. In these algorithms, it is quite complex to measure what the contribution of the probabilistic model to the optimization process is. Due to this, we have limited the experiments to"pure" EDAs since we aim to analyze the performance of the different probabilistic models when solving permutation problems.

Each *algorithm-instance* pair is run 10 times. We have set the following execution parameters for all EDAs (n denotes the size of the instance):

- Population size: $10n$.
- Selection size: $10n/2$.
- Offspring size: $10n - 1$.
- Selection type: truncation selection method.
- Elitism selection: the best individual of the previous generation is guaranteed to survive.
- A maximum number of $100n$ generations are performed.

Regarding specific-EDA parameters, the values suggested by their respective authors have been used. Romero et al. [150] suggest executing REDA with fast execution EDAs since they will be run repeatedly, thus we use UMDA and MIMIC, as the authors do in their experiments. On the other hand, Tsutsui [181] suggests setting the B_{ratio} constant to 0.0002 for EHBSA and NHBSA.

[1] LOLIB home page: `http://www.iwr.uni-heidelberg.de/groups/comopt/software/LOLIB/`

2.3.2 Results

Table 2.2 shows the average error and standard deviation for each problem type[2]. This average error is calculated as the normalized difference between the best fitness value obtained by the algorithm and the best known solution. Note that each entry in the table is the average of 240 values (24 instances $\times$ 10 runs). The lower the values, the better the performance. Looking at these results, it can be seen that Tsutsui's EHBSA$_{WT}$ and NHBSA$_{WT}$ are, by far, the algorithms that provide on average the best results for every problem type. These results show the high influence of the templates (WT approaches) when sampling new individuals. At the opposite end, the results show that the EDAs designed for solving continuos domain problems, the REDA approaches and OmeGA are, without doubt, the algorithms that perform the worst.

The results confirm the insights provided for each problem in relation with the contribution of the items in the permutation to the fitness function (see Section 1.1.5). For instance, in TSP the relevant information for the calculation of the fitness function is given by the relative ordering of the items. The results in Table 2.2 show that, for the TSP, the algorithm that learns the adjacency of the items is the most competitive proposal (EHBSA$_{WT}$). On the other hand, NHBSA$_{WT}$ performs the best for the QAP, as the probabilistic model is focused on estimating the probability distribution of the items in the absolute positions of the permutation. In the problems for which the contribution of the item is not so explicit, i.e. LOP and PFSP, NHBSA$_{WT}$ and EHBSA$_{WT}$ have similar behaviors.

In order to carry out a statistical analysis of the results obtained in the experiments, and following the suggestions given in [62], we decided to use non-parametric tests. The authors state that, for multiple-problem analysis –as is our case–, due to the dissimilarities in the results and the small size of the sample to be analysed, a parametric test may result in erroneous conclusions. Taking into account the diversity of characteristics of the considered problems, a priori we presume different performances among the compared EDAs, and therefore, the statistical comparison has been performed individually for each problem type.

The statistical analysis will be conducted in two steps. First, we will check if significant differences exist among the results obtained. For this purpose, the Friedman's test will be used. This test ranks the algorithms for each problem, providing also an average rank value for each of the algorithms. These ranks can be consulted in Table 2.3.

The p-values resulting from applying Friedman's test are lower than 0.0001, which is below the level of significance considered ($\alpha = 0.05$). This means that there exist significant differences among the observed results. Once the rejection of the null hypothesis has been proved, a post-hoc method will be used

[2] Average and standard deviation results of the fitness value obtained for all the instances tested (4 problem types $\times$ 24 instances) by the 14 algorithms can be found in `http://www.sc.ehu.es/ccwbayes/members/jceberio/home/index.html`.

Table 2.2: Average error and standard deviation for each algorithm in each problem type. Results in bold indicate the best average result found.

EDA		TSP	QAP	LOP	PFSP
UMDA	avg.	0.5077	0.0298	0.1511	0.0538
	dev.	0.3315	0.0153	0.0354	0.0308
MIMIC	avg.	0.6762	0.0390	0.1495	0.0351
	dev.	0.4371	0.0211	0.0351	0.0117
$EBNA_{BIC}$	avg.	0.5051	0.0310	0.1508	0.0545
	dev.	0.3438	0.0153	0.0358	0.0326
TREE	avg.	1.2554	0.0526	0.1761	0.0601
	dev.	0.8637	0.0318	0.0369	0.0223
$UMDA_c$	avg.	1.2792	0.2118	0.3303	0.1535
	dev.	0.9408	0.1420	0.0384	0.0322
$EGNA_{ee}$	avg.	1.1830	0.1655	0.3118	0.1424
	dev.	0.8886	0.1006	0.0481	0.0335
IDEA-ICE	avg.	1.2090	0.0801	0.1743	0.0734
	dev.	0.7610	0.0320	0.0322	0.0253
$EHBSA_{WT}$	avg.	**0.0037**	0.0256	0.1371	**0.0276**
	dev.	**0.0059**	0.0189	0.0328	**0.0232**
$EHBSA_{WO}$	avg.	0.1251	0.0653	0.2239	0.0626
	dev.	0.1544	0.0395	0.0400	0.0453
$NHBSA_{WT}$	avg.	1.0680	**0.0112**	**0.1366**	0.0277
	dev.	0.8659	**0.0130**	**0.0328**	0.0215
$NHBSA_{WO}$	avg.	0.3385	0.0222	0.1375	0.0326
	dev.	0.2443	0.0144	0.0326	0.0226
$REDA_{UMDA}$	avg.	2.0550	0.1426	0.2131	0.0986
	dev.	1.1909	0.0811	0.0467	0.0541
$REDA_{MIMIC}$	avg.	1.7410	0.1727	0.2794	0.1242
	dev.	1.3057	0.0963	0.0750	0.0443
OmeGA	avg.	1.2860	0.1347	0.3336	0.1281
	dev.	0.8513	0.0684	0.0750	0.0734

to carry out all pairwise comparisons. Particularly, Shaffer's static procedure will be used, as suggested for such cases in [61]. Again, the significance level has been fixed to $\alpha = 0.05$. Results obtained from this procedure are represented in Figures 2.2, 2.3, 2.4 and 2.5 by means of critical difference diagrams. These diagrams draw the ranking of the algorithms and link with a horizontal line those groups of algorithms for which no significant differences were found (p-values higher than $\alpha = 0.05$).

The statistical analysis confirms the good performance of NHBSA and EHBSA, and particularly those algorithms that use the template strategy. Even though UMDA, MIMIC and $EBNA_{BIC}$ are not designed specifically to deal with permutation problems, they are the closest to Tsutsui's algorithms, and show an acceptable performance.

Surprisingly, the results achieved by $EBNA_{BIC}$ and TREE do not outperform those achieved by UMDA. As stated in the literature, $EBNA_{BIC}$ and

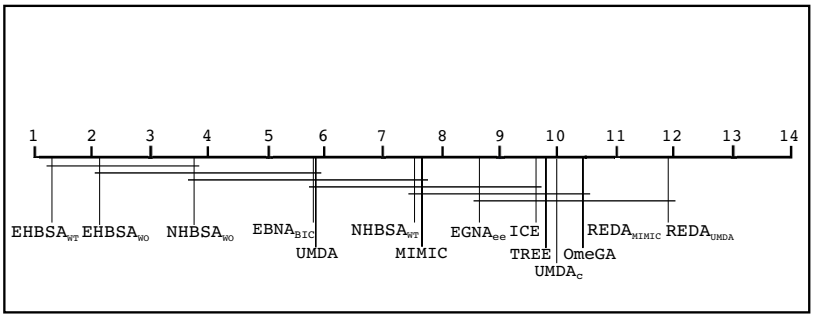


Fig. 2.2: Critical difference diagram of the results on the TSP.

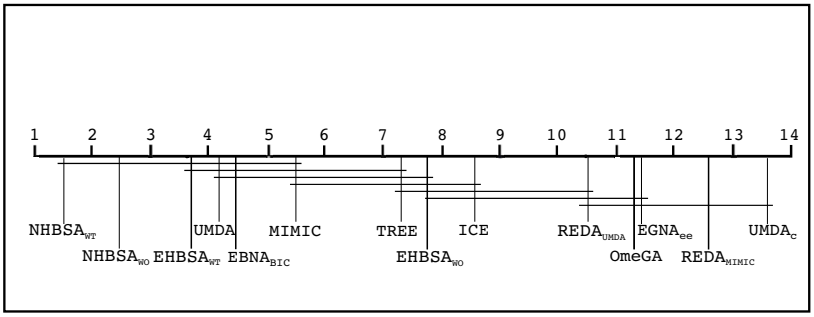


Fig. 2.3: Critical difference diagram of the results on the QAP.

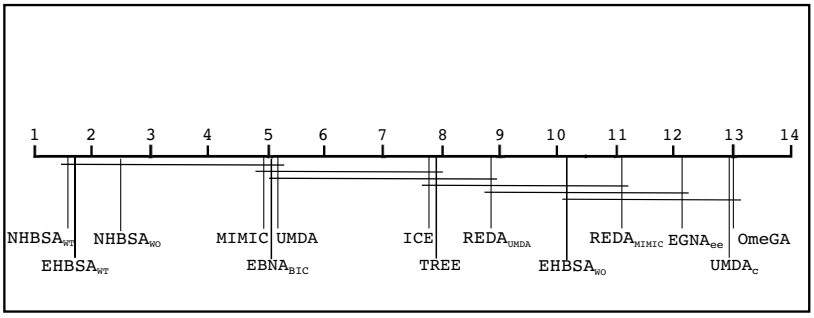


Fig. 2.4: Critical difference diagram of the results on the LOP.

Table 2.3: Average rankings of the algorithms. The lower the rank, the better the performance.

EDA	TSP	QAP	LOP	PFSP
UMDA	5.87	4.16	5.12	6.04
MIMIC	7.70	5.41	4.91	4.37
EBNA$_{BIC}$	5.83	4.41	5.04	5.89
TREE	9.87	7.25	7.91	7.12
UMDA$_c$	9.97	13.66	12.95	13.12
EGNA$_{ee}$	8.64	11.41	12.12	12.08
IDEA-ICE	9.70	8.66	7.79	8.49
EHBSA$_{WT}$	1.27	3.79	1.68	1.91
EHBSA$_{WO}$	2.12	7.83	10.16	6.54
NHBSA$_{WT}$	7.52	1.5	1.56	2.04
NHBSA$_{WO}$	3.79	2.45	2.75	3.20
REDA$_{UMDA}$	11.91	10.54	8.87	10.52
REDA$_{MIMIC}$	10.37	12.62	11.08	11.58
OmeGA	10.37	11.25	13.00	12.04

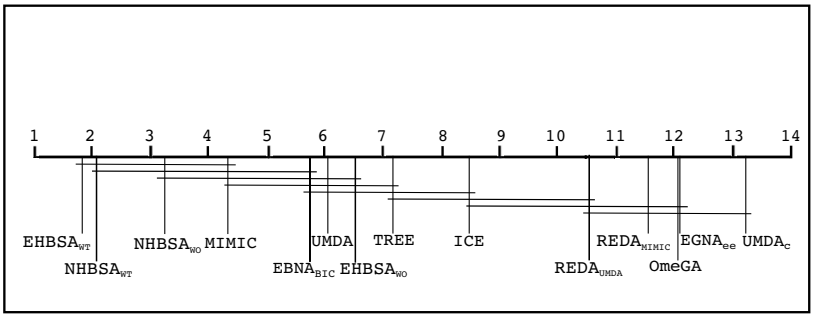


Fig. 2.5: Critical difference diagram of the results on the PFSP.

TREE algorithms are supposed to be more powerful than univariate algorithms, such as UMDA, since the first two learn (in)dependencies between variables, and UMDA assumes independence between them instead. In order to understand this behavior, we studied the probabilistic models learnt by EBNA$_{BIC}$ at each step. We realized that the learnt structure was an empty structure at all the times. And thus, the behavior of EBNA$_{BIC}$ results similar to that of UMDA. The reason why the learning algorithm does not learn any structure is as follows: when we analyze the performance of these algorithms, it is important to note that they work in the probability space of size n^n (being n the size of the problem). This means that when we introduce an arc $X_i \rightarrow X_j$ in the probabilistic model, the number of parameters for codifying the local probability distribution of X_i is multiplied by n. EBNA$_{BIC}$ includes the arcs that improve the BIC score the most. This score is based on the maximum likelihood between nodes and a penalty term related to the complexity

of the structure. Taking into account the population size used in the experiments, when we try to add an arc, the increase in the likelihood is always smaller than the increase in the complexity, and therefore no arc is added. In the case of TREE, due to its design, the structure learning algorithm is forced to add those arcs that have the highest mutual information. However, as the population size does not provide enough information, the learnt tree turns out to be an over-fitted model.

IDEA-ICE shows a moderate efficiency, while REDA, OmeGA and the classical approaches for continuous domains, EGNA_{ee} and UMDA_c obtained the worst results (this might be due to the highly redundant encoding domain, as stated in several works).

In addition to these results, we consider it interesting to provide supplementary information about the number of times that the algorithms are able to get the best known solutions, and the average number of iterations (generations) needed by them. Results in Table 2.4 show again that the highest success rates belong to EHBSA and NHBSA, taking note of the high influence of employing the template strategy. Surprisingly, there is not any EDA able to achieve best known solutions for the LOP instances. In general, such low rates demonstrate the weakness of the compared EDAs.

In order to analyze the behavior of EDAs in relation to the iterations needed by each algorithm to obtain its best solution, Table 2.5 introduces the average and standard deviation of the number of generations needed. Note that the number of iterations in crossover-based algorithms such as IDEA-ICE and OmeGA is dramatically low compared to the rest of EDAs. Another remarkable fact is the high deviation in the number of iterations.

Table 2.4: Success rate of the algorithms achieving best known solutions.

EDA	TSP	QAP	LOP	PFSP
UMDA	1.6	0.8	0.0	0.0
MIMIC	0.4	2.1	0.0	0.0
EBNA_{BIC}	2.1	0.8	0.0	0.8
TREE	0.4	1.2	0.0	0.0
UMDA_c	4.2	0.0	0.0	0.0
EGNA_{ee}	3.7	0.0	0.0	0.0
IDEA-ICE	0.0	0.0	0.0	0.0
EHBSA_{WT}	74.2	9.6	0.0	11.2
EHBSA_{WO}	40.0	5.8	0.0	0.0
NHBSA_{WT}	10.4	31.7	0.0	5.4
NHBSA_{WO}	2.9	7.9	0.0	2.5
REDA_{UMDA}	0.4	0.0	0.0	0.0
REDA_{MIMIC}	1.2	0.0	0.0	0.0
OmeGA	0.0	0.0	0.0	0.0

As a general conclusion, it must be highlighted that the algorithms that obtain the best results are two of the EDAs designed specially to deal with

Table 2.5: Average and standard deviation of the iterations required to find the best solution. REDA algorithms are not included in this analysis due to the recursive strategy that follows the EDA.

EDA		TSP	QAP	LOP	PFSP
UMDA	avg.	5362.78	698.85	4472.73	2973.22
	dev.	3965.94	506.18	884.72	2536.14
MIMIC	avg.	1993.53	531.11	2430.76	775.17
	dev.	1328.35	365.32	521.73	689.54
EBNA$_{BIC}$	avg.	5386.81	717.50	4509.50	3105.74
	dev.	4123.52	529.23	827.77	2626.11
TREE	avg.	5896.24	1487.35	4613.75	2959.04
	dev.	3522.10	840.35	703.43	2246.44
UMDA$_c$	avg.	2440.82	591.67	563.35	447.93
	dev.	2077.21	485.06	722.65	675.64
EGNA$_{ee}$	avg.	3353.38	1123.40	1036.17	1848.13
	dev.	2163.48	499.00	1036.17	1523.86
IDEA-ICE	avg.	258.35	39.32	144.25	289.24
	dev.	224.63	21.21	37.86	341.13
EHBSA$_{WT}$	avg.	3526.13	1702.01	4426.07	3972.13
	dev.	3501.54	817.43	1040.97	2772.07
EHBSA$_{WO}$	avg.	4298.46	1571.58	3724.33	3432.13
	dev.	3719.40	588.09	486.51	2146.82
NHBSA$_{WT}$	avg.	6266.87	1645.08	4254.60	3882.62
	dev.	3592.48	903.79	1099.40	2719.95
NHBSA$_{WO}$	avg.	3979.47	536.72	2571.42	2938.68
	dev.	3610.47	390.46	1020.63	2767.95
OmeGA	avg.	31.70	21.66	32.38	29.58
	dev.	5.17	12.68	1.92	10.26

permutations. Moreover, NHBSA and EHBSA implement models based on 1st and 2nd order marginal probabilities, which theoretically are too simple to efficiently represent the underlying probability distribution. These results should encourage the research community to follow this direction, trying to design more effective probability models over the space of permutations. In the next section, we discuss some ideas that could be useful for this purpose.

2.4 Discussion

As mentioned in the previous sections, in order to deal with permutation-based problems, the proposed EDA approaches are (i) adaptations of algorithms designed for solving integer domain problems, (ii) transforming a permutation problem into a continuous optimization problem and then using EDAs designed for the continuous domain or (iii) ad hoc approaches for permutation domains. Contrary to integer domain problems, where the community has used many mechanisms from machine learning and statistics– graphical models, kernels. . . – to codify probability models, this has not been the case for

permutation problems. In this section, we give a brief review of the most common probabilistic models to deal with permutation spaces. We also point out some ideas on the use of these models in EDAs, focusing briefly on the procedures to learn and sample the probability models.

2.4.1 Models based on marginals

When working with samples of permutations, the trivial approach consists of maintaining the information relative to the first order marginals, which expresses the probability of item i being at position j. This information can be stored in $O(n^2)$ space by using an $n \times n$ matrix. A natural extension consists of storing higher order marginals. Such marginals correspond to the probability of a specific set of items $(i_1, ..., i_k)$ being at specific positions $(j_1, ..., j_k)$. One may also be interested in maintaining the probability of an item being at the position right after another item without specifying a particular position. In fact, this is the kind of information used in the edge histogram model, [176, 180], while the node histogram model, [181], maintains the first order marginals.

This representation is not only compact, but it is also easy to learn. By using the first order marginals, statistics such as the mode can be computed. However, when it comes to sampling, a necessary step in EDAs, further information about the distribution is required. In [181] a distribution with the given marginals is sampled. However, the actual distribution is unknown. There does not seem to be a closed form for it and it is not clear which are the properties of such a distribution. Actually, there can be infinitely many probability distributions that have a given first order marginal probability matrix. Therefore, among all these distributions how can one select the "correct" one? A common approach in statistics and machine learning is to consider the maximum entropy distribution, i.e. the distribution that, by having those marginal probabilities, has the highest uncertainty. This is the procedure followed by [2] which showed that such a distribution happens to have the simple expression given by

$$P(\sigma) = exp(\sum_{i=1}^{n} \boldsymbol{Y}_{i,\sigma(i)} - 1).$$

where $\boldsymbol{Y} \in \mathbb{R}^{nxn}$. Unfortunately, it is also shown in [2] that obtaining the matrix $\boldsymbol{Y}$ is #P-hard. Nevertheless, they also give an approximation algorithm which runs in polynomial time for computing the parameter $\boldsymbol{Y}$.

2.4.2 The Plackett-Luce model

The Plackett-Luce distribution takes its name from the combination of the independent work carried out by Plackett [142] and Luce [109]. Luce's model describes a sequential permutation generator method in which the items are sampled from the first to the last position (i.e., from the most to the least

preferred item). The parameter space consists of n positive weights $(w_1, ..., w_n)$ that sum 1. These probabilities are used to sample the first position of the rank, with $P(\sigma(1) = j) = w_j$. The following positions, $\{2, ..., n\}$, are sampled without replacement until a complete solution is obtained. Note that, in order to sample position i of a permutation by using Luce's model, the probability of selecting item j_1 over j_2 does not depend on the weights of the rest of the items in the set.

The above model induces a distribution over all possible permutations. It was first used by Plackett and can be written as follows:

$$P(\sigma) = \prod_{i=1}^{n-1} \frac{w_{\sigma(i)}}{\sum_{j=i}^{n} w_{\sigma(j)}}$$

Due to the Markovian nature of the model, it is not easy to make inference over sets of items such as $P(\sigma(n) = i)$. Regarding the learning process of the n parameters of a Plackett-Luce distribution, one can find in the literature methods based on maximum likelihood estimation [77] and Power EP (expectation propagation) [70]. Once the parameters of the distribution are known, the sampling procedure consists of following Luce's model.

2.4.3 The Mallows model

The Mallows model [110] is a distance-based exponential probability model over permutation spaces. Given a distance D over permutations, it can be defined by two parameters: the central permutation σ_0, and the spread parameter θ. Eq. 2.1 shows the explicit form of the probability distribution over the space of permutations:

$$P(\sigma) = \frac{1}{\psi(\theta)} e^{-\theta D(\sigma, \sigma_0)} \tag{2.1}$$

where $\psi(\theta)$ is a normalization constant. When $\theta > 0$, the central permutation σ_0 is the one with the highest probability value and the probability of the other $n! - 1$ permutations exponentially decreases with the distance to the central permutation (and the spread parameter θ). Because of these properties, the Mallows distribution is considered analogous to the Gaussian distribution on the space of permutations.

Although the Mallows model is commonly coupled with the Kendall's-τ distance, it has been also used with other distances such as Cayley, Ulam, Hamming or Spearman [52, 56].

Among its many extensions, the *Generalized Mallows* model [56] has received an special attention from the community. Under this model, the distance is decomposed into $n - 1$ terms, $S_j(\sigma, \sigma_0)$, in such a way that, instead of using a single spread parameter, the GM model makes use of $n - 1$ spread parameters, $\boldsymbol{\theta} = (\theta_1, ..., \theta_{n-1})$, each of them affecting a particular component

of the solution. The probability distribution over each distinct permutation is calculated as follows:

$$P(\sigma) = \frac{1}{\psi(\boldsymbol{\theta})} e^{-\sum_{j=1}^{n-1} \theta_j S_j(\sigma,\sigma_0)}$$

Note that when the $n-1$ parameters θ_j are constrained to be equal, the Generalized Mallows model reduces to the Mallows model.

The typical way to learn the parameters of the distribution of a given sample of permutations is to maximize the likelihood of these parameters. However, for both models, Mallows and Generalized Mallows, the problem of finding the maximum likelihood estimators is *NP-hard*. Therefore, the learning step is commonly carried out in two steps: first, the central permutation is calculated with exact [46] or heuristic [3, 112] methods, and second, given σ_0, the MLE $\boldsymbol{\theta}$ estimators are calculated.

As regards the sampling process, it depends on the distance used which has its own procedure.

Although the Mallows and Generalized Mallows models define explicitly probability distributions in the space of permutations, their unimodal nature makes the representation of distributions of multimodal optimization problems impossible.

2.4.4 Non-parametric models

As we have already stated, the unimodality of the Mallows and Generalized Mallows models can be a drawback to deal with multimodal optimization problems. According to Lebanon et al. [99], the unimodality and symmetry of the Mallows model make it a good choice to be used as a kernel. In fact, authors proposed a multimodal non-parametric model placing Mallows kernels on the top of the elements of a given sample of permutations. Given a sample of permutations $\{\sigma_1, \ldots, \sigma_m\}$, the non-parametric estimator of such a distribution is given by the following equation

$$p(\sigma) = \frac{1}{m\psi(c)} \sum_{i=1}^{m} e^{-cD(\sigma,\sigma_i)}$$

where m denotes the number of kernels in the model and c is a spread parameter set manually.

2.5 Conclusions and future work

In this chapter we have reviewed the existing EDA approaches for solving permutation-based optimization problems. Particularly, we classified them in three groups: (i) adaptations of algorithms designed for solving integer domain

problems, (ii) transforming a permutation-based problem into a continuous optimization problem and then using EDAs designed for continuous domains and (iii) permutation-oriented EDAs. The experimental analysis carried out showed that the best results are given by those EDAs that implement ad hoc designs. On the contrary, continuous and REDA approaches are those which perform the worst. The experimental analysis also stated that the EDAs for integer domain problems yield good solutions. But, those algorithms that find the best solutions, EHBSA and NHBSA, are the only EDAs that were originally designed for solving permutation problems. This fact suggests that the future work that the research community of EDAs should follow is the use of the probabilistic models over permutation spaces. Following this idea we have introduced several models that could be used with EDAs in order to solve permutation-based problems.

3

A study of models based on *k*-order marginal probabilities

3.1 Introduction

In the previous chapter, we have seen that two permutation-oriented EDAs, EHBSA [176, 180] and NHBSA [181] are the most competitive algorithms among those studied. Basically, the authors consider models based on 1st and 2nd order marginal probabilities. As an extension to that idea, here we propose a new EDA that implements a probabilistic model based on k-order marginal probabilities, able to determine the relations of the variables assuming k interactions for each variable ($k \geq 1$). These models should be able to better identify the relations (dependencies) between the variables of the problem as k increases.

The remainder of the chapter is as follows: in the next section, we present our proposal, the k-order marginals based EDA. In Section 3.3, some experiments on three permutation problems are carried out. Section 3.4 introduces a discussion on marginal based EDA approaches for permutation problems. Finally, conclusions are drawn in Section 3.5.

3.2 *k*-order marginals EDA

Further developing the idea introduced by Tsutsui et al. [176, 180, 181], we present a k-order marginals based EDA which assumes k-length interactions between all variables in the problem.

Our model is given by a matrix $\mathbf{M}_k = [m_{i,j}]$, where each entry of the matrix is given by each of the possible k-order marginal probabilities:

$$P(\sigma(i_1) = j_1, \ldots, \sigma(i_k) = j_k)$$

In Table 3.1 we show an example of $n = 4$ (problem size) and $k = 2$ order marginal probabilities matrix. The rows of the matrix represent the combinations of positions of the permutation and the columns represent the

combinations of items. For instance, the entry $m_{2,4}$ of the matrix gives the joint probability that item 2 is in position 1 and item 1 is in position 3, i.e., $P(\sigma(1) = 2, \sigma(3) = 1)$.

Table 3.1: Examples of a $\mathbf{M}_2$ marginals matrix for a problem of size 4.

		Item Combinations											
		(1,2)	(1,3)	(1,4)	(2,1)	(2,3)	(2,4)	(3,1)	(3,2)	(3,4)	(4,1)	(4,2)	(4,3)
Positions	(1,2)	0.20	0.05	0.10	0.05	0.10	0.05	0.05	0.10	0.05	0.05	0.10	0.10
	(1,3)	0.05	0.20	0.10	0.10	0.05	0.05	0.05	0.05	0.10	0.10	0.10	0.05
	(1,4)	0.10	0.10	0.15	0.05	0.05	0.10	0.10	0.05	0.05	0.10	0.05	0.10
	(2,3)	0.05	0.05	0.05	0.10	0.15	0.15	0.10	0.10	0.05	0.05	0.05	0.10
	(2,4)	0.05	0.05	0.05	0.10	0.15	0.15	0.10	0.05	0.10	0.05	0.10	0.05
	(3,4)	0.05	0.10	0.10	0.10	0.05	0.05	0.05	0.10	0.15	0.10	0.05	0.10

It is important to take into account the complexity in terms of memory and computation requirements of these models. For the matrix $\mathbf{M}_k$ the number of rows is given by the number of k size subsets from $\{1, \ldots, n\}$:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{3.1}$$

Regarding the number of columns, this is given by the number of the ordered (to avoid redundances) k size subsets from $\{1, \ldots, n\}$:

$$\frac{n!}{(n-k)!} \tag{3.2}$$

Therefore, we can say that the complexity of the algorithm in terms of memory is bounded by:

$$O(n^{2k}/k!) \tag{3.3}$$

Table 3.2 shows the number of entries of the matrix $\mathbf{M}_k$ for different n and k combinations. As can be seen, this approach can be used for small values of k and/or n, because otherwise the memory requirements make it unaffordable. However, this is in agreement with algorithms based on k-order marginals in binary spaces where, while the memory requirements are lower, the number of operations is in the same order of complexity [78].

Table 3.2: Memory complexity of the algorithm according to n and k

$n \backslash k$	1	2	3	4
10	100	4050	86400	1058400
20	400	72200	7797600	$5.63\text{x}10^{8}$
50	2500	3001250	$2.30\text{x}10^{9}$	$1.27\text{x}10^{12}$
100	10000	49005000	$1.56\text{x}10^{11}$	$3.69\text{x}10^{14}$

In the following sections we introduce the learning algorithm of the $\mathbf{M}_k$ matrix and the sampling technique used to generate new individuals.

3.2.1 Learning k-order marginals model

Each of the entries of the matrix $\mathbf{M}_k$ is calculated using maximum likelihood estimation from the set of selected individuals. Basically, it means that the probability $P(\sigma(i_1) = j_1, \ldots, \sigma(i_k) = j_k)$ is calculated from the number of times that the configuration $(\sigma(i_1) = j_1, \ldots, \sigma(i_k) = j_k)$ appears in the set of selected individuals. In addition, in our experiments we have used Laplace correction to avoid probabilities equal to 0, or 1.

3.2.2 Sampling new individuals

In order to sample a new individual from the k order marginals matrix, the procedure is carried out as follows. Firstly, a position is selected uniformly at random. Secondly, we sample an item for that position from the 1st order marginals matrix $\mathbf{M}_1$, which is calculated as detailed in the previous section when $k = 1$. Once the first item has been placed, we repeat the process by choosing another position, and sampling the second item from the 2nd order marginals matrix $\mathbf{M}_2$ taking into account the item sampled in the previous step. Note that the sampling process must be carried out meeting the mutual exclusivity constraints associated with permutations, and therefore, the sampling procedure needs to be adapted in order to not to sample items already set in previous steps. The items are sampled until k items are set in the individual. Beyond this point, the remaining $n - k$ items are sampled directly from the k-order marginals matrix $\mathbf{M}_k$.

In order to illustrate the sampling process, in Table 3.3 we introduce an example of $n = 4$ and $k = 2$. Alternatively, the pseudocode of the sampling algorithm is described in Alg. 3.1. In order to sample a new individual, we start by sampling $\mathbf{M}_1$. We choose a position uniformly at random, and then we obtain an item according to the probability distribution in the row corresponding to that position. Let's assume that we sample position 2 and item 3. Therefore, our partially sampled solution is $\sigma = -3--$. Now, in order to sample the next position we move to $\mathbf{M}_2$. From all the rows in this matrix that have position 2, i.e, (1,2), (2,3) and (2,4), we choose uniformly at random one of them, for instance (2,3). Now we have to sample the probability distribution in that row. However, we have to take into account that only pairs of items that assign item 3 to position 2 are valid. Therefore, the probability distribution is modified to comply with that constraint. We can imagine that we obtain the assignment (3,4) for the positions (2,3). So, our partially sampled individual is now $\sigma = -34-$. This process will continue in $\mathbf{M}_2$ until we have a complete individual.

Table 3.3: Example of creating an individual with the introduced sampling technique

Initialization
The individual is initialized as empty $\sigma = - - - - -$

Step 1
The uniformly random selected position is 2.
$\mathbf{M}_1$ marginals matrix:

	1	*2*	*3*	*4*
1	0.41	0.16	0.16	0.25
2	0.09	0.50	0.25	0.16
3	0.25	0.16	0.33	0.25
4	0.25	0.16	0.25	0.33

Sampled item is 3.

Step 2
Partially sampled individual is $\sigma = -3 - -$
The uniformly random selected combination of positions is $(2, 3)$.
$\mathbf{M}_2$ marginals matrix:

	(1,2)	(1,3)	(1,4)	(2,1)	(2,3)	(2,4)	***(3,1)***	***(3,2)***	***(3,4)***	(4,1)	(4,2)	(4,3)
(1,2)	0.20	0.05	0.10	0.05	0.10	0.05	0.05	0.10	0.05	0.05	0.10	0.10
(1,3)	0.05	0.20	0.10	0.10	0.05	0.05	0.05	0.05	0.10	0.10	0.10	0.05
(1,4)	0.10	0.10	0.15	0.05	0.05	0.10	0.10	0.05	0.05	0.10	0.05	0.10
(2,3)	0.05	0.05	0.05	0.10	0.15	0.15	0.40	0.40	0.20	0.05	0.05	0.10
(2,4)	0.05	0.05	0.05	0.10	0.15	0.15	0.10	0.05	0.10	0.05	0.10	0.05
(3,4)	0.05	0.10	0.10	0.10	0.05	0.05	0.05	0.10	0.15	0.10	0.05	0.10

Sampled items combination is $(3, 2)$.

Step 3
Partially sampled individual is $\sigma = -32-$
The uniformly random selected combination of positions is $(3, 2)$.
$\mathbf{M}_2$ marginals matrix:

	(1,2)	(1,3)	(1,4)	***(2,1)***	*(2,3)*	***(2,4)***	(3,1)	(3,2)	(3,4)	(4,1)	(4,2)	(4,3)
(1,2)	0.20	0.05	0.10	0.05	0.10	0.05	0.05	0.10	0.05	0.05	0.10	0.10
(1,3)	0.05	0.20	0.10	0.10	0.05	0.05	0.05	0.05	0.10	0.10	0.10	0.05
(1,4)	0.10	0.10	0.15	0.05	0.05	0.10	0.10	0.05	0.05	0.10	0.05	0.10
(2,3)	0.05	0.05	0.05	0.10	0.15	0.15	0.10	0.10	0.05	0.05	0.05	0.10
(2,4)	0.05	0.05	0.05	0.10	0.15	0.15	0.10	0.05	0.10	0.05	0.10	0.05
(3,4)	0.05	0.10	0.10	0.66	0.05	0.33	0.05	0.10	0.15	0.10	0.05	0.10

Sampled items combination is $(2, 1)$.

Step 4
Partially sampled individual is $\sigma = -321$
The remaining item is placed in position 1 to end the sampling of the solution. As a result of the sampling process, the new individual is $\sigma = 4321$.

1 Get the list of rows P from $\mathbf{M}_r$ that match σ.
2 Uniformly at random pick a row from P.
3 Get the list of columns V from $\mathbf{M}_r$ that match σ.
4 Modify the probability distribution in the row for the list of columns V.
5 Sample a column from V.
6 Update the individual σ with the new assignment.
7 **if** $r == k$
8 Go to Step 11.
9 Update order counter $r \leftarrow r + 1$.
10 **if** σ is not completed
11 Go to Step 3.
12 Return σ.

Alg. 3.1: The sampling algorithm of the marginals matrix $\mathbf{M}_k$.

3.3 Experiments

In order to analyze the competitiveness of the k-order marginals EDAs, we have performed experiments on instances of the TSP, PFSP and QAP.

It is reasonable to expect that, by increasing k (order), a larger population should be used to estimate an appropriated probabilistic model over the permutations space. For that reason, different population sizes have been tested for the instances. Particularly, seven different population sizes have been used starting from $10n$ (being n the problem size) and multiplying this value by a factor of 2 until an upper bound of $640n$ is reached.

From the set of possible parameter values commonly used for EDAs, we have set the following without performing any previous test:

- k: $\{1, 2, 3\}$.
- Population sizes: $\{10n, 20n, 40n, 80n, 160n, 320n, 640n\}$.
- Selection size: Population size / 2.
- Offspring size: Population size - 1.
- Selection type: truncation selection method.
- Elitism criterion is followed.
- A maximum number of $100n$ generations are performed.

Additionally, in order to avoid premature convergence, we apply a technique to control evolution pressure introduced by Baluja [10] for PBIL. This method allows to maintain information from the previous generation in order to control the evolution rate of the population. At each step, the new probabilistic model is built merging the probabilistic model from the previous generation with the one learnt from the current population. The merging scheme is detailed with the following equation:

$$\mathbf{M}_k^t = (\alpha - 1)\mathbf{M}'_k + \alpha \mathbf{M}_k^{t-1} \tag{3.4}$$

where $\mathbf{M}'_k$ denotes the probabilistic learnt from the current sample and α stands for the proportion of the $\mathbf{M}_k^{t-1}$ maintained.

In our experiments, we set $\alpha = 0.9$. Regarding the instances, we have chosen the following three: *gr17.tsp* [148], *tai20_10_0.pfsp* [169] and *tai30b.qap* [169].

Figure 3.1 shows the performance of the k-order marginals EDA for these instances. Tables 3.4, 3.5 and 3.6 show the best, mean and standard deviation of the fitness values obtained over 10 repetitions of the EDAs with $k = 1$, $k = 2$ and $k = 3$ (only for *gr17.tsp*) order marginals for different population sizes.

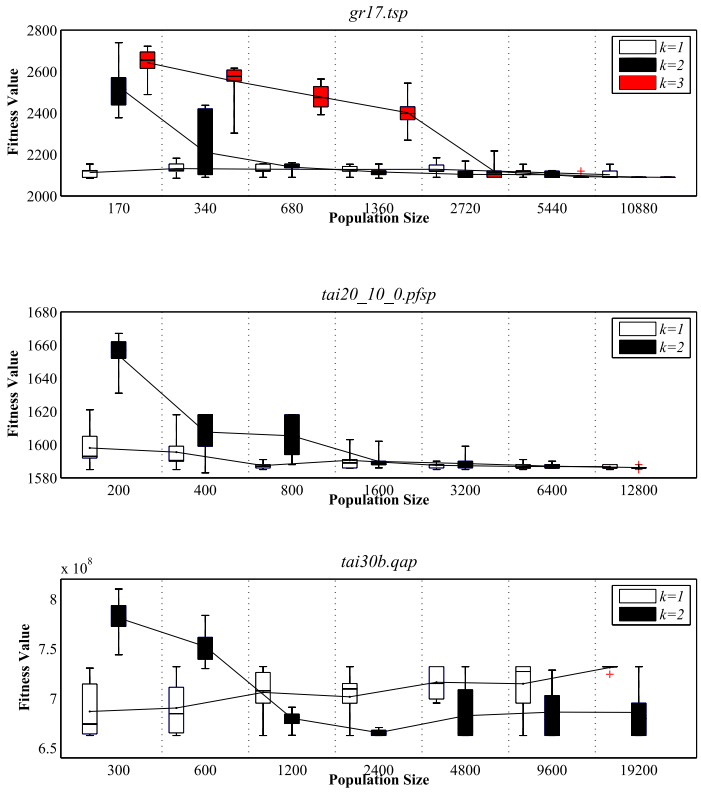


Fig. 3.1: Box-plot diagrams of the results of k-order marginals EDAs on the instances *gr17.tsp*, *tai20_10_0.pfsp* and *tai30b.qap*.

Table 3.4: Results of the k-order marginals EDAs for *gr17.tsp*.

Pop. Size	$k=1$			$k=2$			$k=3$		
	Best	Mean	Dev.	Best	Mean	Dev.	Best	Mean	Dev.
170	2085	**2112.5**	23.6	2377	2526.8	107.0	2489	2642.7	67.5
340	2085	**2132.1**	26.4	2090	2210.9	153.4	2303	2553.5	92.1
680	2090	**2129.0**	23.2	2090	2138.9	26.5	2392	2477.0	56.0
1360	2090	2127.7	20.1	2085	**2115.5**	19.3	2269	2401.7	71.1
2720	2090	2128.6	31.2	2090	**2103.8**	25.7	2090	2118.9	37.9
5440	2090	2115.6	18.2	2090	2102.3	15.9	2090	**2093.0**	9.4
10880	2090	2102.3	21.7	2090	**2090.0**	0.0	2090	**2090.0**	0.0

Table 3.5: Results of the k-order marginals EDAs for *tai20_10_0.pfsp*.

Pop. Size	$k=1$			$k=2$		
	Best	Mean	Dev.	Best	Mean	Dev.
200	1585	**1598.0**	12.5	1631	1653.8	12.9
400	1585	**1595.5**	11.4	1583	1607.6	14.7
800	1585	**1587.4**	1.9	1588	1605.3	12.0
1600	1586	**1590.6**	5.6	1586	1589.9	4.9
3200	1585	**1587.5**	1.7	1585	1588.6	4.1
6400	1585	**1586.8**	1.7	1586	1587.0	1.4
12800	1585	1586.7	1.1	1585	**1586.2**	1.0

Table 3.6: Results of the k-order marginals EDAs for *tai30b.qap*. All the values are $\times 10^5$.

Pop. Size	$k=1$			$k=2$		
	Best	Mean	Dev.	Best	Mean	Dev.
300	6624.4	**6866.5**	263.8	7438.4	7808.9	188.1
600	6624.4	**6902.1**	277.9	7299.3	7520.6	155.3
1200	6624.4	7058.9	217.2	6626.7	**6798.0**	84.8
2400	6624.4	7013.9	222.1	6624.4	**6653.1**	314.5
4800	6951.4	7161.3	154.2	6624.4	**6824.4**	294.5
9600	6624.4	7146.3	246.4	6624.4	**6860.8**	273.4
19200	7241.3	7309.9	241.0	6624.4	**6857.3**	273.2

Results show that the performance of the EDAs ($k = 1, 2, 3$) varies with respect to the population size. In particular, we see that when the population size is small, the best algorithm is $k = 1$ in all the problems. Nevertheless, as the size of the population increases, the EDAs with higher order marginal probabilities obtain better results. Furthermore, results in Fig. 3.1 show that there is a population size above which the mean values obtained by the EDAs with highest k are better.

We think that the reason for such behaviour is as follows, when k increases, the number of samples needed to fill the k order marginals matrix $\mathbf{M}_k$ increases as well. Therefore, when using high order marginal EDAs and small populations, the probabilistic model estimated at every generation is not accurate enough to carry out an efficient optimisation process.

Although more complex models provide the best results once a critical population size is reached, the computational cost of dealing with such populations is large. As a result, only small-size problems are suitable to be solved with complex models, as is the case of *grt17.tsp* and $k = 3$.

A singular behavior has also been seen in these experiments. Initially we expected that higher population sizes would allow the model to represent the search space better, and therefore, the results of the EDA were expected to be better. However, if we analyze the mean values of Table 3.6 for $k = 1$ and $k = 2$, the same behavior is repeated: initially the mean decreases (as we are minimizing) while we increase the population size, and suddenly the mean begins to increase slightly.

3.4 Discussion

In this chapter, an EDA for permutation-based problems has been designed. Particularly, we have based our models on k-order marginals in order to set a parallelism between these models and those used for integer problems. However, there are subtle differences that make this task tricky.

We have used the matrix $\mathbf{M}_k$ as a probability model to capture the characteristics of the individuals. Nevertheless, we do not have an explicit expression for the probability distribution we are sampling. In fact, the only information we have is that the marginals of the sampled distribution coincide with $\mathbf{M}_k$.

Following the parallelism with EDAs designed for integer problems, we would like to have, given the k-orders marginals, the probability with maximal entropy. However, in the case of permutations, this is difficult to find [2]. It is not clear from the above reference paper when it is possible to sample that probability distribution without an explicit expression for it.

3.5 Conclusions and future work

In this chapter, we have introduced a new approach for EDAs based on k-order marginals to solve optimization problems represented by permutations. We completed some experiments to understand the performance of k-order marginals modelling a probability distribution over the search space of permutations.

As we have seen, the models we have considered do not obtain outstanding results. We think that such behaviour could be due to two issues. The first one is the population size. In order to learn 2nd order or higher order marginals,

the population size has to be really large. Secondly, we wonder when it makes sense to keep the relation between the items of two positions that are far apart in the individual. In this sense, we are considering the possibility of restricting the k-order marginals to positions that are close to some threshold in the individual. This change would additionally reduce the memory and computation requirements of the algorithms.

4

A distance-based exponential probability model EDA for the permutation flowshop scheduling problem

4.1 Introduction

Since Johnson published his work on the two-machine flowshop in 1954 [84], numerous papers have dealt with the flowshop scheduling problem. Early research on this topic was highly theoretical and focused mainly on exact methods. In 1976, Garey et al. [64] proved that this problem is *NP-complete* for more than two machine instances, thus stating the difficulty of achieving optimal solutions. In later decades, the problem was extensively addressed in the literature [71, 73]. A simplified version of this problem is the permutation flowshop scheduling problem (PFSP) [71], where the goal is to find the optimal sequence of n jobs that must be processed in m machines. Even though the PFSP was formulated in the 50's, it is still nowadays a meaningful research topic due to its strong engineering background. Initially, the most common optimization criterion in the PFSP was the total completion time of the jobs, also called makespan. However, the minimization of the total flow time (TFT) has captured the attention of the scientific community since it is more relevant than the makespan for the current dynamic production environments. The TFT measures the sum of the times that each job remains in the flow. Thus, minimizing the TFT can lead to the stable utilization of resources, rapid turn-around of jobs, and minimization of work-in-progress inventory costs [57].

The literature contains evidence of a wide variety of strategies that approach the PFSP with respect to the TFT criterion (from now on PFSP-TFT). Exact algorithms accurately work out the optimal solution. However, they are computationally costly and, with the exception of small size examples (instances up to 10-15 jobs), exact algorithms can not be used in a reasonable time span. Examples of such methods are Branch & Bound algorithms [44, 185]. Besides the exact methods, heuristic procedures such as Constructive Heuristics [57, 101, 105, 131, 144, 145, 188] and Composite Heuristics [5, 58, 105, 197] have also been proposed. Unfortunately, such heuristics

have shown a poor performance when solving PFSP instances of 50 jobs or more.

Advances in metaheuristic optimization supplied the research community with new techniques to tackle combinatorial optimization problems. As proof of these advances, we found a wide variety of strategies in the literature for solving the PFSP, such as Local Search-based algorithms [47, 53, 87], Simulated Annealing [134], Tabu Search [133, 147, 167], Genetic Algorithms (GAs) [130, 175, 184, 189, 195], Particle Swarm Optimization [82, 102, 104, 170], Ant Colony Optimization [59, 146, 165, 196] or Estimation of Distribution Algorithms (EDAs) [81]. Nowadays, the literature points to the hybrid approaches, such as those published by Costa et al. (Variable neighbourhood Search 4 - VNS_4) [47] and Xu et al. (Asynchronous Genetic Algorithm - AGA) [189], as the cutting-edge algorithms for optimizing the PFSP-TFT. In the first case, the authors propose the combination of a variable neighbourhood search (VNS) with a constructive algorithm called LR(n/m) [105]. In the second case, a GA is hybridized with VNS. Additionally, in order to guide the initial population, LR(n/m) is also used.

In this chapter, we go deeper into the development of EDAs, proposing (1) a new EDA for permutation-based optimization problems and (2) testing its performance on the PFSP-TFT.

Along the same research line proposed in the discussion of Chapter 2, Ceberio et al. [29] introduced the Mallows EDA, the first attempt to apply a probabilistic model that estimates an explicit probability distribution in the domain of permutations. Given a distance metric on permutations, and two parameters – the central permutation σ_0 and the spread parameter θ– the Mallows model is a distance-based exponential probability model considered analogous to the Gaussian probability distribution. The results reported in [29] showed that the Mallows EDA is able to outperform the best performing EDAs, EHBSA and NHBSA, for the PFPS with the makespan criterion.

In this chapter we propose a generalization of the Mallows model [56] in the framework of EDAs. The resulting algorithm, called the Generalized Mallows EDA (GMEDA), introduces the first serious attempt to apply parametric models for solving permutation-based problems under the framework of EDAs.

In order to demonstrate the validity of our proposal, we approach the PFSP-TFT with a hybrid version of the GMEDA. The hybrid GMEDA is a two-stage algorithm. In the first stage, it runs the GMEDA, and the resulting individual is used as the initial solution for the VNS employed in the second stage. Experimental results show that the hybrid GMEDA is competitive with the state-of-the-art algorithms, being able to obtain new best known results in 152 instances of the 220 tested.

The remainder of the chapter is organized as follows: in the following section, the Generalized Mallows model is described in detail. Section 4.3 introduces the GMEDA. In Section 4.4, the performance of the GMEDA over some PFSP instances is studied. Our hybrid proposal for solving the PFSP and its application to several benchmark instances is introduced afterwards,

in Section 4.5. We discuss the results in Section 4.6. Finally, some conclusions and future work ideas are presented in Section 4.7.

4.2 The Generalized Mallows model

Until the 50's, probability models for permutations were not extensively developed. The psychology domain was the first which demanded specific models to efficiently manage probability distributions over permutations. Early works proposed Thurstonian order statistic models [172]. By the end of the decade, a probabilistic model for permutations called the Mallows model [110] was introduced. The Mallows model is a probabilistic model which has been considered as analogous over permutations to the Gaussian distribution: they are both exponential unimodal models defined by two parameters, the mean and the variance.

Despite having been proposed in the late 50's, it is a lively model which has recently received the attention of the statistical and machine learning communities, and has been applied to problems from diverse fields such as recommender systems [108], multi-label classification [38, 98], data modelling [114], clustering [123] or computer vision [97].

Under this model, the probability value of every permutation $\sigma \in \mathbb{S}_n$ (where $\mathbb{S}_n$ stands for the set of $n!$ permutations of n items) depends on just two parameters: a spread parameter θ, and the distance to a central permutation σ_0, which is calculated by a particular distance metric $D(\sigma, \sigma_0)$. Formally, the Mallows model is defined as follows:

$$P(\sigma) = \psi(\theta)^{-1} \exp(-\theta D(\sigma, \sigma_0)), \tag{4.1}$$

where $\psi(\theta)$ is a normalization constant.

When θ equals 0, Eq. 4.1 assigns equal probability to every permutation σ in $\mathbb{S}_n$. The larger the value of θ, the more peaked the distribution becomes around the central permutation (we do not consider the scenario $\theta < 0$). The second parameter is the central permutation σ_0. This model assigns every permutation $\sigma \in \mathbb{S}_n$ a probability that decays exponentially with respect to its distance to the central permutation σ_0. In this way, $P(\sigma)$ can be considered as an exponential, non-uniform *distance-based probability model*.

The distance metric the Mallows model is traditionally coupled with in the literature (and used in this chapter) is *Kendall's-τ*. Denoted as $D_\tau(\sigma, \pi)$, it measures the number of pairs of items for which σ and π have opposing ordering. An equivalent definition for Kendall's-τ is the minimum number of adjacent transpositions needed to bring σ^{-1} into π^{-1}:

$$\begin{aligned} D_\tau(\sigma, \pi) = |\{(i,j) : i < j,\ (\sigma(i) < \sigma(j) \wedge \pi(i) > \pi(j)) \\ \vee (\pi(i) < \pi(j) \wedge \sigma(i) > \sigma(j))\}| \end{aligned}$$

Kendall's-τ has the *right invariant* property since $D_\tau(\sigma,\pi) = D_\tau(\sigma\gamma,\pi\gamma)$ for every permutation $\gamma \in \mathbb{S}_n$, where $\sigma\gamma$ stands for the composition of the permutations σ and γ, and is defined as $\sigma\gamma(i) = (\sigma \circ \gamma(i)) = \sigma(\gamma(i))$. Thus, by *right invariance*, when $\gamma = \pi^{-1}$, $D_\tau(\sigma,\pi)$ can be simplified as $D_\tau(\sigma\pi^{-1}, e)$ (also denoted as $D_\tau(\sigma\pi^{-1})$), where e stands for the identity permutation $e = 123\ldots n$.

The analogies between the Gaussian and Mallows distributions can be seen by looking at Fig. 4.1. In this figure, every permutation in $\sigma \in \mathbb{S}_5$ (the set of 120 permutations of five items) is placed on the x-axis and their probability values for different θ are plot. The central permutation σ_0 is that with the highest probability value in the middle of the x axis. For representation purposes, all the permutations at the same distance from σ_0 have been equally divided into two groups and drawn on the x-axis in each side of the central permutation.

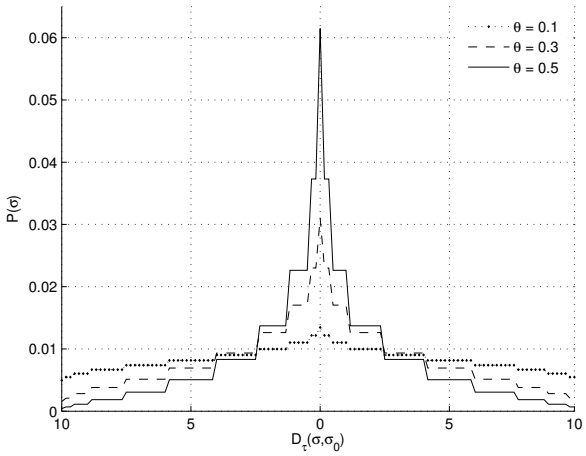


Fig. 4.1: Probability value for each permutation in $\mathbb{S}_5$ for different values of the spread parameter θ.

Among the many proposals, the Generalized Mallows model (GM) [56] is the most popular extension of the Mallows model. This extension requires the distance to be decomposed into $n-1$ terms, in such a way that it can be expressed as $D(\sigma,\sigma_0) = \sum_{j=1}^{n-1} S_j(\sigma,\sigma_0)$, where $S_j(\sigma,\sigma_0)$ is related with the j-th position of the permutation. The central permutation σ_0 in the GM model is also the permutation with the highest probability mass. However, instead of a single spread parameter θ, GM makes use of $n-1$ spread parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_{n-1})$, each θ_j affecting a particular position j of the permutation. The GM model is thus given by the following formula:

$$P(\sigma) = \psi(\boldsymbol{\theta})^{-1} \exp\left(\sum_{j=1}^{n-1} -\theta_j S_j(\sigma, \sigma_0)\right) \tag{4.2}$$

Because of the right invariance property, $S_j(\sigma, \sigma_0)$ can be rewritten as $S_j(\sigma\sigma_0^{-1}, e) = S_j(\sigma\sigma_0^{-1})$. In the particular case of the Kendall's-τ distance, the S_j terms are denoted by V_j, and are defined as $V_j(\pi) = \sum_{i=j+1}^{n} I[\pi(j) > \pi(i)]$, where $I[cond]$ equals 1 when the condition $cond$ equals true, and 0 otherwise. Basically, $V_j(\pi)$ is the number of positions of the permutation on the right of j with values smaller than $\pi(j)$. It follows from the definition that $V_j(\pi)$ ranges from 0 to $n - j$ for $1 \leq j < n$.

Taking into account that $D_\tau(\pi) = \sum_{j=1}^{n-1} V_j(\pi)$, Eq. 4.2 can be written as

$$P(\sigma) = \psi(\boldsymbol{\theta})^{-1} \exp\left(\sum_{j=1}^{n-1} -\theta_j V_j(\sigma\sigma_0^{-1})\right) \tag{4.3}$$

for every $\sigma \in \mathbb{S}_n$. It is worth noting that one can uniquely determine any permutation π with the $n-1$ integers $V_1(\pi), ..., V_{n-1}(\pi)$ in which the Kendall's-τ distance $D_\tau(\pi)$ decomposes.

Under the uniform distribution, the $V_j(\pi)$ variables that define a permutation are independent and, as a consequence [56], the probability distribution of the random variables $V_j(\sigma\sigma_0^{-1})$ under the GM model given by Eq. 4.3 can be written as follows:

$$P(V_j(\sigma\sigma_0^{-1}) = r_j) = \frac{\exp(-\theta_j r_j)}{\psi_j(\theta_j)} \qquad r_j \in \{0, ..., n - j\} \tag{4.4}$$

Furthermore, the normalization constant $\psi(\boldsymbol{\theta})$, which if calculated naively requires $n!$ sums, can be simplified as the product of $n - 1$ terms,

$$\psi(\boldsymbol{\theta}) = \prod_{j=1}^{n-1} \psi_j(\theta_j) = \prod_{j=1}^{n-1} \frac{1 - \exp(-\theta_j(n - j + 1))}{1 - \exp(-\theta_j)} \tag{4.5}$$

The most important consequence of Eq. 4.4 and 4.5 is that, when the GM model considers the Kendall's-τ distance, it can be expressed as a *Multistage ranking model* [55]. This means that the process of constructing a permutation has $n - 1$ stages and the probabilities at a given stage depend only on the current stage, i.e., at stage j the probability $P(V_j(\pi))$, and equivalently the value of $V_j(\pi)$, is not affected by the decisions made at previous stages and, in the same way, this decision will not affect posterior stages. This property implies that the probability distribution of a given permutation σ with $\boldsymbol{V}(\sigma\sigma_0^{-1}) = (r_1, \ldots, r_{n-1})$ can be factorized as:

$$P(\sigma) = \prod_{j=1}^{n-1} P(V_j(\sigma\sigma_0^{-1}) = r_j) \tag{4.6}$$

4.2.1 Learning and sampling the Generalized Mallows model

In order to introduce the GM model into the framework of EDAs, it is necessary to be able to learn the parameters of the model and sample new solutions from it in an efficient way. In this section we detail how we carry out both processes, which are intended to maintain a trade-off between time-complexity and accuracy.

The most common way of learning the parameters of probability distributions is by means of the maximum likelihood estimation (MLE) principle. The MLE parameters, given a sample of N permutations $\{\sigma_1, \ldots, \sigma_N\}$, are the parameters σ_0 and $\boldsymbol{\theta}$ that maximize the likelihood function (we consider its logarithm):

$$ln\ L(\boldsymbol{\theta}, \sigma_0 | \sigma_1, ..., \sigma_N) = -N \sum_{j=1}^{n-1} [\theta_j \bar{V}_j - ln(\psi_j(\theta_j))] \tag{4.7}$$

where

$$\bar{V}_j = \frac{1}{N} \sum_{i=1}^{N} V_j(\sigma_i \sigma_0^{-1}) \tag{4.8}$$

There are several heuristics and exact methods to simultaneously obtain the parameters $\boldsymbol{\theta}$ and σ_0 [113, 122]. However, for the sake of time efficiency (this process is repeated at each generation inside the EDA), we have divided the learning process into two stages, dealing first with the calculation of the central permutation σ_0 and then with the spread parameters $\boldsymbol{\theta}$.

Maximizing the log-likelihood with respect to σ_0 given a sample of N permutations $\{\sigma_1, \ldots, \sigma_N\}$ is equivalent to minimizing $\sum_{j=1}^{n-1} \theta_j \bar{V}_j(\sigma_i \sigma_0^{-1})$. If all θ_j were equal (the case of the Mallows model), this problem would be the same as that of minimizing the sum of the distances with every permutation in the sample, which is a well known *NP-hard* problem [11], referred to as *consensus permutation*. Returning to the GM model, the first stage of the learning process consists of computing an approximation to the central/consensus permutation by using the *Borda* algorithm [18], which is carried out in $O(nN)$. Borda is a simple, yet accurate algorithm that calculates the average permutation of a given sample, which is, in fact, a consistent estimator of the consensus permutation [55]. Its pseudo-code is given in Alg. 4.1. In [4], several methods for the consensus permutation problem are compared, concluding that Borda is one of the most efficient algorithms.

Once the consensus permutation σ_0 is approximated, the second stage of the learning process consists of computing the dispersion parameters θ_j for the previously computed $\hat{\sigma}_0$. The maximum likelihood estimators of the spread parameters, given the central permutation, are calculated by solving the following equation:

$$\bar{V}_j = \frac{1}{\exp(\theta_j) - 1} - \frac{n - j + 1}{\exp(\theta_j (n - j + 1)) - 1} \qquad j = 1, \ldots, n-1 \tag{4.9}$$

```
1  input: A sample of N permutations {σ_1, ..., σ_N}
2    for j from 1 to n
3      π(j) = Σ_{i=1}^{N} σ_i(j)/N;
4    end for
5    visited = {};
6    for j from 1 to n
7      min_i ← arg min_i {π(i) | i ∉ visited};
8      σ̂_0(min_i) = j;
9      visited ← visited ∪ min_i;
10   end for
11 output: σ̂_0
```

Alg. 4.1: Borda algorithm to estimate the consensus permutation $\hat{\sigma}_0$.

Eq. 4.9 does not have a closed form expression, but can be numerically solved by standard iterative algorithms for convex optimization. In this case, we use the Newton-Raphson algorithm. Since the number of iterations performed by this algorithm depends on the initial solution assigned, its time-complexity cannot be exactly determined. However, when the initial solution is a good approximation, $O((\log h)F(h))$ is close to the real time-complexity. $F(h)$ denotes the cost of calculating the $f(\theta)/f'(\theta)$ term with a h-digit precision in the Newton-Raphson algorithm (note that $h << n$)[1]. Since we have $n-1$ spread parameters, the overall time-complexity of estimating the θ_j parameters is $O((n-1)(\log h)F(h))$ [19].

As regards the sampling process, in Eq. 4.6 we showed that the probability distribution defined by the GM model can be factorized into $n-1$ terms. From a computational point of view, this decomposition turns out to be a very efficient way of sampling the probability distribution, a necessary condition in the context of EDAs. The process of generating the new offsprings follows by drawing the $\boldsymbol{V}(\sigma_s\sigma_0^{-1})$ vectors with the distribution in Eq. 4.4 and Eq. 4.6, $1 \leq s \leq M$, where M denotes the number of sampled solutions. Then, by applying the algorithm proposed by Meila et al. [122], the corresponding permutation $(\sigma_s\sigma_0^{-1})^{-1}$ is obtained (see pseudo-code in Alg. 4.2). Finally, each permutation σ_s is obtained by inverting and composing with the consensus permutation σ_0, $((\sigma_s\sigma_0^{-1})^{-1})^{-1}\sigma_0 = \sigma_s\sigma_0^{-1}\sigma_0 = \sigma_s$. The overall sampling procedure of one solution is carried out in $O(n^2)$.

In order to illustrate the conversion from $\boldsymbol{V}(\pi)$ to π^{-1}, Fig. 4.2 introduces an example.

The sampling step in the EDA takes time $O(n^2M)$, being M the number of new solutions sampled at each generation. Since the complexity of the

[1] The precision parameter h is set to 10^{-4}.

```
1 input: A vector of V(π)
2    Create an empty list π^{-1}
3    Insert n in position 1
4    for j = n − 1 : 1 in decreasing order
5       Insert j in position V_j of π^{-1}
6 output: π^{-1}
```

Alg. 4.2: Conversion from $\boldsymbol{V}(\pi)$ to π^{-1}.

Input: $\boldsymbol{V}(\pi) = (2,\ 0,\ 1)$

insert $n = 4$ in	0	$\rightarrow$	$\pi^{-1} = (\mathbf{4})$
insert $j = 3$ in $V_3(\pi) = 1$		$\rightarrow$	$\pi^{-1} = (4\ \mathbf{3})$
insert $j = 2$ in $V_2(\pi) = 0$		$\rightarrow$	$\pi^{-1} = (\mathbf{2}\ 4\ 3)$
insert $j = 1$ in $V_1(\pi) = 2$		$\rightarrow$	$\pi^{-1} = (2\ 4\ \mathbf{1}\ 3)$

Output: $\pi^{-1} = (2\ 4\ 1\ 3)$

Fig. 4.2: Example of the conversion from $\boldsymbol{V}(\pi)$ to π^{-1}.

sampling step is higher than that of the learning step, Borda-$O(nN)$ and Newton-Raphson- $O((n-1)(\log h)F(h))$, then the overall time-complexity of a generation of the GMEDA is dominated by this term, and is $O(n^2M)$.

4.3 Generalized Mallows EDA

The Generalized Mallows EDA is basically a classical EDA where the probabilistic model used is the GM model introduced in the previous section. In Alg. 4.3 the general outline of the GMEDA is described.

The behaviour of the GMEDA not only depends on the general parameters of a classical EDA (population size, selection mechanism, stopping criterion, etc.), but also on the specific parameters of the probabilistic model used. In this particular case, the vector $\boldsymbol{\theta}$ is the set of parameters that determines the shape of the GM model, and thus, it is closely related to the behaviour of the EDA. Therefore, before applying the algorithm in a general setting, we consider it very valuable to analyze the behaviour of the GMEDA in relation to these parameters.

In order to understand such behaviour, we analyze the evolution of the GM model through the generations. To do that, we carried out a preliminary experiment studying the average value of the vector $\boldsymbol{\theta}$ at each generation. One instance from the Taillard's PFSP benchmark [169] of size 20, 50 and

1 $D_0 \leftarrow$ Generate M individuals (the initial population) uniformly at random.
2 $t \leftarrow 1$
3 **do**
4 $\quad D_{t-1} \leftarrow$ Evaluate individuals.
5 $\quad D_{t-1}^{Se} \leftarrow$ Select $N \leq M$ individuals from D_{t-1}.
6 $\quad \hat{\sigma}_0 \leftarrow$ Estimate the consensus permutation from D_{t-1}^{Se}.
7 $\quad \hat{\boldsymbol{\theta}} \leftarrow$ Estimate the spread parameters from D_{t-1}^{Se} and $\hat{\sigma}_0$.
8 $\quad D_M \leftarrow$ Sample $M-1$ individuals from $p_t(\sigma) = p(\sigma | D_{t-1}^{Se}, \hat{\sigma}_0, \hat{\boldsymbol{\theta}})$.
9 $\quad D_t \leftarrow$ Create the new population from D_{t-1} and D_M.
10 **until** Stopping criterion is met

Alg. 4.3: The Generalized Mallows EDA.

100 (introduced later in Section 4.4) are used. The general parameters of the GMEDA are also described in the next section.

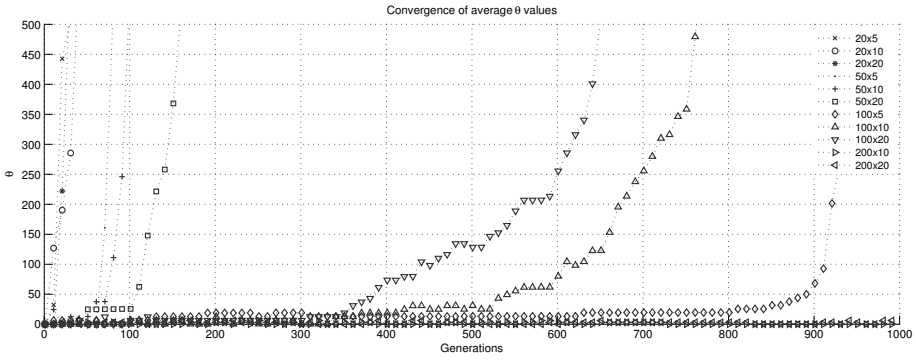


Fig. 4.3: Convergence of the average $\boldsymbol{\theta}$ values for the first instances of the 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10 and 100×20 configurations of the Taillard's benchmark.

According to the results shown in Fig. 4.3, in the initial generations the average $\boldsymbol{\theta}$ values are close to 0 for all the instances, denoting a diverse population, which is reasonable taking into account the random nature of the initialization procedure. As the generations move forward, the average $\boldsymbol{\theta}$ values increase rapidly, reaching values of up to 500. In the smallest size instances -those with 20 or 50 jobs- the average $\boldsymbol{\theta}$ values converge much faster than for instances of 100 jobs. However, in all the cases, the results indicate that, after

a certain amount of generations, the convergence of the population accelerates dramatically.

With the aim of understanding the meaning of the average $\boldsymbol{\theta}$ values seen in Fig. 4.3, we studied the probability assigned to the consensus permutation σ_0 in relation to $\boldsymbol{\theta}$ (the GM model considers $n-1$ spread parameters, but for the sake of simplicity, we set the same value of $\boldsymbol{\theta}$). According to the results shown in Fig. 4.4, for $\boldsymbol{\theta}$ values below 2, the probability of σ_0 is close to 0, thus encouraging an exploration stage. However, once a threshold is reached, the probability assigned to σ_0 increases quickly, leading the algorithm to an exploitation phase.

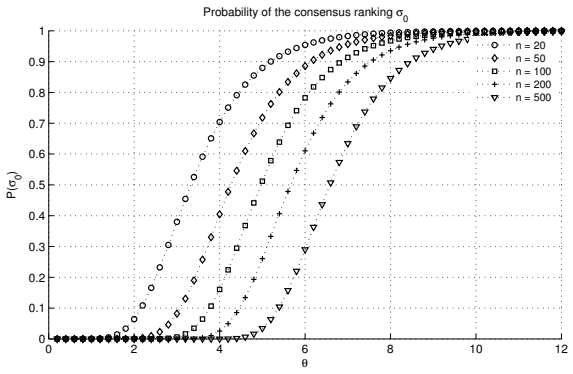


Fig. 4.4: Probability assigned to σ_0 for different $\boldsymbol{\theta}$ and n values.

In view of the results in Fig. 4.4, when θ takes values higher than 10, the probability of σ_0 is almost 1, and Fig. 4.3 shows that, in the first few generations of the EDA, such a threshold is easily reached. In order to prevent the EDA from getting stuck prematurely, and to enhance the exploration ability of the model, we decided to set an upper bound value θ_{upper} to the spread parameters so that a trade-off between exploitation and exploration is guaranteed.

4.4 Generalized Mallows EDA for PFSP

As mentioned previously, in order to analyze the performance of the new GMEDA, we apply our proposal to an extensively studied problem in the literature: the permutation flowshop scheduling problem with respect to the total flowtime criterion. The PFSP has already been presented in Chapter 1, however, in view of the learning algorithms used to estimate the parameters σ_0 and $\boldsymbol{\theta}$, and motivated by complexity issues, we propose a slight modification of the encoding: a solution of the PFSP will be codified as a permutation of

the items $\{1, \ldots, n\}$ where $\sigma(i)$ is the position of the job i in the scheduling. For instance, the solution $\sigma = 231$ denotes that job 1 is scheduled second, job 2 is scheduled the last, and job 3 is scheduled first.

Although several algorithms have been presented in the literature, in order to obtain real feedback, we compare our approach with the best two state-of-the-art algorithms: Asynchronous Genetic Algorithm (AGA) [189] and Variable neighbourhood Search 4 (VNS_4) [47].

AGA hybridizes a GA with a VNS. The algorithm starts initializing the population randomly, with the exception of one solution, which is calculated with the constructive algorithm LR(n/m) [105]. At each iteration, $N/2$ pairs of individuals in the population (being N the population size) are selected uniformly at random and submitted to an asynchronous crossover operator. The asynchronous crossover operator consists of three steps: first, an *enhanced VNS* (eVNS) is applied to the parent individuals. Next, a twice two-point crossover (TTP) is performed with the resulting solutions of the eVNS. Finally, the individuals obtained from the crossover are again submitted to the eVNS. The asynchronous behaviour of the operator is due to the number of improvements permitted in the local search procedures in the eVNS, the second call being much more powerful than the first. The best two individuals obtained from the crossover operator are added to the new population. Additionally, when all the individuals in the population have the same TFT value, the population is restarted randomly, keeping only the best solution in the new population.

The recently published paper by Costa et al. [47] introduces an exhaustive analysis of different variations of the VNS, concluding that VNS_4 (the version proposed in [47]) is the most successful one. VNS_4 explores solutions alternating between the insert neighbourhood (considered as the main neighbourhood) and the interchange neighbourhood. The insert operator considers as neighbours all those solutions that can be obtained by moving a job into another position. The interchange operator considers as neighbours those solutions that can be obtained by swapping a pair of jobs. When a local optima for both neighbourhoods is reached, a shake procedure is performed in order to escape from that area of the search space. The shake consists of randomly carried out 14 insert movements of jobs. In addition, VNS_4 also starts from an initial solution provided by the constructive algorithm LR(n/m) [105].

As it has not been possible to obtain the original codes of AGA and VNS_4 from the authors, we have implemented both proposals. The implementation of AGA and VNS_4 was made by honestly following the indications of the respective papers[2]. All the algorithms were coded in C++ programming language and the experimentation was conducted on a cluster of 20 nodes, each of them equipped with two Intel Xeon X5650 CPUs and 48GB of memory.

[2] Both programs, as well as supplementary material can be obtained from `http://www.sc.ehu.es/ccwbayes/members/jceberio/GMEDA.html`.

Regarding the experimentation benchmark, AGA, VNS_4 and GMEDA were tested using Taillard's [168] benchmark. This benchmark gathers a total of 120 instances with the following configurations of n jobs $\times$ m machines: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20 and 500×20 (10 instances of each type).

4.4.1 Parameter settings

The parameters employed by GMEDA are the following:

- Population size: $10n$.
- Selected individuals: n.
- Selection type: truncation.
- Sampled individuals: $10n - 1$.
- Elitism criterion is followed.
- Restart mechanism: as seen in the literature, the elitism criterion and truncation selection implemented in this approach may lead the algorithm to lose diversity in the population. In order to prevent GMEDA from getting stuck, a restart mechanism is applied when the fitness of all the individuals in the population is the same. The restart mechanism generates a new population by applying a shake procedure over the best individual found so far. The shake procedure consists of five random insert movements of jobs around their current position (the 5 preceding and 5 succeeding locations are considered). The seed individual used is not added to the population in order to avoid super-individual effects.
- The upper bound value for $\boldsymbol{\theta}$ was calculated by testing which value of θ_{upper} makes GMEDA perform the best over the first instance of each $n \times m$ configuration. Different θ_{upper} between θ_{min} and θ_{max} in steps of 0.1 were studied (the average value of 10 repetitions of the GMEDA was calculated (see appendix 4.7)).
- As regards the stopping criterion, AGA authors proposed setting an execution time of $n \times m \times 0.4$ seconds, as suggested in many other previous works. As this value is strongly related to the hardware used for the experiments and to the quality of the code, we think that the total number of evaluations is a much better criterion. In order to set a maximum number of evaluations, we run AGA for each instance for $n \times m \times 0.4$ seconds, recording the number of evaluations (see Appendix 4.7). These values were used throughout the experiments as the stopping criterion for GMEDA, AGA and VNS_4.

The parameters of AGA and VNS_4 have been set following the indications of their respective papers (with the exception of the stopping criterion).

4.4.2 Taillard's benchmark results

Each *algorithm* - *instance* pair is run 20 times. The performance measure employed in our study is the average relative percentage deviation (ARPD):

$$ARPD = \left(\sum_{i=1}^{20} \frac{(Algorithm_i - Best) \times 100}{Best}\right) / 20$$

being $Algorithm_i$ the solution found by the algorithm in the i-th repetition, and $Best$ the best known solution for the instance under study. The ARPD results of the executions are summarized in Tables 4.1, 4.2 and 4.3. Results in bold denote the algorithm that obtained the lowest relative error among the compared approaches. The results marked as (*) denote that the optimum or best known value has been achieved at least once among the performed repetitions. The best results in italics denote new best known solutions.

As can be observed (in columns AGA, VNS_4 and GMEDA), the efficiency of the algorithms varies with respect to the size of the problem. For instances of size 20, AGA and VNS_4 obtain the optimal results in all the cases. GMEDA, meanwhile, achieves the optimal result in 22 instances of 30. When comparing instances of 50, 100, 200 and 500, AGA clearly outperforms VNS_4[3] and GMEDA.

Although the results of GMEDA for the medium-large instances are non-competitive, it is worth noting the ability of GMEDA to achieve optimal solutions for the smallest size instances. Referring to the literature in the field, hybrid approaches are generally the best performing proposals when facing real problems, since they sum the abilities of the combined algorithms.

4.5 Hybrid Generalized Mallows EDA

In this section we introduce the Hybrid Generalized Mallows EDA (HGMEDA). The HGMEDA is proposed as a two-stage algorithm, which in the first stage executes the GMEDA previously introduced, and in the second stage applies a VNS to the solution provided by GMEDA. Since VNS is the most preferred algorithm seen in the literature with which to hybridize, in the following section we introduce a specific design of this algorithm.

4.5.1 Variable Neighbourhood Search

The VNS that we propose in this work explores solutions combining the two most used neighbourhoods in the literature: the *interchange* neighbourhood and the *insert* neighbourhood.

The interchange neighbourhood (N_S) considers that two permutations are neighbours if one is obtained by interchanging two elements in the other. It is formally defined as:

$$N_S(\sigma) = \{\sigma' | \sigma'_k = \sigma_k, \forall k \neq i, j, \sigma'_i = \sigma_j, \sigma'_j = \sigma_i\}$$

[3] The results reported by Costa et al. [47] do not match those observed in this experimentation. The authors in their paper report better results for VNS_4 and worse results for AGA.

On the other hand, two solutions are neighbours under the insert neighbourhood (N_I) if one is obtained by moving an element of the other solution to a different place. It is formally defined as:

$\sigma' \in N_I(\sigma) \Leftrightarrow \exists i, j \in \{1, \ldots, n\}$ with $i \neq j$ s.t.

$$\begin{cases} (\sigma'_k = \sigma_k,\ k < i \wedge k > j) \wedge \\ (\sigma'_k = \sigma_{k+1},\ i \leq k < j) \wedge & \text{when } i < j \\ \sigma'_j = \sigma_i \\ \\ (\sigma'_k = \sigma_k,\ k < i \wedge k > j) \wedge \\ (\sigma'_k = \sigma_{k-1},\ i < k \leq j) \wedge & \text{when } i > j \\ \sigma'_i = \sigma_j \end{cases}$$

The VNS performs as follows (see pseudo-code in Alg. 4.4): in a first step, a greedy local search procedure is carried out using the interchange neighbourhood ($LocalOpt(N_S, \sigma^C)$) until a local optimum is reached (σ^S). In the next step, the algorithm chooses the best neighbour of σ^S in N_I.

The algorithm repeats this procedure until a local optimum is found for both neighbourhoods. If the obtained solution σ^C is better than that found by the VNS, the best solution (σ_{best}) is updated. Next, a shake procedure, similar to that employed in the restart mechanism, is applied (this time 10 jobs are shifted instead of 5). The whole process is repeated until the maximum number of evaluations is reached.

4.5.2 Experimentation

In order to study the performance of HGMEDA, we repeat the previous experimentation.

4.5.2.1 Parameter tuning

Being HGMEDA a two stage algorithm, the number of evaluations assigned to each stage is set to half of the maximum allowed. In addition, the number of restarts allowed for GMEDA has been limited to $10n$ (these parameters have been decided without any previous experiment). In the case that the criterion on restarts is fulfilled before half of the evaluations have run, the remaining evaluations will be performed by the VNS. The rest of the parameters are set as described in Section 4.4.

4.5.2.2 Taillard's benchmark results

We evaluate AGA, VNS_4, GMEDA, VNS (the version used in our hybrid approach) and HGMEDA algorithms over the instances of Taillard's benchmark.

```
1  input: The best solution obtained by GMEDA, σ_best.
2     σ^C ← σ_best
3     do
4        do
5           σ^S ← LocalOpt(N_S, σ^C)
6           σ^C ← arg min_{σ∈N_I(σ^S)} f(σ)
7        until (no improvement) /* a local optimum is found */
8        if (f(σ^C) < f(σ_best)) then
9           σ_best ← σ^C
10       else
11          σ^C ← σ_best
12       Shake(σ^C)
13    until (MaxEvaluations are reached)
14 output: σ_best
```

Alg. 4.4: Variable neighbourhood Search.

Each *algorithm - instance* pair is run 20 times. The best result and ARPD values are summarized in Tables 4.1, 4.2 and 4.3.

According to these results, from the 120 Taillard's benchmark instances tested, HGMEDA finds the best TFT solution for 85 instances (for which 48 are new best known solutions), AGA for 70 and VNS_4 for 31. Regarding the ARPD results, AGA is the best in 88 instances, HGMEDA in 62, and VNS_4 is the best in 30 out of 120 instances.

In order to state whether there exist statistical differences among the results, we applied a non-parametric Friedman's test to the best TFT results obtained by the AGA, VNS_4 and HGMEDA algorithms among the 20 repetitions for each $n \times m$ configuration. A level $\alpha = 0.05$ of significance was set. The statistical test reported significant differences between the algorithms for some configuration types. Therefore, a post-hoc method was used to carry out all pairwise comparisons and determine which algorithms stand out from the rest of the approaches. In particular, Shaffer's static procedure is used, as suggested for such cases in [61]. Again, the significance level was fixed to $\alpha = 0.05$. The statistical analysis confirmed the following points:

- No significative difference was found for instances of size 20 between AGA, VNS_4 and HGMEDA.
- AGA and HGMEDA perform better than VNS_4 for instances with 50 and 100 jobs. However, no statistical differences were found to distinguish between AGA and HGMEDA.
- HGMEDA is the best algorithm for solving instances of 200 jobs, followed by AGA, with VNS_4 in third place.

Table 4.1: ARPD results for the Taillard's benchmark instances with the configurations 20×5, 20×10, 20×20, 50×5.

Instance	Best	AGA		VNS_4		GMEDA		VNS		HGMEDA	
20×05	14033	**0.00**	*	**0.00**	*	0.18	*	**0.00**	*	**0.00**	*
	15151	**0.00**	*	**0.00**	*	0.48		**0.00**	*	**0.00**	*
	13301	**0.00**	*	**0.00**	*	0.50	*	**0.00**	*	**0.00**	*
	15447	**0.00**	*	**0.00**	*	0.43	*	**0.00**	*	**0.00**	*
	13529	**0.00**	*	**0.00**	*	0.21	*	**0.00**	*	**0.00**	*
	13123	**0.00**	*	**0.00**	*	0.08	*	**0.00**	*	**0.00**	*
	13548	**0.00**	*	**0.00**	*	0.79		**0.00**	*	**0.00**	*
	13948	**0.00**	*	**0.00**	*	0.18	*	**0.00**	*	**0.00**	*
	14295	**0.00**	*	**0.00**	*	0.18	*	**0.00**	*	**0.00**	*
	12943	**0.00**	*	**0.00**	*	0.46	*	**0.00**	*	**0.00**	*
20×10	20911	**0.00**	*	**0.00**	*	0.45	*	**0.00**	*	**0.00**	*
	22440	**0.00**	*	**0.00**	*	0.54	*	**0.00**	*	**0.00**	*
	19833	**0.00**	*	**0.00**	*	0.31	*	**0.00**	*	**0.00**	*
	18710	**0.00**	*	**0.00**	*	0.75		**0.00**	*	**0.00**	*
	18641	**0.00**	*	**0.00**	*	0.35		**0.00**	*	**0.00**	*
	19245	**0.00**	*	**0.00**	*	0.77		**0.00**	*	**0.00**	*
	18363	**0.00**	*	**0.00**	*	0.48	*	**0.00**	*	**0.00**	*
	20241	**0.00**	*	**0.00**	*	0.47	*	**0.00**	*	**0.00**	*
	20330	**0.00**	*	**0.00**	*	0.27	*	**0.00**	*	**0.00**	*
	21320	**0.00**	*	**0.00**	*	0.24	*	**0.00**	*	**0.00**	*
20×20	33623	**0.00**	*	**0.00**	*	0.65	*	**0.00**	*	**0.00**	*
	31587	**0.00**	*	**0.00**	*	0.29	*	**0.00**	*	**0.00**	*
	33920	**0.00**	*	**0.00**	*	0.04	*	**0.00**	*	**0.00**	*
	31661	**0.00**	*	**0.00**	*	0.28	*	**0.00**	*	**0.00**	*
	34557	**0.00**	*	**0.00**	*	0.26		**0.00**	*	**0.00**	*
	32564	**0.00**	*	**0.00**	*	0.30	*	**0.00**	*	**0.00**	*
	32922	**0.00**	*	**0.00**	*	0.61		**0.00**	*	**0.00**	*
	32412	**0.00**	*	**0.00**	*	0.52		**0.00**	*	**0.00**	*
	33600	**0.00**	*	**0.00**	*	0.56	*	**0.00**	*	**0.00**	*
	32262	**0.00**	*	**0.00**	*	0.41	*	**0.00**	*	**0.00**	*
50×5	64803	**0.05**	*	0.78		0.79		0.16		0.12	*
	68062	**0.06**	*	0.88		0.94		0.13		0.12	
	63162	**0.19**	*	1.21		1.34		0.35		0.38	
	68226	**0.17**		1.12		1.27		0.28		0.22	*
	69392	**0.09**	*	0.87		0.89		0.15	*	0.15	
	66841	**0.10**	*	0.80		0.82		0.23	*	0.18	*
	66258	**0.03**		0.74		0.95		0.11	*	0.07	*
	64359	**0.05**		0.89		0.97		0.22		0.23	*
	62981	**0.09**	*	0.83		0.81		0.14	*	0.14	*
	68898	**0.07**	*	1.05		0.93		0.23	*	0.22	

- Finally, AGA and VNS_4 outperform HGMEDA for instances of size 500. However, the test does not determine which of them is better.

To demonstrate that the contribution of GMEDA is essential for an efficient performance of the hybrid approach, VNS was also evaluated independently in this experiment. The obtained results prove that, when run separately, each stage of the hybrid approach, GMEDA and VNS, does not perform as well as when they are run combined.

Table 4.2: ARPD results for the Taillard's benchmark instances with the configurations 50×10, 50×20, 100×5, 100×10.

Instance	Best	AGA	VNS_4	GMEDA	VNS	HGMEDA
50×10	87207	**0.33**	1.12	2.10	0.42	0.38 *
	82820	**0.22** *	1.09	2.45	0.57	0.60 *
	79987	**0.23** *	1.07	1.84	0.35 *	0.36
	86581	**0.17** *	0.90	1.83	0.28	0.32
	86450	**0.14** *	0.90	2.01	0.42	0.38
	86637	**0.13** *	0.77	1.55	0.29	0.29
	88866	**0.25** *	0.89	1.97	0.51	0.48
	86824	**0.19** *	0.94	2.03	0.33	0.36
	85526	**0.29**	1.11	2.10	0.49	0.42 *
	88077	**0.09** *	0.76	2.00	0.45	0.45
50×20	125831	**0.10** *	0.65	1.76	0.30	0.39 *
	119259	**0.04** *	0.51	1.58	0.32 *	0.22 *
	116459	**0.19** *	0.73	2.24	0.56	0.44 *
	120712	**0.22** *	0.61	1.92	0.40	0.34
	118184	**0.40**	0.86	2.30	0.50	0.52 *
	120703	**0.19** *	0.62	1.78	0.42	0.35
	122962	**0.38**	0.71	2.10	0.47	0.47 *
	122489	**0.16**	0.75	2.24	0.45	0.55 *
	121872	**0.17**	0.76	1.79	0.40	0.37 *
	124064	**0.23** *	0.90	1.95	0.45	0.42
100×05	*253713*	0.25	1.21	0.82	0.39	**0.19** *
	242777	**0.22**	1.71	1.00	0.38	0.27 *
	238180	**0.18** *	1.45	0.80	0.27	0.22
	227889	**0.17** *	1.29	0.78	0.23	0.20
	240589	**0.21**	1.29	0.80	0.31	0.23 *
	232936	0.22	1.41	0.80	0.31	**0.17** *
	240669	**0.15** *	1.34	1.00	0.34	0.34
	231428	**0.13**	1.63	0.90	0.25	0.19 *
	248481	0.23	1.48	0.87	0.29	**0.20** *
	243360	**0.15** *	1.40	0.96	0.29	0.24
100×10	*299431*	**0.32**	1.52	1.69	0.51	0.33 *
	274593	**0.59**	1.57	2.07	0.87	0.68 *
	288630	**0.34** *	1.53	1.71	0.59	0.35
	302105	**0.32**	1.60	1.89	0.56	0.35 *
	285340	0.39	1.42	1.73	0.58	**0.32** *
	270817	**0.30**	1.64	1.70	0.62	0.33 *
	280649	**0.23**	1.44	1.51	0.51	0.26 *
	291665	**0.35**	1.62	1.88	0.74	0.46 *
	302624	**0.36** *	1.46	1.76	0.64	0.41
	292230	**0.30** *	1.65	1.50	0.52	0.32

4.5.2.3 Taillard's extended benchmark results

From the previous experiments, it can be observed that HGMEDA performs better as the size of the problem enlarges. However, for problems of size 500, HGMEDA does not work as expected. In order to analyze this behaviour, we decided to create new instances to fill the gap between 200 and 500. 100 new instances were proposed with the following configurations: 250×10, 250×20, 300×10, 300×20, 350×10, 350×20, 400×10, 400×20, 450×10 and 450×20 (10

Table 4.3: ARPD results for the Taillard's benchmark instances with the configurations 100×20, 200×10, 200×20 and 500×20.

Instance	Best	AGA	VNS_4	GMEDA	VNS	HGMEDA
100×20	*367267*	0.57	1.47	2.03	0.69	**0.44** *
	374032	**0.31** *	1.19	1.80	0.56	0.34
	371417	0.47	1.31	1.93	0.62	**0.36** *
	373822	0.53 *	1.30	1.86	0.67	**0.38**
	370459	0.42	1.20	1.77	0.56 *	**0.32**
	372768	0.51	1.46	2.17	0.79	**0.41** *
	374483	0.42	1.54	1.90	0.75	**0.35** *
	385456	0.46	1.41	1.96	0.66	**0.43** *
	376063	0.43 *	1.32	1.82	0.66	**0.33**
	379899	**0.48**	1.29	2.05	0.64	0.49 *
200×10	*1047662*	0.48	1.25	1.19	0.89	**0.17** *
	1036042	0.91	1.52	1.46	1.04	**0.30** *
	1047571	0.48	1.44	1.12	0.79	**0.14** *
	1032095	0.52	1.40	1.13	0.65	**0.21** *
	1037053	0.62	1.29	1.32	0.87	**0.13** *
	1006650	0.50	1.36	1.39	0.98	**0.19** *
	1053390	0.89	1.60	1.17	1.11	**0.18** *
	1046246	0.50	1.38	1.26	0.87	**0.13** *
	1025145	0.63	1.43	1.11	0.98	**0.10** *
	1031176	0.78	1.67	1.29	1.09	**0.20** *
200×20	*1226879*	0.63	1.35	1.59	1.11	**0.26** *
	1241811	0.86	1.46	1.45	1.20	**0.33** *
	1266153	0.84	1.41	1.33	1.00	**0.24** *
	1237053	0.96	1.55	1.43	1.46	**0.29** *
	1223551	0.84	1.51	1.64	1.15	**0.25** *
	1225254	1.01	1.52	1.52	1.29	**0.29** *
	1241847	0.64	1.26	1.27	1.07	**0.25** *
	1240820	1.10	1.57	1.57	1.17	**0.36** *
	1229066	1.12	1.58	1.48	1.25	**0.27** *
	1247156	0.92	1.38	1.44	1.14	**0.28** *
500×20	6708053	**0.11** *	0.35	8.90	2.09	2.02
	6829668	**0.25** *	0.38	8.58	1.85	1.94
	6747387	**0.24** *	0.41	8.46	2.04	2.04
	6787054	**0.26** *	0.45	8.75	1.96	1.89
	6755257	**0.39**	0.41 *	8.72	2.01	1.92
	6751496	**0.19** *	0.42	8.58	2.15	2.13
	6708860	**0.27** *	0.45	9.15	2.06	2.05
	6769821	**0.31** *	0.58	8.62	2.24	2.09
	6720474	**0.15** *	0.46	8.69	1.99	1.91
	6767645	**0.19** *	0.44	8.51	2.12	2.00

instances for each configuration)[4]. The processing times of the jobs were generated uniformly at random (integer values between 1 and 100 as reported by Taillard [168]).

We executed 20 independent runs. θ_{upper} and the maximum number of evaluations for the stopping criterion were calculated following the same pro-

[4] Both the Taillard's benchmark and the extended benchmark can be obtained from `http://www.sc.ehu.es/ccwbayes/members/jceberio/GMEDA.html`.

cedure explained for Taillard's benchmark (see appendix 4.7). Results can be found in Table 4.4. Results in bold denote the algorithm that obtained the lowest relative error among the compared approaches. The results marked as (*) denote that the optimum or best known value has been achieved at least once among the performed repetitions.

According to the results (see Table 4.4), HGMEDA obtains the best TFT solution for 67 instances and AGA for 33. On the other hand, VNS_4 was not successful in any of the cases. Regarding the ARPD, HGMEDA is the best in 61 instances out of 100 and AGA in 39.

In order to state whether there exist statistical differences among the algorithms, we applied the same statistical test as previously, obtaining:

- HGMEDA is the best algorithm for solving instances of 250, 300 and 350 jobs. Followed by AGA, with VNS_4 in third place.
- AGA and HGMEDA outperform VNS_4 for instances of 400×10. However, no statistical differences were found to distinguish between AGA and HGMEDA.
- For configurations 400×20 and 450×10, AGA is statistically the best, followed by VNS_4, and HGMEDA in third position.
- Finally, AGA and VNS_4 outperform HGMEDA for instances of 450×20. However, the test does not determine which of them is better.

4.5.3 Additional experimentation

As previously mentioned, in this work we considered optimizing the PFSP with respect to the TFT due to its complexity and relevance for the current production environments. However, taking into account that the makespan is also an interesting criterion when optimizing the PFSP, we decided to carry out some additional executions with GMEDA and HGMEDA for the Taillard's PFSP instances with the makespan criterion. The results were summarized in a short report and uploaded to the website as additional material[5].

Considering that the proposed algorithm is more complex than its competitors, we carried out some additional experiments using the execution time as stopping criterion. A report with the results and some conclusions was uploaded to the website as additional material.

4.6 Discussion

In the previous section, we saw that HGMEDA is a strong alternative to optimize the PFSP. However, the unexpected performance observed for the largest instances deserves additional study. In this section we propose methods to improve the efficiency of HGMEDA.

[5] `http://www.sc.ehu.es/ccwbayes/members/jceberio/GMEDA.html`

Table 4.4: ARPD results for the Taillard's extended benchmark instances.

Instance	Best	AGA	VNS_4	HGMEDA	Instance	Best	AGA	VNS_4	HGMEDA
250×10	1566623	0.65	1.53	**0.15** *	350×20	3462591	1.03	1.11	**0.23** *
	1589117	0.89	1.56	**0.25** *		3472888	0.70	0.97	**0.41** *
	1633818	0.64	1.48	**0.22** *		3448696	0.79	0.98	**0.20** *
	1603824	1.03	1.57	**0.23** *		3487674	0.34	0.85	**0.33** *
	1652808	0.60	1.26	**0.19** *		3422898	0.85	1.11	**0.36** *
	1568866	0.38	0.65	**0.14** *		3451601	0.71	1.10	**0.39** *
	1592812	0.86	1.43	**0.16** *		3482315	0.52	0.92	**0.31** *
	1552918	0.66	1.46	**0.17** *		3448931	0.93	1.25	**0.41** *
	1594956	0.76	1.30	**0.13** *		3471376	1.04	1.12	**0.34** *
	1592563	0.91	1.41	**0.19** *		3432934	1.00	1.29	**0.35** *
250×20	1847471	0.71	1.40	**0.21** *	400×10	3933307	**0.18** *	0.51	0.81
	1892483	1.01	1.38	**0.17** *		3910048	**0.18**	0.71	0.36 *
	1879796	0.61	1.13	**0.17** *		3934557	**0.10** *	0.52	0.53
	1852134	0.90	1.32	**0.18** *		3875265	0.62	1.23	**0.36** *
	1836747	0.76	1.37	**0.28** *		3932228	**0.15**	0.34	0.43 *
	1859562	0.86	1.27	**0.21** *		3913491	**0.18** *	0.42	0.98
	1876186	0.99	1.38	**0.23** *		3865710	**0.19**	0.47	0.47 *
	1850963	0.87	1.46	**0.27** *		3918728	**0.34**	0.73	0.80 *
	1859994	0.98	1.41	**0.18** *		3903109	**0.38**	0.69	0.62 *
	1837633	0.89	1.25	**0.22** *		3934891	**0.12**	0.44	0.56 *
300×10	2241165	0.73	1.32	**0.09** *	400×20	4465942	**0.34** *	0.48	1.11
	2221209	0.81	1.47	**0.17** *		4442588	**0.26** *	0.57	1.36
	2239369	0.79	1.34	**0.16** *		4419467	**0.25** *	0.49	0.97
	2216046	0.57	1.05	**0.19** *		4450298	**0.42** *	0.67	1.32
	2255074	0.75	1.17	**0.13** *		4380600	**0.32** *	0.66	1.29
	2259913	0.76	1.31	**0.15** *		4460906	**0.23** *	0.50	1.02
	2244410	0.77	1.36	**0.18** *		4472922	**0.50** *	0.89	1.20
	2203072	0.84	1.17	**0.21** *		4438057	**0.16** *	0.49	1.19
	2245325	0.83	1.22	**0.21** *		4419300	**0.15** *	0.52	1.20
	2239058	0.61	1.23	**0.08** *		4402417	**0.19** *	0.48	1.02
300×20	2586158	1.01	1.39	**0.27** *	450×10	4870929	**0.26** *	0.70	1.67
	2569577	0.91	1.21	**0.29** *		4884041	**0.16** *	0.48	1.75
	2588574	0.96	1.39	**0.23** *		4930649	**0.19** *	0.59	0.98
	2584806	0.73	1.23	**0.15** *		4933807	**0.02** *	0.06	1.48
	2601077	0.91	1.28	**0.20** *		4934094	**0.12** *	0.27	1.66
	2605560	1.06	1.47	**0.32** *		4922342	**0.09** *	0.51	1.37
	2593450	0.88	1.10	**0.21** *		4931140	**0.09** *	0.50	1.43
	2556756	0.99	1.55	**0.35** *		4933524	**0.11** *	0.55	1.37
	2583311	1.13	1.32	**0.27** *		4906973	**0.13** *	0.66	1.39
	2625824	0.75	1.03	**0.24** *		4976526	**0.11** *	0.57	1.63
350×10	3051090	0.69	1.28	**0.15** *	450×20	5532585	**0.30** *	0.39	1.78
	2996234	0.56	0.86	**0.11** *		5545906	**0.33** *	0.62	1.76
	2977597	1.26	1.34	**0.24** *		5584602	**0.28** *	0.41	1.36
	3026374	0.87	1.37	**0.18** *		5588641	**0.20** *	0.22	1.61
	3031433	0.84	1.49	**0.16** *		5556343	**0.26** *	0.52	1.73
	3023478	0.72	1.32	**0.17** *		5507299	**0.26** *	0.73	2.02
	3037375	0.68	1.20	**0.14** *		5534081	**0.41** *	0.60	1.85
	3078709	0.70	0.88	**0.14** *		5580460	**0.20** *	0.43	1.60
	3029894	0.82	1.35	**0.17** *		5558982	**0.37** *	0.30	1.49
	3014352	0.71	1.26	**0.21** *		5559057	**0.15** *	0.06	1.28

4.6.1 Why does the performance of HGMEDA decrease for the largest instances?

In Table 4.4, we can see that the fitness difference between HGMEDA and AGA increases from configuration 400×20 onwards, which favors AGA. Since HGMEDA is based on GMEDA, we analyze two aspects of HGMEDA: the performance of GMEDA against AGA, and the contribution of each stage of the hybrid approach to the optimization.

To study the performance of GMEDA versus AGA, in Fig. 4.5 the average relative percentage deviation (ARPD) between GMEDA and AGA is introduced. This time, for each instance i within the set of instances of a given configuration, we consider the best result of the 20 runs performed by GMEDA and AGA, called $GM\text{-}EDA_i^{best}$ and AGA_i^{best} respectively. Then, we calculate the ARPD for each $n \times m$ configuration (note that each configuration contains a set of 10 instances):

$$ARPD = \left(\sum_{i=1}^{10} \frac{(GMEDA_i^{best} - AGA_i^{best}) \times 100}{AGA_i^{best}} \right) / 10$$

According to the results, GMEDA shows a regular ARPD below 0.02 with

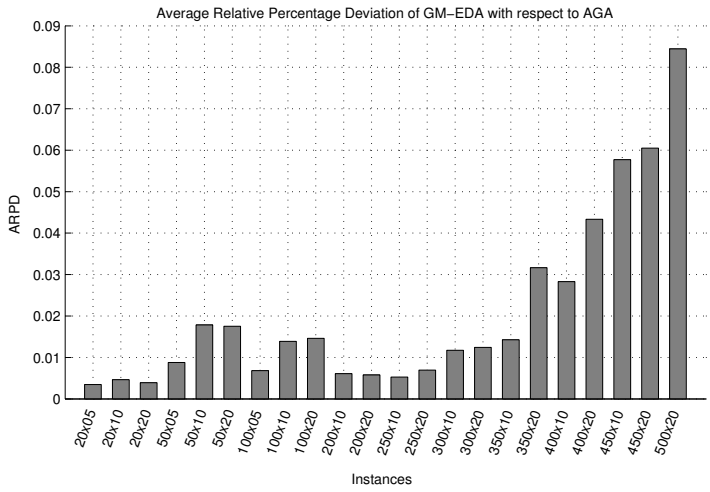


Fig. 4.5: ARPD between the best results obtained by GMEDA and AGA.

respect to AGA up to configuration 350×10. The low deviation described for such instances coincides with the good performance of HGMEDA. However, the results for configurations from 400×20 onwards show a dramatical increase of ARPD. In fact, the largest differences in Fig. 4.5 match the poor results seen for HGMEDA for such instances.

The results observed suggest that HGMEDA relies its performance to the ability of GMEDA to find promising solutions. As HGMEDA is a two-stage algorithm, in the following lines we propose a study of the contribution ratio of each stage on the final solution. For this analysis, we consider three particular solutions from HGMEDA: the best individual of the first population of GMEDA (Sol_{init}), the solution returned by GMEDA when the first stage finishes (Sol_{GMEDA}), and the final solution returned by the VNS in the second stage of the HGMEDA (Sol_{final}). The contribution ratios (CR) are computed as $CR_{GMEDA} = (Sol_{init} - Sol_{GMEDA})/Sol_{final}$ and $CR_{VNS} = (Sol_{GMEDA} - Sol_{final})/Sol_{final}$. The ratios were calculated averaging the results obtained from 20 independent runs over the first instance of each configuration instance set. The results are shown in Fig. 4.6. Note that the ratio axis is only shown from 0.5 to 1. The plotted results support the con-

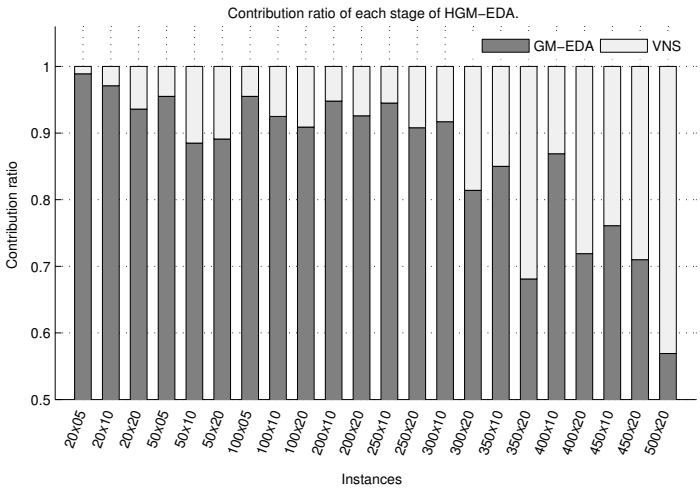


Fig. 4.6: Contribution ratios performed by each of the stages of HGMEDA.

clusion that we had come to previously: GMEDA is the stage of the algorithm that conditions the performance of HGMEDA. While improvement ratios for instances between 20 and 250 jobs show a regular contribution around 0.9 for GMEDA, the results for the instances from 400×20 onwards are significantly lower.

4.6.2 Why does GMEDA become less competitive than AGA for the largest instances?

Since the GM model characterises the performance of GMEDA, we analyze the convergence of the probabilistic model when optimizing the largest instances.

As aforementioned in Section 4.3, the vector $\boldsymbol{\theta}$ determines the exploration or intensification behaviour of the GM model when optimizing. Therefore, in this new experiment, we examine the average value of the dispersion parameters $\boldsymbol{\theta}$ obtained at the end of the executions. In this case, the first instance of each configuration $n \times m$ was studied. For comparison purposes, the results are normalized to the range [0,1] by dividing the average $\boldsymbol{\theta}$ calculated for each instance, by the θ_{upper} set in each case.

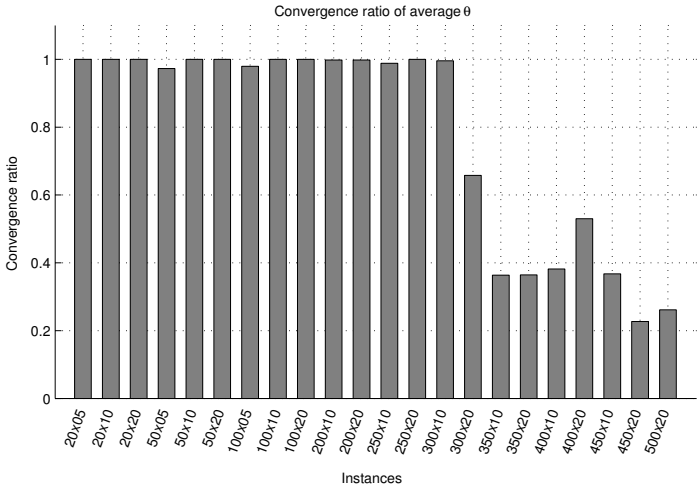


Fig. 4.7: Convergence ratio of the average $\boldsymbol{\theta}$ values to the θ_{upper} at the end of the execution.

According to the results shown in Fig. 4.7, in instances up to 300×10, the average $\boldsymbol{\theta}$ converged to the θ_{upper} set. However, in larger instances, the average $\boldsymbol{\theta}$ values do not achieve values higher than 0.7 in each case. This means that the convergence of the model did not even reach 70% of the upper bound fixed to each configuration type.

This experiment suggests that the GM model learnt was in an exploration phase (taking into account Fig. 4.4) when stopped, instead of intensifying around the promising area.

Alternatively, examining the initialization of AGA in detail, we realized that the constructive heuristic employed, LR(n/m) (also in VNS_4), has a decisive contribution when solving large scale instances. Some preliminary experiments showed that the performance of AGA and VNS_4 was significantly worse when LR(n/m) was removed from the procedure.

In order to state what the real advantage of including LR(n/m) is, we compare the intermediate result calculated by LR(n/m) as the first stage of AGA and that of GMEDA. We define the ARPD between LR and GMEDA

as:

$$ARPD = \left(\sum_{i=1}^{10} \frac{(LR_i - GMEDA_i)\times 100}{GMEDA_i}\right)/10$$

($GMEDA_i$ denotes the average of the fitness value obtained by the GMEDA in the 20 repetitions). The results are shown in Fig. 4.8. The values used to calculate the ARPD are the average values of 20 repetitions over the 10 instances of each configuration. Note that the instances with histograms above 0 denote that the solutions obtained by GMEDA are, on average, better than those achieved by LR(n/m) and the opposite when the results obtained are below 0.

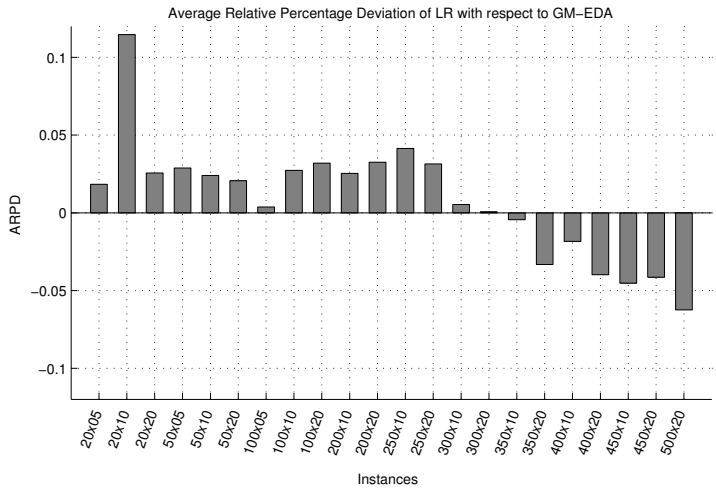


Fig. 4.8: ARPD of LR with respect to GMEDA performed at the first stage of HGMEDA.

The results report that the performance of GMEDA is better than that of LR(n/m) for instances up to 350×10. However, for larger instances, LR(n/m) is better. The results suggest that the poor performance seen for larger instances might also be motivated by the random initialization of the initial population of GMEDA.

4.6.3 Guided HGMEDA

Previous analysis suggests that the random initialization and the lack of enough evaluations could be the main reasons for the unexpected performance of HGMEDA in the largest instances (instances from 350×10 onwards). Inspired by the state-of-the-art algorithms that we compared it with, we pro-

pose a method for guiding the initialization of the first population, based on LR(n/m).

As with every constructive algorithm, LR(n/m) measures the preference to append a job to the current partial solution. For each non-scheduled job, the heuristic calculates an index value that considers the increase of the TFT when the non-scheduled job is appended to the partial solution, and the increase of the TFT to the solution by the averaged processing times of the remaining jobs. The job with the lowest index value is selected to be appended to the partial solution.

In this case, in order to guide the initialization of the first population, we propose a modification of the LR(n/m) to transform the constructive heuristic into a non-deterministic proposal. A non-deterministic proposal allows the presumable provision of a different seed individual at each repetition of the algorithm. The modification proposed replaces the top-ranking job selection by sampling a probability distribution inversely proportional to the index value of the jobs in the ranking.

In order to analyze the impact of the guided initialization and the premature stopping of GMEDA, we compare AGA, HGMEDA and Guided HGMEDA over the first instance of the configurations 400×20, 450×10, 450×20 and 500×20. In addition, the algorithms are run from 1 to 8 times the maximum number of evaluations specified in Appendix 4.7. The results are shown in Fig. 4.9.

It can be seen that Guided HGMEDA achieves better results than HGMEDA and outperforms the results of AGA when extending the number of evaluations employed. The successful results achieved by Guided HGMEDA against AGA and HGMEDA support the potential of the EDA introduced in this paper and confirm the determinant effect of implementing a guided initialization when dealing with large size instances.

4.7 Conclusions and future work

In this chapter we have presented a novel algorithm for dealing with permutation-based problems, called Generalized Mallows EDA. GMEDA introduces a distance-based exponential probability model. As this chapter is the first serious attempt to introduce an EDA of such characteristics, in order to demonstrate its efficiency it was tested on the well-known permutation flowshop scheduling problem.

In particular, a hybrid version of the GMEDA is proposed (GMEDA + VNS) in order to obtain state-of-the-art results.

The experiments carried out stated that the hybrid version of the GMEDA (HGMEDA) is the best performing algorithm for solving the PFSP-TFT. From 220 instances tested, HGMEDA obtained the best solution for 152 instances. The analysis of the results concluded that the success of HGMEDA is due to the ability of GMEDA to find optimal regions in the search space.

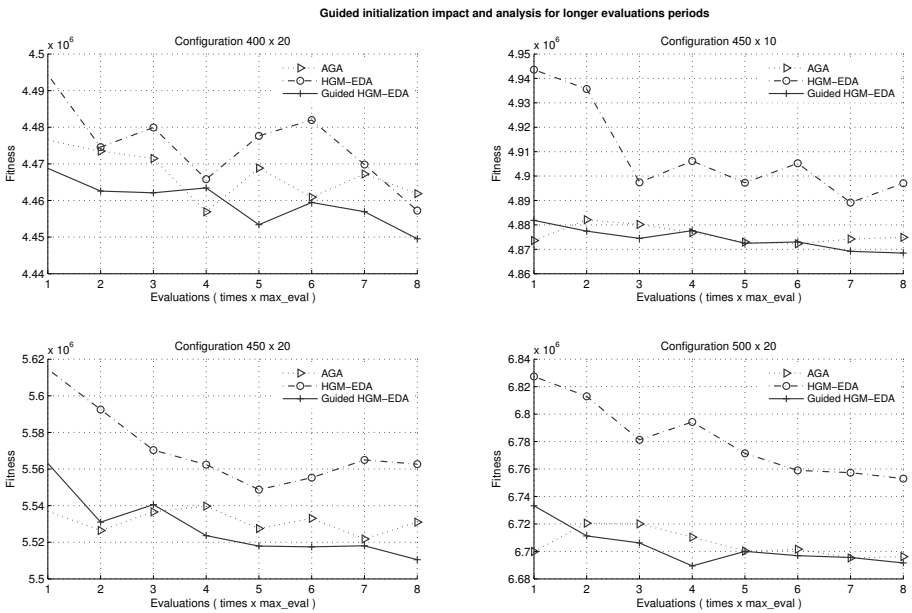


Fig. 4.9: Average fitness results of 10 repetitions of AGA, HGMEDA and Guided HGMEDA over the first instance of the configurations 400×20, 450×10, 450×20 and 500×20.

Furthermore, the experiments proved that employing a heuristic for initializing the algorithm is preferable than a random initialization, mainly when solving large size instances.

Additional experiments revealed that GMEDA and HGMEDA are the most time consuming algorithms among the studied approaches. However, further executions stated that HGMEDA is still very competitive compared with its competitors when they are executed with the same running time.

The incorporation of the Generalized Mallows model to the framework of EDAs leaves many trends that are worth extending in future work. In the following paragraphs we will briefly mention some of them.

As introduced previously, the GM model is defined by two parameters: the consensus permutation σ_0 and the vector of spread parameters $\boldsymbol{\theta}$. In Section 4.2, we highlighted the difficulty of calculating these parameters exactly, and we proposed approximation methods that permit us to use the GM model in EDAs. However, we believe that there is still room for improvement. More advanced methods for calculating the σ_0 or self-adapting procedures for the estimation of $\boldsymbol{\theta}$ may lead GMEDA to improve its efficiency.

Another feature of the GM model that deserves a deeper study is that related to how the unimodal property of the model behaves when solving problems/instances with many global optima. In this sense, mixtures of Mallows models [129] might be considered in order to optimize different areas of the landscape separately.

In addition, in this proposal, Kendall's-τ distance was considered due to its decomposable properties, which allow to factorize the GM model, and thus, learn and sample the model efficiently. However, there exist many other distances on permutations that could be developed such as Cayley, Ulam or Hamming [51]. In fact, finding the best distance for approaching a certain permutation problem is challenging research.

The inclusion of the information of the instance into the GM model is another interesting issue. Due to the characteristics of the GM model, the inclusion procedure of external information into the model is relatively easy. However, extracting relevant information from a given instance is a very hard task.

Regarding the proposed algorithm, there are also other points that should be analyzed in order to obtain better results: the percentage of evaluations used in each stage of the HGMEDA, the effect of developing more elaborate population initialization methods, the shake method used in the restart scheme, or the selection of the neighbourhoods for the proposed VNS algorithm as a part of HGMEDA.

Finally, with the aim of ratifying these initial results, we consider it interesting to test the Generalized Mallows EDA on a wider set of permutation-based problems, such as the quadratic assignment problem or the linear ordering problem.

Appendix: Execution parameters

Tables 4.5 and 4.6 show the maximum number of evaluations and θ_{upper} values used in the experimentation for the Taillard's benchmark and the Taillard's extended benchmarks respectively. θ_{upper} values have been obtained testing different values of θ, starting at θ_{min} and ending at θ_{max}, in steps of 0.1. Note that only the first instance of each configuration was used to look for the most adequate parameter values.

Table 4.5: Spread parameters and maximum number of evaluations for Taillard's benchmark.

Instance	θ_{min}	θ_{max}	θ_{upper}	Evaluations
20×05	1.0	3.0	1.5	182224100
20×10	1.0	3.0	1.4	224784800
20×20	1.0	3.0	1.4	256896400
50×05	2.5	5.5	3.7	220712150
50×10	2.5	5.5	2.8	256208100
50×20	2.5	5.5	3.0	275954150
100×05	3.5	6.0	4.9	235879800
100×10	3.5	6.0	3.7	266211000
100×20	3.5	6.0	4.7	283040000
200×10	4.0	6.0	5.3	272515500
200×20	4.0	6.0	5.5	287728850
500×20	4.0	7.0	4.4	260316750

Table 4.6: Spread parameters and maximum number of evaluations for the Taillard's extended benchmark.

Instance	θ_{min}	θ_{max}	θ_{upper}	Evaluations
250×10	4.0	7.0	5.2	267779100
250×20	4.0	7.0	4.4	284574350
300×10	4.0	7.0	4.6	273847500
300×20	4.0	7.0	5.2	284672900
350×10	4.0	7.0	6.6	278369000
350×20	4.0	7.0	7.0	286225300
400×10	4.0	7.0	5.5	275491800
400×20	4.0	7.0	5.0	283913500
450×10	4.0	7.0	4.0	277455350
450×20	4.0	7.0	6.7	269271450

5

A review of distances for the Mallows and Generalized Mallows EDAs

5.1 Introduction

In the previous chapter we presented the first serious attempt to introduce a probability model for permutation domains in the context of EDAs. Particularly, we introduced the Generalized Mallows (GM) model achieving successful results on the permutation flowshop scheduling problem [26]. The GM model, as well as its simpler version, the Mallows model, are unimodal and assign every permutation in the search space a probability that depends on a distance function on permutations. Commonly, the Kendall's-τ has been the distance of choice under the Mallows and GM models. However, as demonstrated in [27], other distances such as Cayley [79] and Ulam [80] could be considered in the development of Mallows and Generalized Mallows EDAs (MaEDAs and GMEDAs). Preliminary experiments with the three distances in [27] reported evidence of large performance variations depending on the problem being dealt with.

In the same research line, in this chapter, we review the Kendall's-τ, Cayley and Ulam distances, and show that the behaviour of EDAs based on the Mallows and Generalized Mallows models strongly depends on the distance used. In fact, we see that, for most of the tested problems, there exists one distance that clearly outperforms the rest. In view of their influence on the performance of MaEDAs and GMEDAs, we find it interesting to study their characteristics and behaviour in order to predict the most suitable distance for any given problem.

Trying to seek the underlying reasons for the different behaviours of the distances in each problem, we focus our attention on the neighbourhood of solutions that each distance naturally induces. Particularly, we see that the sets of solutions at distance one of the Cayley and Ulam distances correspond to the *interchange* and *insert* neighbourhoods respectively [156]. With regard to the Kendall's-τ distance, though not so explicitly, the set at distance one is related to the *swap* neighbourhood [156].

Taking into account this correspondence between distances and neighbourhoods, in this chapter we investigate whether the research carried out in the field of local search can be applied to extract information in relation to the prediction of the most suitable distance for each problem type.

In this sense, we first calculated the correlation coefficients between the performance of MaEDAs and GMEDAs under the Kendall's-τ, Cayley and Ulam distances, and the performance of multistart local search algorithms (MLSs) under the *swap*, *interchange* and *insert* neighbourhoods on a set of four well-known permutation-based problems. The study revealed that the performance of the EDAs is strongly correlated to that of MLSs for certain *distance-neighbourhood* pairs. Therefore, making use of these correspondences, we applied fitness landscape analysis techniques on the *swap*, *interchange* and *insert* neighbourhoods, in order to analyse in detail the reasons for the different behaviours that distances show from problem to problem. By means of *information analysis* techniques [183], we measured the shape of the landscapes generated by the neighbourhoods above. The experimental study showed that the neighbourhoods that best perform in MLSs, are those that produce the smoothest landscapes. Taking into account the relation between EDAs and MLSs, we make use of the tools and measures proposed for landscape analysis in order to choose the most suitable distance for a given problem.

The remainder of the chapter is organised as follows: in Section 5.2, a gentle introduction to the different distances on permutations is given. Next, in Section 5.3, we compare the behaviour of MaEDAs and GMEDAs under the Kendall's-τ, Cayley and Ulam distances on the benchmark problems used throughout the dissertation. Next, in Section 5.4, we evaluate the performance of the *swap*, *interchange* and *insert* neighbourhoods in MLS, studying the correlation with the results obtained by EDAs. Based on landscape analysis techniques, in Section 5.5 we study the shape of the landscapes generated by the neighbourhoods above. In Section 5.6, we introduce a discussion about the characteristics of the operators that influence the shape of the fitness landscapes generated by them. Finally, general conclusions and ideas for future work are summarised in Section 5.7.

5.2 Distances

In order to measure the distance between permutations, multiple metrics have been proposed in the literature [52]. However, only Kendall's-τ, Cayley and Ulam have been considered in the field of EDAs, and thus, our study will be restricted to these three distances. The Kendall's-τ, Cayley and Ulam distances have the *right invariant* property since $D(\sigma, \pi) = D(\sigma\gamma, \pi\gamma)$ for every permutation γ in the set of permutations of n indices, $\gamma \in \mathbb{S}_n$, where $\sigma\gamma$ stands for the composition of the permutations σ and γ, and is defined as $\sigma\gamma(i) = ((\sigma \circ \gamma)(i)) = \sigma(\gamma(i))$. Thus, by right invariance, taking $\gamma = \pi^{-1}$,

$D(\sigma, \pi)$ can be simplified as $D(\sigma\pi^{-1}, e)$ (also denoted as $D(\sigma\pi^{-1})$), where e stands for the identity permutation $e = 123\ldots n$.

A. Kendall's-τ

The Kendall's-τ distance, $D_\tau(\sigma, \pi)$, counts the minimum number of pairwise disagreements between σ and π. An equivalent definition states that $D_\tau(\sigma, \pi)$ measures the number of adjacent transpositions needed to convert σ^{-1} into π^{-1}. Given two permutations $\sigma = 21453$ and $\pi = 12345$, then $\sigma^{-1} = 21534$ and $\pi^{-1} = 12345$. A minimum sequence of adjacent transpositions (in bold) needed to convert σ^{-1} into π^{-1} is as follows:

$$21534 \rightarrow 21\mathbf{35}4 \rightarrow 213\mathbf{45} \rightarrow \mathbf{12}345$$

In order to use a distance in the GM model, it needs to be decomposable as a sum of $n-1$ terms. In this sense, the Kendall's-τ distance $D_\tau(\sigma)$ is decomposed as a vector $V(\sigma) = (V_1(\sigma), \ldots, V_{n-1}(\sigma))$ such that $D_\tau(\sigma) = \sum_{j=1}^{n-1} V_j(\sigma)$. Each term $V_j(\sigma)$ equals to the number of items smaller than $\sigma(j)$ on the last $n-j+1$ positions of the permutation.

B. Cayley

The Cayley distance, $D_c(\sigma, \pi)$, counts the minimum number of swaps (not necessary adjacent) that have to be performed to transform σ into π. When the reference permutation is the identity, $D_c(\sigma, e)$, the number of swaps equals to n minus the number of *cycles* of σ. A cycle is understood as a closed walk of elements in the permutation such that for every $1 \leq i, j \leq n$ that $\sigma(i) = j$, then i and j are in the same cycle. For instance, $\sigma = 156324$ is written in cycle notation as $(1)(2,5)(3,6,4)$. Note that a minimal sequence of swaps to convert σ into e will always imply swapping indices in the same cycle. A possible minimal sequence from σ to e is as follows:

$$156324 \rightarrow 1\mathbf{2}63\mathbf{5}4 \rightarrow 12\mathbf{36}54 \rightarrow 1234\mathbf{5}\mathbf{6}$$

The Cayley distance $D_c(\sigma)$ can be also decomposed into a vector $X(\sigma) = (X_1(\sigma), \ldots, X_{n-1}(\sigma))$ of $n-1$ terms such that $D_c(\sigma) = \sum_{j=1}^{n-1} X_j(\sigma)$. Each term $X_j(\sigma)$ of the decomposition vector is binary and $X_j(\sigma) = 0$ iff j is the largest item in its cycle in σ.

C. Ulam

The Ulam distance, $D_u(\sigma, \pi)$, equals the minimum number of insert operations that have to be performed in order to transform σ into π. An equivalent definition when the reference permutation is the identity, states that, $D_u(\sigma, e)$ equals n minus the length of the longest increasing subsequence of σ.

A classical example to illustrate the Ulam distance $D_u(\sigma, \pi)$ [52] considers a shell of books in the order specified by σ. The objective is to order the books

as specified by π with the minimum number of moves, where a move consists of taking a book and inserting it into another position (insert movement). The minimum number of movements is exactly $D_u(\sigma, \pi)$. For $\sigma = 43152$, a sequence of possible insert movements (in bold) to convert σ to the identity permutation is as follows:

$$43152 \rightarrow 31\mathbf{4}52 \rightarrow 1\mathbf{3}452 \rightarrow 1\mathbf{2}345$$

Inversely to the Kendall's-τ and Cayley distances, the decomposition of the Ulam distances is yet to be discovered, and therefore, the GM model can only be extended to the first two distances.

5.2.1 Learning and sampling Mallows and Generalized Mallows models

In order to introduce the Mallows and GM models under the Kendall's-τ, Cayley and Ulam distances in the framework of EDAs, it is necessary to define efficient learning and sampling methods for each model-distance. As regards the learning, we have chosen a process that consists of two steps (similar for all the models-distances): first, given a sample of permutations, we calculate the central permutation σ_0 as the *set median permutation*, which is the permutation in the sample that minimizes the sum of the distances to the rest of the permutations in the sample. Secondly, the spread parameter θ for the Mallows model (or $\boldsymbol{\theta}$ for the GM model) is estimated. The expression for this parameter is obtained by equaling to zero the derivative of the log-likelihood function. Although this expression differs depending on the distance, in most of the cases numerical methods, such as Newton-Raphson, can be used to obtain an approximate value. As regards the sampling of new solutions, each distance has a particular procedure, and thus, for specific expressions we refer the reader to [26, 79, 80].

5.3 Evaluating the performance of the distances on Mallows and Generalized Mallows EDAs

In this section we analyse the influence that the Kendall's-τ, Cayley and Ulam distances have on MaEDAs and GMEDAs in the benchmark problems considered in this dissertation, i.e. TSP, LOP, PFSP and QAP (in Table 5.1 a summary of the fitness functions is given).

5.3.1 Instances settings

In relation to the instances to be used in the experimentation, the existing benchmarks for the LOP, QAP, PFSP and TSP belong to different sources and, in view of the irregular variety of sizes and types of instances, these

Table 5.1: Summary of the functions associated to the LOP, QAP, PFSP and TSP.

Problem	max / min	$f(\sigma)$	Parameters	Refs.
TSP	min	$\sum_{i=2}^{n} d_{\sigma(i-1),\sigma(i)} + d_{\sigma(n),\sigma(1)}$	$D = [d_{i,j}]_{n\times n}$	[67]
LOP	max	$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} b_{\sigma(i),\sigma(j)}$	$B = [b_{i,j}]_{n\times n}$	[116, 31]
PFSP	min	$\sum_{i=1}^{n} c_{\sigma(i),m}$	$P = [p_{i,j}]_{m\times n}$	[9, 26]
QAP	min	$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j}h_{\sigma(i),\sigma(j)}$	$D = [d_{i,j}]_{n\times n}$, $H = [h_{k,l}]_{n\times n}$	[92]

benchmarks do not provide a suitable framework to test the scalability of MaEDAs and GMEDAs in this work. Due to this, we designed an artificial benchmark of 200 instances (50 instances of each problem) of sizes $\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ to be used throughout the chapter. Five instances of each size have been generated.

As regards the LOP, we follow the procedure proposed by [155] which chooses an instance from another benchmark (in our case LOLIB), and samples uniformly at random n^2 parameters from this, until the new instance is completed. In the case of the TSP and QAP, we follow a similar procedure by sampling instances from the TSPLIB [148] and Eric Taillard's QAP [169] benchmarks respectively. With respect to the PFSP, the instances have been generated by sampling the processing times of the jobs uniformly at random from the interval $[0, 100]$ (as reported by Taillard [168]). Note that in the PFSP the size of the instance is given by the number of jobs. The number of machines has been fixed to 10.

All the algorithms have been implemented in C++ programming language. The experimentation was conducted on a cluster of 20 nodes, each of them equipped with two Intel Xeon X5650 CPUs and 48GB of memory[1].

5.3.2 Experimental settings

We run five different EDAs: the Mallows EDAs under the Kendall's-τ (M_{ken}), Cayley (M_{cay}) and Ulam (M_{ula}) distances, and the Generalized Mallows EDAs under the Kendall-τ (GM_{ken}) and Cayley (GM_{cay}) distances. Recall that the use of Ulam cannot be extended to the GM model. We follow the parameter settings detailed in [27]:

- Population size: $10n$.

[1] Supplementary results, source codes, instances, and extended material of the experiments can be downloaded from `http://www.sc.ehu.es/ccwbayes/members/jceberio/COAP/Review_Distances.html`.

- Selected individuals: n.
- Selection type: truncation.
- Sampled individuals: $10n - 1$.
- Elitism criterion is followed.
- Stopping criterion: $1000n^2$ evaluations.
- Newton-Raphson: 100 iterations, and a minimum variation threshold of 0.001.

Alternatively, in order to enhance the exploration/exploitation abilities of the model, we set an adaptive interval of values for the θ parameters. The aim of this interval is to avoid extreme shapes of the model. When θ is near to 0, the Mallows and GM models are close to the uniform distribution. Inversely, when θ is higher than 10, the probability of the central permutation is close to 1 (for the considered sizes). In both cases, the expected improvement of the EDAs by sampling models of these characteristics is very small. Due to this, at the beginning of the process the interval of values θ is set to [0,10]. Every iteration at which the best solution is not improved, the lower bound is increased by 0.001. On the contrary, every time that a better solution is found the lower bound is decreased by 0.001. The upper bound remains fixed throughout the optimisation.

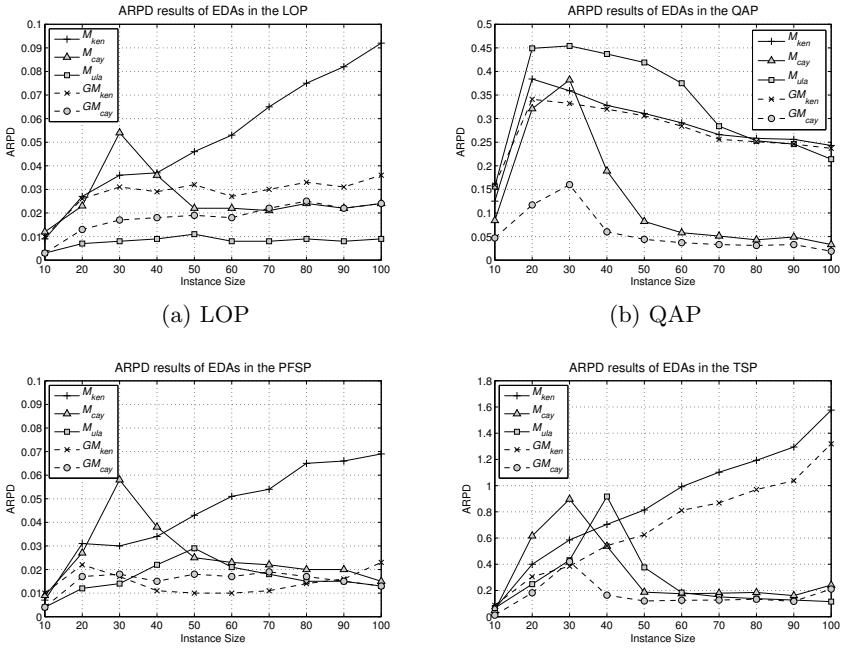
5.3.3 Results

Each *algorithm* - *instance* pair was run 20 times. As in previous cases, the performance measure employed in our study is the average relative percentage deviation (ARPD) [26]:

$$ARPD = \frac{|AvgRes - Best|}{Best}$$

where $AvgRes$ denotes the average result obtained throughout the 20 repetitions of the algorithm, and $Best$ stands for the best solution obtained throughout the whole experimental study by any of the EDAs. So, the lower the ARPD value, the better the performance. The ARPD results of the executions are introduced in Fig. 5.1. For illustrative purposes, ARPD results of equal size instances have been averaged.

The results in Fig. 5.1 show that the performance of the proposed five EDAs is influenced by the distance used to learn the model. Furthermore, it can be seen that, in most of the problems, there is a distance that outperforms the rest. In order to state whether there exist statistical differences among the algorithms in each problem, we applied four Friedman's tests, one for each problem, to the ARPD results obtained by M_{ken}, M_{cay}, M_{ula}, GM_{ken} and GM_{cay} on the 50 instances of each problem. A level $\alpha = 0.05$ of significance was set. The statistical test reported significant differences between the five EDAs, obtaining p-values below 0.001 in all the problems. Following the recommendation of Garcia et al. [62] for such cases, we applied the



(a) LOP (b) QAP

(c) PFSP (d) TSP

Fig. 5.1: ARPD results of the M_{ken}, M_{cay}, M_{ula}, GM_{ken} and GM_{cay} for the LOP, QAP, PFSP and TSP.

Shaffer's static procedure to carry out all the pairwise comparisons and determine which algorithm performs the best in each problem. Again, we fixed the significance level to $\alpha = 0.05$. The results of the test are summarised using critical difference diagrams in Fig. 5.2. Low ranks denote good performance, and horizontal lines indicate that the algorithms tied with the line do not perform statistically different. The statistical test states the following:

- In the LOP, the best performing algorithm is M_{ula}.
- As regards the QAP, M_{cay} and GM_{cay} outperform the rest of the EDAs, being GM_{cay} the best.
- For the PFSP, there is no proposal which is statistically superior. However, we see that M_{cay}, and especially M_{ken}, are significantly worse than the rest.
- For the TSP, GM_{cay} is statistically the best performing algorithm.

From the experiments we conclude that:

- The competitiveness of the five EDAs varies drastically depending on the problem at hand. An illustrative example is the case of M_{ula} in the LOP

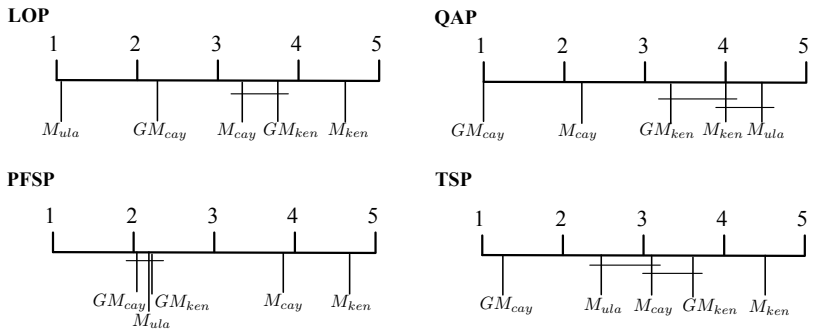


Fig. 5.2: Critical difference diagrams of the results obtained.

and the QAP, where in the former problem it is the best EDA, and in the latter it is one of the worst.
- GMEDAs statistically outperform MaEDAs under the same distance.
- In general, GM_{cay} is the most promising algorithm, obtaining competitive ranks (Fig. 5.2) in all the problems. On the contrary, M_{ken} is constantly the algorithm with the highest rank (poorest behaviour).

According to the results, the behaviour of the EDAs introduced in this chapter is clearly characterised by the probabilistic model used (Mallows or GM), and in particular, by the distance considered under the model. It means that, in order to solve a particular permutation problem, practitioners should carefully select the most suitable distance. Trying to understand the underlying reasons for the different performances, and taking as a basis the relation between the local structure defined by the distances and the neighbourhood systems, we will focus our attention on the framework of local search algorithms. Particularly, in the next section we intend to analyse whether the relation between the distances and neighbourhoods can be observed in terms of performance between EDAs and MLSs. This could permit us to apply fitness landscape techniques in order to determine, a priori, the most suitable distance to solve a given problem.

5.4 Evaluating the performance of the different neighbourhoods in MLSs

As previously mentioned, there exists a direct correspondence between the distances described in Section 5.2 and some of the most acknowledged neighbourhood systems for permutations. In the particular case of Cayley and Ulam, we can see that the sets of permutations at distance one correspond to the *interchange* and *insert* neighbourhood systems [156] respectively. Given a permutation σ of size n, the *interchange* neighbourhood considers as neighbours

all the solutions that can be obtained by performing any exchange of two items i and j. The *insert* neighbourhood considers as neighbours all the solutions that can be obtained by moving any item i to any position j.

In the case of the Kendall's-τ distance, though not so explicitly, the set of solutions at distance one is related to the *swap* neighbourhood. The Kendall's-τ does not count the distance as the minimum number of operations to convert π in σ. However, this distance can be expressed as the minimum number of adjacent transpositions needed to convert π^{-1} into σ^{-1}, and thus, although indirectly, it is related to the *swap* neighbourhood. This neighbourhood considers as neighbours those solutions that can be obtained performing an exchange of any two adjacent items.

In this section, first we evaluate the performance of the *swap*, *interchange* and *insert* neighbourhood systems in Multistart Local Search algorithms (MLS), and then we calculate the correlations between the results of EDAs and MLSs focusing on the correspondence between the distances and neighbourhoods.

5.4.1 Experimental settings

We run the *swap* (MLS_S), *interchange* (MLS_X) and *insert* (MLS_I) multistart local search algorithms on the benchmark of 200 instances used in the previous section. Regarding the neighbours selection, the greedy approach is followed at each step of the optimisation. The initial solutions at every restart are generated uniformly at random. Following the settings of the previous experiments, a maximum number of $1000n^2$ evaluations is considered as the stopping criterion.

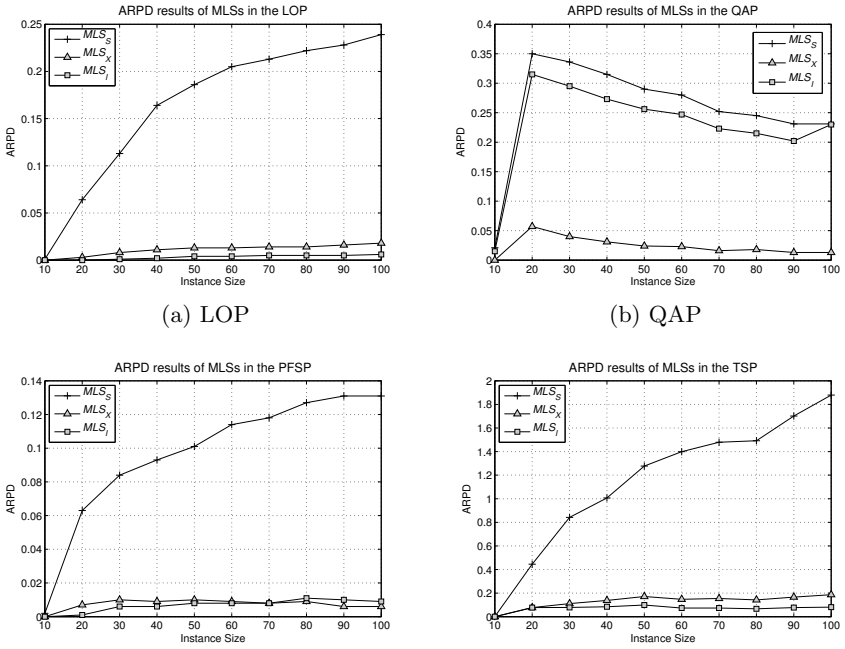
5.4.2 Results

We run each *algorithm-instance* pair 20 times. The ARPD results are presented in Fig. 5.3 [2]. For illustrative purposes, ARPD results of equal size instances have been averaged.

In order to state whether there exist statistical differences among MLS_S, MLS_X and MLS_I, we applied the Friedman's test to the ARPD results obtained by the three algorithms in each problem. The significance level is set to $\alpha = 0.05$. The test found significant differences among MLS_S, MLS_X and MLS_I in all the problems, reporting p-values below 0.001. Therefore, we applied the Shaffer's static procedure in order to perform pairwise comparisons with the same significance level. The post-hoc procedure confirmed significant differences in all the cases except between MLS_X and MLS_I in the PFSP. The statistical test states the following:

- In the LOP and the TSP, MLS_I outperforms the rest of the algorithms.

[2] The ARPD results have been calculated using the best known results obtained by the MLSs. Results from the previous section have not been used here.



(a) LOP (b) QAP

(c) PFSP (d) TSP

Fig. 5.3: ARPD results of MLS_S, MLS_X and MLS_I.

- In the QAP, MLS_X is the best algorithm.
- In the PFSP, there are no significant differences between MLS_I and MLS_X, but both are better than MLS_S.
- MLS_S is the worst algorithm.

5.4.3 Analysis of correlations

The results in Fig. 5.3 share certain similarities with the results obtained by the five EDAs in Section 5.3. In order to measure in detail the intensity of the relation between these EDAs and MLSs, we calculate the Pearson product-moment correlation coefficients between the ARPD results obtained by each algorithm, EDA and MLS, over the 200 tested instances. Results are described in Table 5.2. Results in bold highlight the highest correlation coefficient found for each EDA. The analysis confirms the following relations:

- M_{ken} and GM_{ken} correlate the most with MLS_S, obtaining coefficients near 1. It is worth mentioning that the coefficients are also remarkably high with respect to MLS_X.
- M_{cay} and GM_{cay}, correlate the most with MLS_X.

Table 5.2: Pearson product-moment correlation coefficients between the results obtained by the EDAs and the MLSs.

	MLS_S	MLS_X	MLS_I
M_{ken}	**0.975**	0.902	0.288
M_{cay}	0.439	**0.523**	0.290
M_{ula}	0.336	0.347	**0.772**
GM_{ken}	**0.955**	0.877	0.359
GM_{cay}	0.695	**0.745**	0.255

- M_{ula} correlates the most with the MLS_I

According to the results, we conclude that the most appropriate *operator* (distance-neighborhood) to solve the LOP is Ulam-*insert*. In addition, in order to solve the QAP, the best performing operator is Cayley-*interchange*. In these two problems the superiority of the mentioned operators is clear, but it does not seem to be a preferred operator for the PFSP and the TSP. In the former problem, there are instances in which Kendall's-τ, Cayley or Ulam obtain the best result (see Fig. 5.1c). Similarly, the *interchange* and *insert* neighbourhoods share the leadership as regards the MLSs. In relation to the TSP, we observe that GM_{cay} is the best EDA while MLS_I turns out to be the most efficient MLS. Even though the distance and the neighbourhood operators do not match in this case, note that the differences of GM_{cay} and MLS_I with respect to their competitors, M_{ula} and MLS_X are very small (see Fig. 5.1d and Fig. 5.3d). Besides, bear in mind that due to the symmetric nature of the solutions in the TSP (see Chapter 1), there are $2n$ equivalent solutions which can produce noise when optimising the problem.

The analysis of the correlations confirms that the correspondence between the Kendall's-τ and *swap*, Cayley and *interchange*, and Ulam and *insert* can be extrapolated to the result that EDAs and MLSs obtain under these distances and neighbourhoods. Such correspondence suggests that beyond the optimisation process imposed by each framework, the shape of the landscapes produced by the operators in each problem characterises the search.

In order to analyse these shapes and motivate the suitability of the different distances in each problem, in the next section we apply fitness landscape techniques to measure the ruggedness of the landscapes generated by the *swap*, *interchange* and *insert* neighbourhoods.

5.5 Analysis of the fitness landscapes

It is widely known that the smoothness/ruggedness of fitness landscapes can be studied in terms of number of local optima and their basins of attraction. Unfortunately, due to the factorial cardinality of the permutation search spaces, in practice, these measures are computationally affordable only for

quite small instances. Due to the complexity of the task, many works in the literature have dealt with the ruggedness of fitness landscapes proposing statistical measures that can describe their shape. Some examples are the *auto-correlation* [186], *correlation length* [111], statistical analysis of time series [22] or *information analysis* [183].

Due to the diversity of the landscapes considered in this chapter, we decided to follow the *information analysis* introduced in [183]. This proposal is based on the assumption that each fitness landscape can be considered as an ensemble of various objects, which are characterised by their size, form and distribution. In this sense, Vassilev et al. [183] proposed three information measures that characterise the structure of a fitness landscape defined over a time series obtained by a walk on the landscape: *information content* ($H(\epsilon)$), *partial information content* ($M(\epsilon)$) and *information stability*. For our analysis, we only consider the first two since they come to characterise the distribution and the number of local optima encountered in a walk on the landscape.

Instead of defining the measures directly on the sequence of fitness values obtained in a walk across the landscape, the authors translate this sequence to a string $S(\epsilon)$ of the symbols $\{\hat{1}, 0, 1\}$, where $\hat{1}$ denotes $(f_i - f_{i-1}) < \epsilon$, 0 stands for $|f_i - f_{i-1}| < \epsilon$ and 1 when $(f_i - f_{i-1}) > \epsilon$. ϵ stands for a minimum variation constant and i denotes the i-th step of the walk. Taking the string $S(\epsilon)$ as a basis, $H(\epsilon)$ calculates the entropy of all the adjacent substrings of length two, where the symbols in the substring are different. In other words, it estimates the variety of shapes in the walk with respect to the flat areas. On the other hand, $M(\epsilon)$ studies the modality of the path which calculates the percentage of different slopes in relation to the length of $S(\epsilon)$. The higher the number of slopes, the more rugged the landscape[3].

In order to analyse the ruggedness of the landscapes generated by the *swap*, *interchange* and *insert* neighbourhood for the TSP, LOP, PFSP and QAP, we calculate the average $H(\epsilon)$ and $M(\epsilon)$ of 300 *adaptive walks* of 1000 steps (as suggested in [183]) for one instance of each size. ϵ was set to 0 without any previous experimentation. The results obtained for $H(\epsilon)$ and $M(\epsilon)$ were quite similar, and thus, Table 5.3 only summarises the results for $H(\epsilon)$. Results in bold denote the neighbourhood that produced the smoothest landscape. Additionally, for one instance of size $n = 10$ of each problem, we calculated by brute force the exact number of local optima solutions generated by the *swap*, *interchange* and *insert* neighbourhoods. These results can be found in Table 5.4. Again, results in bold denote the lowest number of local optima.

Information analysis and local optima measures on the fitness landscape shown in Tables 5.3 and 5.4 support the conclusions drawn in the previous MLS experiments: the neighbourhoods that achieve the best results in each problem are those which generate the smoothest landscape (lowest $H(\epsilon)$). Such a statement turns out especially evident in Table 5.4 in terms of number of

[3] For the specific expressions we refer the interested reader to address the original paper [183].

Table 5.3: Average $H(\epsilon)$ results of 300 *adaptive walks* of 1000 steps in the *swap (S)*, *interchange (X)* and *insert (I)* neighborhoods.

Instance	LOP			QAP			PFSP			TSP		
Size	*S*	*X*	*I*	*S*	*X*	*I*	*S*	*X*	*I*	*S*	*X*	*I*
10	0.251	0.141	**0.135**	0.223	**0.133**	0.198	0.158	0.112	**0.088**	0.204	0.142	**0.137**
20	0.123	0.062	**0.054**	0.131	**0.070**	0.123	0.086	0.058	**0.047**	0.126	0.080	**0.070**
30	0.082	0.038	**0.034**	0.094	**0.050**	0.094	0.064	0.039	**0.031**	0.093	0.055	**0.047**
40	0.067	0.030	**0.025**	0.076	**0.039**	0.079	0.043	0.029	**0.025**	0.075	0.043	**0.036**
50	0.060	0.025	**0.020**	0.064	**0.030**	0.068	0.034	0.023	**0.019**	0.064	0.034	**0.029**
60	0.051	0.019	**0.019**	0.054	**0.025**	0.058	0.030	0.019	**0.014**	0.055	0.029	**0.024**
70	0.047	0.019	**0.014**	0.048	**0.022**	0.051	0.019	**0.014**	**0.014**	0.049	0.025	**0.021**
80	0.042	**0.014**	**0.014**	0.042	**0.019**	0.046	0.022	**0.014**	**0.014**	0.044	0.024	**0.019**
90	0.038	**0.014**	**0.014**	0.039	**0.019**	0.042	0.017	0.014	**0.010**	0.040	**0.019**	**0.019**
100	0.035	**0.014**	**0.014**	0.035	**0.014**	0.039	0.016	0.011	**0.008**	0.037	0.019	**0.014**

Table 5.4: The number of local optima generated by the *swap*, *interchange* and *insert* neighbourhood for one instance of size $n = 10$ for each problem.

Problem	*swap*	*interchange*	*insert*
LOP	105628	538	**9**
QAP	64367	**352**	13640
PFSP	20700	85	**11**
TSP	43424	1160	**1020**

local optima. The large number of local optima that the *swap* neighbourhood produced with respect to the other options is of special interest.

Regarding EDAs, the results obtained in Tables 5.3 and 5.4 also match the conclusions reported in Section 5.3.3. The preferred distances are those related to the neighbourhoods with the lowest $H(\epsilon)$ and number of local optima. In view of the results, given a new permutation problem, it is possible to find the most suitable distance for MaEDAs and GMEDAs based on the *information analysis* measures applied to the corresponding neighbourhood systems.

5.6 Discussion

In the previous sections, successive experiments showed that, for most of the problems there is an operator (distance or neighbourhood, depending on the framework) that has a superior performance. In addition, the fitness landscape analysis revealed that this is motivated by the low ruggedness of the landscape. But, which aspects of the operators and the problems influence the ruggedness of the landscape? According to our study, there are essentially two aspects that influence the shape of a landscape:

- *The size of the neighbourhood*: that is, the number of neighbours (solutions) at distance one. In local search optimization, large neighbourhoods are usually preferred since, at each step, more solutions can be evaluated.

Note that the number of local optima is bounded to $\frac{n!}{|N_x|}$, where $n!$ is the number of solutions in the search space, and $|N_x|$ stands for the size of the neighbourhood. Thus, the larger the neighbourhood, the lower the number of local optima. Even though this statement is correct, it is also very tricky. Let us consider the hypothetical scenario in which the neighbourhood consists of the whole search space. That scenario would be ideal since the number of local optima would correspond to the number of global optimum solutions. But would the search be trivial? No. The time required to evaluate such a large neighbourhood would be unaffordable ($O(n!)$).
In the particular case of *swap*, *interchange* and *insert*, the sizes of the neighbourhoods are $(n-1)$, $\frac{n(n-1)}{2}$ and $(n-1)^2$ respectively, and the time complexities to evaluate each neighbourhood are $O(n)$, $O(n^2)$ and $O(n^2)$.

- *The number of parameters that differ between neighbouring solutions.* According to the *information analysis*, the most successful neighbourhood (and by correspondence the distance) for each problem produces the smoothest fitness landscape. This implies that the fitness difference of a solution with respect to its neighbouring solutions is lower in smooth shapes than in rugged ones. In this sense, we think that the fitness variation of two neighbouring solutions can be considered in terms of the number of parameters that are different in the calculation of the fitness of both solutions[4]. We think that the smoother the landscape, the fewer the parameters that differ between neighbouring solutions. In Table 5.5 we summarise the number of parameters that are different between neighbouring solutions for the studied neighbourhoods in each problem. Note that i and j denote the positions of the items involved in the neighbourhood operation. n stands for the size of the instance, and m is the number of machines for the PFSP.

Table 5.5: The number of parameters that are different in the calculation of the fitness of a solution with respect to neighbouring solutions.

Problem	*swap*	*interchange*	*insert*
LOP	1	$2\|i-j\|-1$	$\|i-j\|$
QAP	$4(n-1)$	$4(n-1)$	$2n(\|i-j\|+1)-(\|i-j\|+1)^2$
PFSP	$m(n-i)$	$m(n-\min(i,j))$	$m(n-\min(i,j))$
TSP	2	4	3

According to Table 5.5, the *swap* neighbourhood has the lowest number of different parameters between neighbouring solutions in all the problems, which is a point in favour. However, it also produces the smallest neighbourhood. As regards *interchange* and *insert*, the size of the neighbourhoods are larger

[4] With *parameter* we refer to the smallest component of an instance that is used to calculate the fitness of a solution.

than in the *swap* neighbourhood and the number of parameters is also larger. In addition, it can be seen that the neighbourhood with the lowest number of different parameters, matches the neighbourhood that the smoothest landscape generates in each problem (see Table 5.3), and also obtains the best results with EDAs and MLSs (see Fig. 5.1 and 5.3).

5.7 Conclusions and future work

In this chapter, we reviewed the Kendall's-τ, Cayley and Ulam distances, and studied their performance under the Mallows and Generalized Mallows EDAs on a benchmark of 200 instances of the TSP, LOP, PFSP and QAP. We showed that the performance of Mallows and Generalized Mallows EDAs is clearly influenced by the distance used. Furthermore, the experimental study demonstrated that, for most of the problems, there is a distance that outperforms the rest almost systematically.

In order to better understand the impact of the distances, and have a priori knowledge that allows us to predict the most suitable distance for any given problem, we moved our analysis into the framework of local search algorithms. Based on the explicit relation between the Kendall's-τ, Cayley and Ulam, and the *swap*, *interchange* and *insert* neighbourhood systems respectively, we investigated whether such correspondence could be extrapolated to the performance of EDAs and LSs. According to our study, there exist strong correlations when the distance and neighbourhood used in each framework hold the correspondence. Moreover, we discovered that, beyond the optimisation process that each framework imposes, the shape of the landscape generated by the distance or neighbourhood is determinant in the search.

Based on *information analysis* techniques, we showed that the most competitive distances in each problem are those that generate the smoothest landscape. In fact, further research on landscape analysis suggested that, in order to generate smooth shapes, having large neighbourhoods and a low number of different parameters between neighbouring solutions are critical aspects.

An interesting extension of the work presented here is to study the convergence of the EDAs to local optima solutions of the distance considered in the model. In the hypothetical case that this could be confirmed, the connection between the results obtained by the EDAs and the shape of the landscape produced by the distance would be straightforward.

Alternatively, apart from the Kendall's-τ, Cayley and Ulam distances, there are others, such as Hamming, that could be developed to be used under the Mallows and GM models.

Finally, regarding the different behaviours shown by the Kendall's-τ, Cayley and Ulam distances, we find it interesting to combine different distances under the same EDA following a similar procedure to that seen for variable neighbourhood search algorithms.

6

The Plackett-Luce EDA

6.1 Introduction

As stated in the discussion introduced in the review of EDAs (Chapter 2), there exist multiple probability models for permutation spaces that could be introduced in the context of EDAs for solving permutation problems. In this sense, chapters 4 and 5 have been devoted to introduce and apply distance-based exponential models on this class of problems. In this chapter, we take a step further in this research line by introducing the Plackett-Luce probability model [109, 115, 142] to the framework of EDAs. The Plackett-Luce model is a stage-wise probability model, which decomposes the process of generating a permutation of n items into n sequential stages. Defined by a vector of scores $\boldsymbol{w}$, at the i-th stage, an item is selected and assigned to position i according to the probability based on the scores of the unassigned items.

In order to study the efficiency of the Plackett-Luce EDA, we deal with two different problems, the LOP and the PFSP, and we compare the results with those obtained by other permutation-oriented EDAs, such as EHBSA [176, 180] and NHBSA [181], and also with the newly introduced Mallows EDA (MaEDA) [29] and Generalized Mallows EDA (GMEDA) [26].

The remainder of the chapter is organized as follows: in the next section, the Plackett-Luce model is described in detail. Section 6.3 introduces the Plackett-Luce EDA. An experimental study over some LOP and PFSP instances is proposed afterwards in Section 6.4. Section 6.5 is devoted to discussing the experimental results. Finally, some conclusions and future work ideas are presented in Section 6.6.

6.2 The Plackett-Luce model

The Plackett-Luce model is a generalization of the well-known Bradley-Terry model [23], which induces a probability model on permutations by paired

comparisons of alternatives. The Bradley-Terry model specifies the probability that item a beats item b in terms of

$$P(a \prec b) = \frac{w_a}{w_a + w_b}$$

where w_a and w_b are positive-valued parameters associated with the items a and b. Note that the larger w_a is in comparison to w_b, the higher the probability that a is chosen.

Termed by Marden [115], the Plackett-Luce model takes its name from the combination of the independent work carried out by Plackett [142] and Luce [109]. This model extends the comparison that makes the Bradley-Terry model, to any number of items, which is specified by a parameter vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$.

Formally, for each item $i \in B$, where B ranges over all the possible non-empty subsets of the items $\{1, \ldots, n\}$, $P_B(i)$ is the probability that item i is chosen as the most preferred item among those listed in B. Formally,

$$P_B(i) = \frac{w_i}{\sum_{j \in B} w_j}$$

For identifiability, w_i parameters can be scaled to sum 1, so that w_i becomes the probability that item i is most preferred among the full set of items.

The ranking process defined by the Plackett-Luce model is best described by a simple illustration of the vase model interpretation: let us consider a multi-stage experiment where at each stage we are drawing a ball from a vase of infinite coloured balls. The parameter vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$ denotes the proportion of the balls of each colour. At the first stage a ball, $\sigma(1)$, is drawn from the vase; the probability of this selection is

$$\frac{w_{\sigma(1)}}{\sum_{j=1}^{n} w_{\sigma(j)}}$$

At the second stage, another ball is drawn. If it has the same colour as the previously drawn ball, that trial is discarded, and we keep trying until a new colour ball, $\sigma(2)$, is selected with probability

$$\frac{w_{\sigma(2)}}{\sum_{j=2}^{n} w_{\sigma(j)}}$$

The process is repeated until a complete permutation σ of coloured balls is obtained. The probability of the chosen permutation of balls is assumed to be the product of choice probabilities across the various stages.

Formally, for every permutation σ and a parameter vector $\boldsymbol{w}$, the Plackett-Luce model is given by

$$P(\sigma|\boldsymbol{w}) = \prod_{i=1}^{n-1} \frac{w_{\sigma(i)}}{\sum_{j=i}^{n} w_{\sigma(j)}} \tag{6.1}$$

Obviously, the larger the parameter w_i in comparison to w_j, $i \neq j$, the higher the probability that the item i appears on a top rank of σ.

According to the definition of the Plackett-Luce model, the choice probabilities at the k-th stage depend only on the set of items remaining at that stage and not on the relative ordering of the $k-1$ items previously selected. Therefore, the probability associated to every permutation σ can be expressed in the form

$$P(\sigma) = P_{\{i_1,\ldots,i_n\}}(i_1)P_{\{i_2,\ldots,i_n\}}(i_2)\cdots P_{\{i_{n-1},i_n\}}(i_{n-1}) \tag{6.2}$$

where $P_B(i)$ is the probability that item i is chosen as the most preferred item among those listed in B, and B ranges over all the non-empty subsets of the items $\{1,\ldots,n\}$. Any ranking model $P(\sigma)$ which can be expressed in this form, for some collection of choice probabilities, is said to be *L-decomposable*.

At first glance, it may appear that all ranking models are *L-decomposable*, since according to the *chain rule*, the joint probabilities can be computed as products of conditional probabilities. However, as Critchlow et al. [48] noted, consider for example the conditional probability $P(\sigma(3) = i_3|\sigma(1) = i_1, \sigma(2) = i_2)$. This probability may depend on the relative ordering of items i_1 and i_2, making the probability of $P(\sigma(3) = i_3|\sigma(1) = i_2, \sigma(2) = i_1)$ different from the former one. In other words, *L-decomposability* requires that the preceding conditional probability does not depend on the relative ordering of i_1 and i_2.

6.2.1 Learning and sampling

In order to introduce the Plackett-Luce model in the framework of EDAs, it is necessary to provide efficient learning and sampling methods.

With respect to the learning process, the estimation of the parameter vector $\boldsymbol{w}$ has been addressed several times in the literature. Hunter [77] proposes a maximum likelihood estimation (MLE) method based on a Minorization-Maximization (MM) algorithm. Alternatively, Guiver et al. [70] adopted a method based on Bayesian inference with a Power EP (Expectation Propagation) scheme. However, since the method proposed by Hunter [77] is the only algorithm in the literature able to calculate the MLE parameters, this is the method implemented in this paper. In what follows, we describe the MM method proposed by Hunter. In Alg. 6.1, a pseudocode of the MM algorithm is introduced.

Assuming a set of $\{\sigma_1,\ldots,\sigma_N\}$ independent permutations, the likelihood of this set, given $\boldsymbol{w} = (w_1,\ldots,w_n)$ is,

$$L(\boldsymbol{w}|\sigma_1,\ldots,\sigma_N) = \prod_{k=1}^{N}\prod_{i=1}^{n-1}\frac{w_{\sigma_k(i)}}{\sum_{j=i}^{n} w_{\sigma_k(j)}} \tag{6.3}$$

The maximum likelihood estimation of $\boldsymbol{w}$ is then given by those parameters that maximize Eq. 6.3 or, equivalently, the log-likelihood function

```
input: initial guess for the parameters w^0, γ precision digit
  do
    l ← l + 1
    Construct function Q(w|w^l)
    w^{l+1} ← arg max_w Q(w|w^l)
  while ||w^{l+1} − w^l|| < γ
output: Parameters vector w
```

Alg. 6.1: Pseudocode of the Minorization-Maximization algorithm.

$$\ln L(\boldsymbol{w}|\sigma_1,\ldots,\sigma_N) = \sum_{k=1}^{N}\sum_{i=1}^{n-1}\left(\ln w_{\sigma_k(i)} - \ln\sum_{j=i}^{n} w_{\sigma_k(j)}\right) \tag{6.4}$$

In order to find the maximum likelihood $\boldsymbol{w}$ parameters, Hunter [77] proposed an iterative algorithm whose idea is to maximise, in each iteration, a function that minorizes the original log-likelihood. In order to fit Eq. 6.4, Hunter proposes to construct a minorizing function using the inequality

$$\forall x, y > 0 \quad \text{we have} \quad -\ln x \geq 1 - \ln y - \frac{x}{y}$$

and by replacing x and y with $\boldsymbol{w}$ and $\boldsymbol{w}^l$ respectively, we define the surrogate function

$$Q(\boldsymbol{w}|\boldsymbol{w}^l) = \sum_{k=1}^{N}\sum_{i=1}^{n-1}\left(\ln w_{\sigma_k(i)} - \frac{\sum_{j=i}^{n} w_{\sigma_k(j)}}{\sum_{j=i}^{n} w^l_{\sigma_k(j)}}\right) \tag{6.5}$$

that minorizes the original log-likelihood function $L(\boldsymbol{w})$ at $\boldsymbol{w}^l$, where $\boldsymbol{w}^l$ stands for the estimated parameter vector $\boldsymbol{w}$ at iteration l.

At this point, in order to find the $\boldsymbol{w}$ parameters that maximise $Q(\boldsymbol{w}|\boldsymbol{w}^l)$, this is derived and equated to 0, which gives

$$w_r^{l+1} = \frac{h_r}{\sum_{k=1}^{N}\sum_{i=1}^{n-1}\delta_{kir}[\sum_{j=i}^{n} w^l_{\sigma_k(j)}]^{-1}} \tag{6.6}$$

where h_r stands for the number of permutations in which the r-th item is ranked higher than last, and

$$\delta_{kir} = \begin{cases} 1 \text{ if an item larger than } i-1 \text{ is ranked } r \\ \quad \text{in the } k\text{-th permutation} \\ \\ 0 \text{ otherwise} \end{cases}$$

In other words, δ_{kir} is the indicator of the event that item r receives a rank no better than i in the k-th permutation. For a more detailed description of

the MLE method for the estimation of the parameters, we refer the interested reader to Hunter [77].

As regards the time-complexity of learning the MLE parameters, each iteration of the MM method is done in time $O(nN)$, where N denotes the cardinality of the set of permutations $\{\sigma_1, \ldots, \sigma_N\}$. However, the number of iterations performed by the algorithm depends on the data itself, and thus, its time-complexity cannot be asserted.

On the other hand, Hunter also proves that the MM algorithm is guaranteed to converge to the unique maximum likelihood estimators if the following assumption holds:

> *"Assumption 1. In every possible partition of the samples into two non-empty subsets, some item in the second set ranks higher than some item in the first set at least once."*

Unfortunately, this assumption can not always be satisfied in real examples involving sparse data, or on the contrary, very homogeneous data. And thus, in Section 6.3 we introduce some additional strategies that make use of this learning approach efficiently.

Once the parameter vector $\boldsymbol{w}$ is known, the sampling procedure consists of following the model of Luce. An illustrative example is that of the vase model introduced previously. This model describes a sequential permutation generator method in which the items are sampled from the first to the last position (i.e., from the most to the least preferred item). The parameter space consists of n positive weights $(w_1, ..., w_n)$ that sum 1. These probabilities are used to sample the first position of the permutation, with $P(\sigma(1) = j) = w_j$. The following $\{2, ..., n\}$ positions are filled by sampling the remaining w parameters normalized to sum 1. The process is repeated until a complete permutation is obtained. The sampling process of a solution is carried out in time $O(n^2)$.

6.3 Plackett-Luce EDA

The Plackett-Luce EDA (PLEDA) is a classical EDA where the probabilistic model used is the Plackett-Luce model introduced in the previous section. In Alg. 6.2 the general outline of the PLEDA is described.

As aforementioned in Section 6.2, the MM algorithm is not guaranteed to converge to the maximum likelihood estimators if assumption 1 is not satisfied. Due to the homogeneous populations that are usually sampled in the advanced generations of the PLEDA, it is foreseeable that Assumption 1 will no longer hold. In order to deal with this drawback, we introduce a random solution in the selected set of individuals used to learn the probability model. Although, such a procedure fixes the problem in most cases, it does not guarantee convergence, and therefore, additionally, when a maximum number of iterations is exceeded, the MM algorithm is stopped and the parameter vector $\boldsymbol{w}$ from the previous generation is considered for the sampling step.

1 $D_0 \leftarrow$ Generate M individuals (the initial population) uniformly at random.
2 t $\leftarrow 1$
3 **do**
4 $\quad D_{t-1} \leftarrow$ Evaluate individuals
5 $\quad D_{t-1}^{Se} \leftarrow$ Select $N \leq M$ individuals from D_{t-1}
6 $\quad \boldsymbol{w} \leftarrow$ Estimate maximum likelihood parameters from D_{t-1}^{Se}
7 $\quad D_M \leftarrow$ Sample $M-1$ individuals from $p_t(\sigma) = p(\sigma|D_{t-1}^{Se}, \boldsymbol{w})$.
8 $\quad D_t \leftarrow$ Create the new population from D_{t-1} and D_M.
9 **until** Stopping criterion is met

Alg. 6.2: Pseudocode of the Plackett-Luce EDA.

6.4 Experimentation

In order to analyze the performance of PLEDA, we apply our proposal on two very well known permutation problems, the LOP and the PFSP with the total flow time criterion. In order to have a real feedback of the efficiency of the proposed algorithm, we compare our approach to the Mallows EDA (MaEDA), the Generalized Mallows EDA (GMEDA) under the Kendall's-τ distance, and also to NHBSA and EHBSA, which have been included in order to have a reference with respect to probability models based on low-order marginals.

MaEDA, GMEDA and PLEDA were coded in C++ programming language. As regards to NHBSA and EHBSA, the original code obtained from the authors is in Java. The experimentation was conducted on a cluster of 20 nodes, each of them equipped with two Intel Xeon X5650 CPUs and 48GB of memory.

In relation to the experimentation instances, we chose 28 LOP instances with sizes $n = 150$ and $n = 250$ (14 instances of each size) from the XLOLIB instances benchmark proposed by Schiavinotto et al. [155]. Regarding the PFSP instances, 10 instances with configuration 100×20 other 10 with 200×20 were considered from the Taillard's [168] benchmark. Another 10 instances of configurations 250×20 were taken from the random benchmark of PFSP instances proposed in [26].

6.4.1 Parameter settings

In the list below we summarize the parameters employed in all the EDAs:

- Population size: $10n$.
- Selected individuals: n.
- Selection type: truncation.
- Sampled individuals: $10n - 1$.

- Elitism criterion is followed.
- Stopping criterion: $1000n^2$ evaluations.

Other particular settings for each of the algorithms:

- The template strategy (WT) was considered in NHBSA and EHBSA (see Chapter 2).
- The upper-bound θ values for MaEDA [29] and GMEDA [26] were set as specified in their respective papers.
- A maximum of 1000 iterations were set to the MM algorithm in the learning step of the Plackett-Luce model.

6.4.2 Results

Each *algorithm* - *instance* pair was run 20 times. The performance measure employed in our study is the average relative percentage deviation (ARPD):

$$ARPD = \left(\sum_{i=1}^{20} \frac{(Res_i - Best)\times 100}{Best}\right)/20$$

where Res_i denotes the best solution found by the algorithm in the i-th repetition, and $Best$ stands for the best known solution for the instance under study. The ARPD results of the executions are summarized in Table 6.1 and 6.2. Results in bold indicate the algorithm that obtained the lowest ARPD among the compared approaches.

As can be observed in Tables 6.1 and 6.2, the efficiency of the algorithms changes drastically from one problem to the other. In the case of the LOP instances, PLEDA obtains the best results in 26 out of 28 instances, and NHBSA obtains the best result in three instances. The good performance of the GMEDA is very remarkable, particularly in the instances of size $n = 250$. On the contrary, EHBSA obtains the worst result in all the instances.

With respect to the results obtained for the PFSP instances, GMEDA outperforms the rest of the algorithms in all the instances. Although the results of MaEDA are not as good as those obtained by GMEDA, it is worth noting that, for instances of size $n = 100$, MaEDA is the second best performing algorithm. On the other hand, we see that PLEDA, NHBSA and EHBSA, do not obtain competitive results compared to GMEDA.

6.4.3 Statistical analysis

In order to state whether there exist statistical differences among the algorithms, we applied a non-parametric Friedman's test to the average results obtained by PLEDA, MaEDA, GMEDA, NHBSA and EHBSA for each size of the LOP ($n = 150$ and $n = 250$), and each configuration of the PFSP (100×20, 200×20 and 250×20) separately. The level $\alpha = 0.05$ of significance was set. The statistical test reported significant differences between the algorithms in all the cases. Thus, a post-hoc method was used to carry out

Table 6.1: ARPD results of 20 repetitions of the EDAs for the LOP instances.

Size	Instance	Best Known	PLEDA	MaEDA	GMEDA	NHBSA	EHBSA
150	t65b11xx	6547627	**0.056**	0.072	0.065	0.064	0.107
	t65d11xx	3646934	**0.064**	0.081	0.074	0.078	0.123
	t65f11xx	3231148	**0.062**	0.081	0.074	0.077	0.121
	t65l11xx	255390	**0.041**	0.058	0.052	**0.041**	0.082
	t65n11xx	558953	**0.052**	0.070	0.065	0.060	0.113
	t70b11xx	9802850	**0.058**	0.074	0.068	0.065	0.109
	t70d11xx	6305710	**0.067**	0.084	0.076	0.076	0.117
	t70f11xx	5285191	**0.069**	0.088	0.078	0.082	0.123
	t70l11xx	438087	**0.052**	0.074	0.069	0.054	0.102
	t70n11xx	959803	**0.056**	0.073	0.066	0.061	0.112
	t75d11xx	9850135	**0.064**	0.078	0.073	0.071	0.115
	t75e11xx	42053463	**0.048**	0.069	0.062	0.300	0.300
	t75k11xx	1569847	**0.055**	0.074	0.064	0.292	0.291
	t75n11xx	1767716	**0.060**	0.078	0.071	0.302	0.299
250	t65b11xx	17751127	**0.074**	0.086	0.080	0.088	0.146
	t65d11xx	9745769	**0.083**	0.099	0.090	0.105	0.161
	t65f11xx	8739670	**0.078**	0.094	0.086	0.097	0.159
	t65l11xx	679527	**0.061**	0.075	0.071	0.068	0.132
	t65n11xx	1475116	**0.075**	0.092	0.084	0.094	0.162
	t70b11xx	26133557	**0.074**	0.088	0.081	0.092	0.157
	t70d11xx	16658483	**0.080**	0.095	0.086	0.097	0.157
	t70f11xx	14178278	**0.087**	0.101	0.094	0.109	0.162
	t70l11xx	1132769	**0.066**	0.082	0.076	0.078	0.150
	t70n11xx	2515339	**0.074**	0.091	0.081	0.094	0.162
	t75d11xx	25972723	**0.075**	0.088	0.080	0.090	0.155
	t75e11xx	110318580	0.079	0.094	0.086	**0.078**	0.143
	t75k11xx	4249417	**0.079**	0.096	0.087	0.082	0.142
	t75n11xx	4658704	0.075	0.090	0.082	**0.074**	0.140

all pairwise comparisons and determine which algorithms stand out from the rest of the approaches. In particular, Shaffer's static procedure was used, as suggested for such cases in [61]. Again, the significance level was fixed to $\alpha = 0.05$. The statistical analysis confirmed the following points about the results obtained for the LOP instances:

- For instances of size $n = 150$, PLEDA is significantly better than the rest of the algorithms.
- For instances of size $n = 250$, PLEDA is significantly better than the rest of the algorithms except GMEDA, for which no significative differences were found.
- EHBSA is the worst algorithm for solving LOP instances, except MaEDA in $n = 150$, for which no significative differences were found.

Regarding to the PFSP instances:

Table 6.2: ARPD results of 20 repetitions of the EDAs for the PFSP instances.

Instance		Best Known	PLEDA	MaEDA	GMEDA	NHBSA	EHBSA
100×20	1	367267	0.074	0.045	**0.026**	0.060	0.062
	2	374032	0.076	0.044	**0.026**	0.059	0.057
	3	371417	0.075	0.047	**0.026**	0.057	0.057
	4	373822	0.075	0.048	**0.027**	0.061	0.060
	5	370459	0.077	0.048	**0.026**	0.061	0.060
	6	372768	0.080	0.053	**0.029**	0.061	0.061
	7	374483	0.076	0.044	**0.027**	0.061	0.062
	8	385456	0.074	0.047	**0.025**	0.058	0.057
	9	376063	0.071	0.042	**0.025**	0.058	0.055
	10	379899	0.078	0.052	**0.030**	0.061	0.059
200×20	1	1226879	0.083	0.070	**0.027**	0.071	0.076
	2	1241811	0.085	0.073	**0.023**	0.074	0.082
	3	1266153	0.085	0.081	**0.054**	0.070	0.070
	4	1237053	0.087	0.077	**0.024**	0.076	0.082
	5	1223551	0.083	0.066	**0.022**	0.075	0.085
	6	1225254	0.079	0.050	**0.020**	0.073	0.084
	7	1241847	0.080	0.051	**0.019**	0.074	0.085
	8	1240820	0.085	0.073	**0.029**	0.074	0.081
	9	1229066	0.085	0.062	**0.022**	0.078	0.086
	10	1247156	0.082	0.074	**0.022**	0.073	0.077
250×20	1	1847471	0.082	0.076	**0.041**	0.075	0.085
	2	1892483	0.085	0.081	**0.067**	0.075	0.082
	3	1879796	0.077	0.065	**0.024**	0.072	0.082
	4	1852134	0.084	0.076	**0.037**	0.076	0.089
	5	1836747	0.090	0.085	**0.075**	0.079	0.086
	6	1859562	0.080	0.074	**0.037**	0.074	0.086
	7	1876186	0.078	0.074	**0.041**	0.070	0.077
	8	1850963	0.088	0.080	**0.032**	0.078	0.085
	9	1859994	0.080	0.074	**0.023**	0.074	0.083
	10	1837633	0.085	0.079	**0.029**	0.076	0.084

- For instances with configurations 100×20 and 200×20, GMEDA is significantly better than PLEDA, EHBSA and NHBSA. However, no significant differences were found with respect to MaEDA.
- PLEDA and EHBSA are significantly worse that the rest of the algorithms (no significative differences were found between them) for instances with configurations 100×20 and 200×20.
- For instances with configuration 250×20, GMEDA is significantly better than EHBSA and PLEDA. On the other hand, EHBSA is significantly worse than NHBSA.

6.5 Discussion

In the previous section, the experimental study showed that the performance of the analyzed EDAs varies according to the problem being optimized. Obviously, in the end, the main reason for that behaviour is the probabilistic model each EDA uses. In order to shed some light on the results obtained, this section is devoted to discussing the properties of the probabilistic models under study, and their possible resemblances with the characteristics of the LOP and the PFSP.

6.5.1 Choice probabilities and the LOP

The experimental results revealed that Plackett-Luce is the best performing model for solving the LOP. According to the definition of the problem, the contribution of an item to the fitness function in the j-th position depends on the items that are located in the previous $\{1, \dots j-1\}$ positions and in the posterior $\{j+1, \dots, n\}$ positions, but not on the relative ordering within each of the sets of position (see Section 1.1.5).

As previously introduced in Section 6.2, the probability of every permutation σ in the Plackett-Luce model can be expressed as a product of choice probabilities P_B in the form of Eq. 6.2, where for each item i there is a value p_i such that

$$P_B(i) = p_i / \sum_{k \in B} p_k,$$

for any non-empty subset B of $\{1, \dots, n\}$. This implies that the choice probability at the j-th stage depends only on the set of items remaining at that stage, and not on the relative ordering of the $j-1$ items previously selected.

The similarity between the definition of the LOP and the definition of the choice probabilities in the Plackett-Luce model, suggests that the successful performance of the PLEDA in this problem is motivated by the ability of this model to represent the characteristics of the LOP efficiently.

As regards the Generalized Mallows model, when the metric employed is the Kendall-τ distance (as in this case), the probability distribution can be factorized into a sequence of $n-1$ stages as follows

$$P(\sigma) = \prod_{j=1}^{n-1} P(V_j(\sigma\sigma_0^{-1}) = r_j) \tag{6.7}$$

where $P(V_j(\sigma\sigma_0^{-1}) = r_j)$ denotes the choice probability made at stage j, and r_j indicates the number of mistakes made at that stage with respect to a central permutation σ_0. Note that, unlike the Plackett-Luce, the set of choice probabilities at stage j, P_{B_j}, is a fixed set of probabilities that does not depend on the position of the items in B_j. As a consequence, the resemblance with the LOP is not as direct as that for the Plackett-Luce model.

6.5.2 Distance-based probability models and the PFSP

Regarding the PFSP, the conducted experiments show that the EDAs that implement *distance-based probability models* stand out over the rest of the algorithms, especially the Generalized Mallows model. Surprisingly, the Plackett-Luce is not competitive in this problem.

The definition of the PFSP states that, in this problem, the contribution of the i-th item (job) to the fitness function is highly conditioned by the relative ordering of the previously fixed items. Taking that into account, in the case of Plackett-Luce, the choice probabilities P_B do not consider the relative ordering of previously made choices, we conclude that the bad performance of the PLEDA might be induced by the inappropriate characteristics of the model.

On the contrary to the Plackett-Luce, the Mallows and Generalized Mallows models demonstrated very good performance for instances of 100×20. However, this performance was not maintained by the Mallows model for larger instances.

As aforementioned in Eq. 6.7, when the Kendall-τ distance is used, the Generalized Mallows can be expressed as a *multistage ranking model*. This means that the processes of constructing a permutation has $n-1$ stages, and the choice probabilities at stage j, defined as

$$P(V_j(\sigma\sigma_0^{-1}) = r_j) = \frac{e^{-\theta_j r_j}}{\psi_j(\theta_j)}$$

depend only on the current stage and are not affected by the decisions made at previous and posterior stages. Note that the parameter θ_j completely determines the probabilities for stage j and is inversely related to the expected number of mistakes made at the stage (for further details and proofs we suggest the reader refer to [48, 55]). As a consequence, the Generalized Mallows model has higher expressiveness than the Mallows model (which only uses one spread parameter).

The experimental study demonstrated that such generalization is beneficial for both problems, since the GMEDA obtains better results than the MaEDA in all the instances tested.

6.5.3 Low-order marginal model EDAs

The experimentation showed that, in general, NHBSA and EHBSA are not competitive enough to beat the best performing probability models for permutations in each problem. In the LOP instances, we see that NHBSA performs remarkably better than EHBSA, but PLEDA is the reference algorithm with the exception of 2 instances. When solving the PFSP instances, NHBSA and EHBSA obtain similar results (the highest ARPD difference is 0.013 in favour of NHBSA) far from the best performing algorithm in this problem: the GMEDA.

6.6 Conclusions and future work

In this chapter, we presented a novel estimation of distribution algorithm for dealing with permutation problems called Plackett-Luce EDA (PLEDA). The PLEDA makes use of a probability model for permutations called Plackett-Luce.

In order to evaluate the efficiency of our proposal, it was tested on two well-known permutation problems, the LOP and the PFSP. In addition to the PLEDA, other meaningful EDAs, such as the Mallows EDA (MaEDA), the Generalized Mallows EDA (GMEDA), and NHBSA and EHBSA, were also included.

The experimental study revealed that the performance of each of the studied EDAs varies from one problem to the other. Furthermore, the results obtained showed that the Plackett-Luce model is the most efficient model for solving the LOP, but, in contrast, it is not competitive when optimizing the PFSP. The *distance-based probability models*, and especially the Generalized Mallows model, turned out to be the best ones for solving the PFSP.

For the sake of understanding the results, a brief discussion about the properties of the models and the problems was conducted. A preliminary analysis suggests that the results seen in the experimentation might be motivated by some specific properties of the models, such as the definition of the choice probabilities, the number of parameters used, or the distance-based nature of the models. However, in order to understand the use of these probability models in the framework of EDAs, we think that the relation between probability models for permutations and permutation problems deserves deeper analysis.

In order to extend this work, we find it very valuable to incorporate more probability models for permutations, particularly those based on pairwise comparisons. Moreover, we consider it interesting to see the behavior of the proposed model on a wider set of permutation-based problems, such as the travelling salesman problem, the quadratic assignment problem or the vehicle routing problem.

Part II

Studying the linear ordering problem

7

The linear ordering problem revisited

7.1 Introduction

The Linear Ordering Problem (LOP) is a classical combinatorial optimisation problem which has received the attention of the research community since it was studied for the first time by Chenery and Watanabe [37]. Garey and Johnson [63] demonstrated that the LOP is an *NP-hard* problem, thereby evidencing the difficulty of solving the LOP instances up to the optimality. However, due to its numerous applications in diverse fields such as archeology [65], economics [100], graph theory [35], machine translation [174] or mathematical psychology [88], we can find a wide variety of papers that have dealt with the LOP by means of exact, heuristic and metaheuristic strategies.

Among the exact methods, the most meaningful include Branch and Bound [34, 86], Branch and Cut [68] and Cutting Plane algorithms [126, 127]. These methods, as highlighted in [155], behave competitively for instances of specific characteristics with up to a few hundred columns and rows. However, their computation time increases notably with the size of the instances, and thus, for most of the existing benchmarks it is not possible to solve large instances in a reasonable time span. Beyond the exact proposals, pioneering works proposed constructive heuristics [8, 14, 37]. Such approaches were later outperformed by the advances produced in metaheuristic optimisation. Proof of this are the solutions based on Local Search [32, 89], Genetic Algorithms [33], Genetic Programming [143], Tabu Search [93], Scatter Search [25], Variable Neighborhood Search [60], Ant Colony Optimisation [42, 141], and, recently, Estimation of Distribution Algorithms [30].

According to a recent review of Martí et al. [117], the Memetic Algorithm (MA) and the Iterated Local Search (ILS) proposed by Schiavinotto and Stützle [155], are the algorithms that currently shape the state-of-the-art of the LOP. The MA is a hybrid algorithm which combines the canonical structure of a Genetic Algorithm with a high presence of local search procedures, either in the initialisation of the population or in the evolutionary process itself. On the other hand, the ILS is a strategy that iteratively applies

a local search algorithm to a single solution. When the process gets trapped in a local optimal solution, the ILS applies a perturbation to the current solution, and continues with the optimisation process until a termination criterion is satisfied. Both algorithms include an efficient implementation of a greedy local search algorithm with the insert neighbourhood designed specifically to solve the LOP.

As seen for most of the combinatorial optimisation problems, the hardness of solving a specific instance is not only related to its size, but also to other additional parameters, unknown in most cases. In this regard, the community has tried to better understand the characteristics of the LOP that determine the difficulty of the instances, and similarly, has tried to identify the features that could be useful to guide the algorithms throughout the optimisation process. In this sense, Schiavinotto and Stützle [155] sketched out the properties that could somehow characterise the difficulty of the LOP instances. The authors defined the *sparsity*, *variation coefficient (VC)*, *skewness* and *fitness distance correlation* as measures of instance hardness, and showed that real-life instances, which are apparently more difficult than artificial ones, present significant differences in *sparsity*, *VC* and *skewness* with respect to random benchmark instances. Nevertheless, the relation between the mentioned properties and the suitability of the MA and the ILS to solve the LOP is not straightforward.

In the same research line, Betzler et al. [16] published a detailed work on the parameterised complexity for intractable median problems, and particularly on the Kemeny ranking problem, which can be seen as a subclass of the LOP. Although the parameterised complexity studied in the cited work is of great relevance, the analysis of [16] stands on a specific property of the Kemeny, which does not hold for the general LOP, and thus, the extension of the parameterised complexity to the LOP is not straightforward.

The aforementioned works and the absence of a detailed work that performs an in-depth analysis of the LOP motivated this paper. In this work we study the properties that the optimal solutions of the LOP hold in the framework of local search algorithms, placing special emphasis on the position where the items are located, and identifying the role of the associated matrix entries in the generation of local optima.

The chapter is divided into two parts: first, we provide a detailed description of the LOP, introducing definitions and theorems that study the structure of the problem with respect to the optimality of the solutions in the context of local search algorithms. Particularly, we emphasise the influence that the position of the items that compound a solution have when generating local optima. As a result of the theoretical study, a *restricted* version of the insert neighbourhood is proposed. This neighbourhood discards specific insert operations that involve moving items to positions at which they cannot produce local optima. The theoretical analysis demonstrates that these insert operations will never be the operations that most improve the solution in the neighbourhood.

The second part of the paper is devoted to demonstrating the validity of the *restricted insert neighbourhood*. In this sense, we develop a *restricted* version of the two best performing algorithms for the LOP: the MA and the ILS. Experimental results show that the restricted versions of the algorithms outperform the classical designs in 90% and 93.3% of the executions respectively, obtaining the same results for the rest of the cases (under the same number of evaluations). Moreover, additional experiments devoted to measuring the execution time needed to perform a given number of iterations show that the *restricted* approaches are faster than the classical versions for most of the instances.

The remainder of the paper is organised as follows: in the next section, a detailed definition of the LOP is given. In Section 7.3, the structural analysis of the LOP is introduced, emphasising the contribution of the items to the fitness function. Next, in Section 7.4 the optimality of the LOP solutions is described in the context of local search algorithms, and in particular for the insert neighbourhood system. Section 7.5 is devoted to investigating the basis for the restricted insert neighbourhood system. In order to demonstrate the validity of the introduced analysis, a complete experimental study is introduced in Section 7.6. Finally, some conclusions and ideas for future work are drawn in Section 7.7.

7.2 The linear ordering problem

Given a matrix $\mathbf{B} = [b_{k,l}]_{n \times n}$ of numerical values, the linear ordering problem consists of finding a simultaneous permutation σ of the rows and columns of B, such that the sum of the entries above the main diagonal is maximized (or equivalently, the sum of the entries below the main diagonal is minimized). The equation below formalizes the LOP function:

$$f(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{\sigma(i),\sigma(j)} \tag{7.1}$$

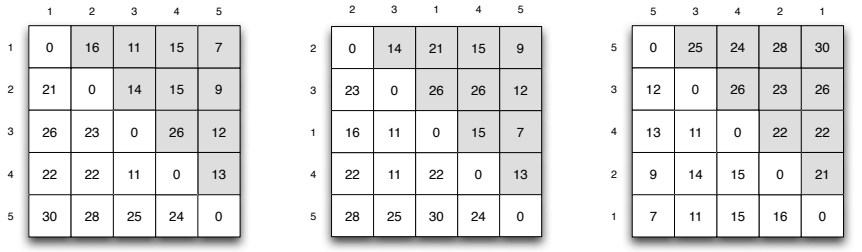
where $\sigma(i)$ denotes the item associated to the row (and column) ranked at position i in the solution σ[1]. This representation of the LOP is also known as the *triangulation problem of input-output matrices*. Although alternative representations of the problem can be found in [116] and [35], due to the theoretical simplicity and readability of the exposed approach, in the remainder of the paper the *triangulation* representation will be considered.

Example 7.1 Let us introduce an example for a $n = 5$ LOP instance which will be used throughout the paper[2]. In Fig. 7.1, three different solutions, e, σ

[1] From now on, σ will denote any permutation in $\mathbb{S}_n$, and e will stand for the identity permutation $12 \ldots n$ of size n. In addition k and l will denote the items within a permutation σ, and i, j and z will be used to identify the positions of σ.

[2] This example was extracted from [116].

and σ^* are described. The initial matrix is represented by the identity permutation $e = 12345$ (see Fig. 7.1a), and its fitness, $f(e)$, is 138. The solution $\sigma = 23145$ introduces a different ordering of the items that provides a solution better than e (see Fig. 7.1b), $f(\sigma)$ is 158. The optimal solution for this example is given by $\sigma^* = 53421$ (see Fig. 7.1c), with fitness $f(\sigma^*) = 247$.



(a) $e = 12345$, $f(e) = 138$ (b) $\sigma = 23145$, $f(\sigma) = 158$ (c) $\sigma^* = 53421$, $f(\sigma^*) = 247$

Fig. 7.1: Three different solutions of a $n = 5$ instance.

7.3 Analysis of the problem

In this section, we analyse the LOP by explaining the association between the items in σ and the arrangement of the $b_{k,l}$ entries in the matrix $\mathbf{B}$. In addition, we describe the fitness variation that provokes changing the position of an item within σ, and the role of the $b_{k,l}$ entries in this regard. As a necessary background to understand the latter content of the paper, in the following list we outline some meaningful properties of the LOP that define the association between the items in σ, and the $b_{k,l}$ entries in the $\mathbf{B}$ matrix.

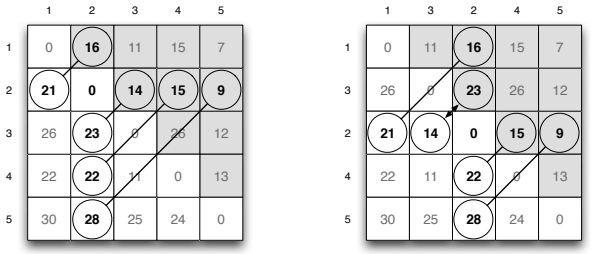
For any permutation of items σ of size n and a matrix $\mathbf{B}$ of size $n \times n$:

- Every item $\sigma(i) = k$, $i = 1, \ldots, n$, has associated $2(n-1)$ entries of $\mathbf{B}$: $n-1$ from row k and $n-1$ from column k.
- The set of associated entries of every item $\sigma(i) = k$, $i = 1, \ldots, n$, can be organised in pairs, i.e. every entry in row k, $b_{k,\sigma(j)}$ (where $j = 1, \ldots, n$), has a pair in column k, $b_{\sigma(j),k}$, symmetrically located with respect to the main diagonal.
- All the pairs of entries associated to item $\sigma(i) = k$, $\{b_{k,1}, b_{1,k}\}, \ldots, \{b_{k,n}, b_{n,k}\}$, remain associated to this item regardless of its position and the position of the rest of the $n-1$ items.
- Every entry $b_{\sigma(i),\sigma(j)}$ is associated to two items, $\sigma(i)$ and $\sigma(j)$.

- For every pair $\{b_{\sigma(i),\sigma(j)}, b_{\sigma(j),\sigma(i)}\}$ of entries, one of the entries is always located above the main diagonal, and the other entry is located below, thereby bounding the best fitness contribution of this pair to $\max\{b_{\sigma(i),\sigma(j)}, b_{\sigma(j),\sigma(i)}\}$ in the best case, and to $\min\{b_{\sigma(i),\sigma(j)}, b_{\sigma(j),\sigma(i)}\}$ in the worst case.

In the remainder of the section, these characteristics and some extra definitions will be detailed with the help of illustrative examples.

Example 7.2 In this example, two different solutions are introduced, $e = 12345$ in Fig. 7.2a, and $\sigma = 13245$, in Fig. 7.2b. In both figures, the entries associated to the item 2 are highlighted in bold. We see that, in spite of the different ordering, in both solutions the set of the entries associated to the item 2 are the same i.e. $(21, 14, 15, 9, 16, 23, 22, 28)$. Moreover, even though the position of item 2 is different in e and σ ($e(2) = 2$ and $\sigma(3) = 2$), the pairwise relation of the associated entries remains unchanged (see the circled items in Fig. 7.2).



(a) $e = 12345,\ f(e) = 138.$ (b) $\sigma = 13245,\ f(\sigma) = 147.$

Fig. 7.2: Two different solutions for an instance of $n = 5$. Circled entries linked with edges denote the pairs of entries associated to item 2.

Checking the location of the associated entries of item 2 in σ, we note that the pair $(14, 23)$ has exchanged its positions in σ, being now 14 below the main diagonal and 23 above it. Prior to studying the implications that the movement of an item has in the fitness, in the following lines we first introduce a new term: the *contribution* of an item to the fitness function.

When item $k = 1, \ldots, n$ is located at position i in σ, i.e. $\sigma(i) = k$, the contribution of item k to the fitness function is given by the sum of the entries of column k in the rows $\{\sigma(1), \ldots, \sigma(i-1)\}$ and the sum of the entries of row k in the columns $\{\sigma(i+1), \ldots, \sigma(n)\}$. That is to say, the previous $i-1$ items $\{\sigma(1), \ldots, \sigma(i-1)\}$ and the posterior $n-i$ items $\{\sigma(i+1), \ldots, \sigma(n)\}$

determine the contribution of the item k to the fitness function. Formally, it is expressed as

$$c(\sigma, i) = \sum_{j=1}^{i-1} b_{\sigma(j),\sigma(i)} + \sum_{j=i+1}^{n} b_{\sigma(i),\sigma(j)} \tag{7.2}$$

Back to Example 7.2, due to the exchange of the pair $(14, 23)$, the contribution of item 2 has varied from 54 $(16 + 14 + 15 + 9)$ in e (Fig. 7.2a) to 63 $(16 + 23 + 15 + 9)$ in σ (Fig. 7.2b). In the case of item 3, its contribution has also increased, since the pair of $(14, 23)$ is associated to both items, 2 and 3. Inversely, in the cases of the items 1, 4 and 5, their contribution does not change from e to σ.

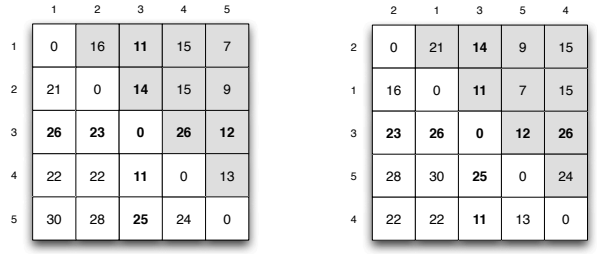
If we look carefully at Eq. 7.2, we realise that the contribution of item $\sigma(i) = k$ is not actually determined by the specific ordering of the items in the previous and posterior positions of i, but just by their grouping in these two sets of positions. As shown in Example 7.2, the contribution of the items 1, 4 and 5, does not change from e to σ since the grouping of the rest of the items into the previous and posterior sets of positions associated to the items 1, 4 and 5 was the same.

Proposition 7.1 *Given a solution σ, the contribution of the item $\sigma(i), i \in \{1, \ldots, n\}$, to the fitness function, $c(\sigma, i)$, is independent of the ordering of the previous items $\{\sigma(1), \ldots, \sigma(i-1)\}$ and of the ordering of the posterior items $\{\sigma(i+1), \ldots, \sigma(n)\}$.*

Example 7.3 This example illustrates how the contribution of item 3 is *independent* of the ordering of the previous and posterior sets of items. In Fig. 7.3a, the contribution of item 3, $c(e, 3)$ is 63 as a result of the sum (11+14+26+12). If we check the contribution of item 3 in σ (see Fig. 7.3b), we see that it also sums 63, even though the items $\{1, 2\}$ and $\{4, 5\}$ have swapped their positions.

In Proposition 7.1, we saw that the contribution of item k to the fitness function is independent of the ordering of the items in the previous and posterior sets. But, what happens if an item $\sigma(j) = l$ is moved from the previous set of items of $\sigma(i)$ to the posterior set of items? Contrarily to the previous case, the contribution of the item $\sigma(i)$, $c(\sigma, i)$ changes, and therefore, does not hold Proposition 7.1. At this point, it is worth remembering that each pair of entries $\{b_{\sigma(i),\sigma(j)}, b_{\sigma(j),\sigma(i)}\}$ in the matrix is associated to two items, $\sigma(i)$ and $\sigma(j)$, and thus, any exchange of location of $\sigma(i)$ by definition affects the contribution to the fitness function of $\sigma(i)$ and $\sigma(j)$. In fact, moving $\sigma(i)$ to position j, affects the contribution of all the items located between positions i and j. The example below illustrates the fitness variations produced by the movement of an item.

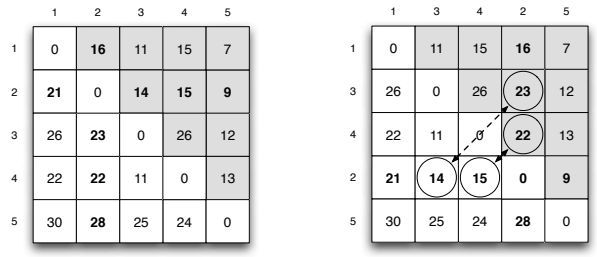
Example 7.4 Fig. 7.4a and Fig. 7.4b show matrix $\mathbf{B}$ according to solutions $e = 12345$ and $\sigma = 13425$. In this example we analyse the implications of moving item $e(2) = 2$ to position 4. Due to this modification, items 3 and 4,



(a) $e = 12345,\ c(e,3) = 63.$ (b) $\sigma = 21354,\ c(\sigma,3) = 63.$

Fig. 7.3: Illustration of the effect of swapping items at positions 1,2 and 4,5 with respect to the contribution of item 3 to the fitness function.

are shifted one position to the left, thus changing their contribution to the fitness function. Particularly, we observe that the pairs $\{14, 23\}$ and $\{15, 22\}$ associated to the items 2-3 and 2-4, have exchanged their positions. Therefore,



(a) $e = 1\mathbf{2}345,\ c(e,2) = 54,$
$c(e,3) = 63,\ c(e,4) = 69.$

(b) $\sigma = 134\mathbf{2}5,\ c(\sigma,2) = 70,$
$c(\sigma,3) = 72,\ c(\sigma,4) = 76.$

Fig. 7.4: Illustration of the effect of moving item 2 (from position 2 to position 4) to the contribution of the items 2, 3 and 4 to the fitness function. Numbers in bold denote the entries associated to item 2. Circled pairs of entries highlight exchanged entries.

the contribution of item 3, $c(e,3)$ changes from 63 $(11 + 14 + 26 + 12)$ to 72 $(11 + 26 + 23 + 12)$. Similarly, the contribution of item 4 changes from 69 $(15 + 15 + 26 + 13)$ to 76 $(15 + 26 + 22 + 13)$. And as regards item 2, its contribution also changes from 54 $(16 + 14 + 15 + 9)$ to 70 $(16 + 23 + 22 + 9)$.

Note that the variation in the fitness contribution of item 2 can be calculated as the sum of the variations of items 3 and 4.

7.4 The insert neighborhood and local optimality

As previously mentioned in the introduction, many of the most successful algorithms proposed for solving the LOP are partially or totally based on local search procedures. For that reason, we adopted the framework of local search algorithms in order to identify and extract meaningful information that could be used to improve their performance. It is well known that local search algorithms start from an initial solution and iteratively try to replace the current solution with a better one found in a previously defined neighbourhood system [17]. Among the different neighbourhood systems proposed in the literature for the LOP, most of the works [60, 155] clearly point to the *insert* neighbourhood system as the one that performs the best. For that reason, this is the system considered in this paper.

In what follows, we start by introducing some basic definitions about the insert neighborhood system and the local optimality of solutions.

Definition 7.1 *Two solutions σ and σ' are neighbors under the insert neighborhood (N_I) if σ' is obtained by moving an item of σ from position i to position j. It is formally defined as*

$$\sigma' \in N_I(\sigma) \Leftrightarrow \exists i, j \in \{1, \ldots, n\}, i \neq j \text{ s.t.}$$

$$\begin{cases} (\sigma'(z) = \sigma(z),\ z < i \wedge z > j) \wedge \\ (\sigma'(z) = \sigma(z+1),\ i \leq z < j) \wedge & \textit{when } i < j \\ \sigma'(j) = \sigma(i) \\ \\ (\sigma'(z) = \sigma(z),\ z < i \wedge z > j) \wedge \\ (\sigma'(z) = \sigma(z-1),\ i < z \leq j) \wedge & \textit{when } i > j \\ \sigma'(i) = \sigma(j) \end{cases}$$

When an insert operation is performed, that is to move item $\sigma(i)$ to position j, some entries in the upper triangle are exchanged with their respective pairs in the lower triangle, as we showed in Section 7.3. In the following lines, we distinguish two different insert operation scenarios, and we highlight in each case the specific entries that are exchanged:

- $i > j$ (see Fig. 7.5a). The entries $\{b_{\sigma(j),\sigma(i)}, b_{\sigma(j+1),\sigma(i)}, \ldots, b_{\sigma(i-1),\sigma(i)}\}$ inside the upper triangle (light line pattern cells) are moved to positions $\{(j+1, j), (j+2, j), \ldots, (i, j)\}$ in the lower triangle, and the entries $\{b_{\sigma(i),\sigma(j)}, b_{\sigma(i),\sigma(j+1)}, \ldots, b_{\sigma(i),\sigma(i-1)}\}$ outside the upper triangle (dark line pattern cells) are moved to positions $\{(j, j+1), (j, j+2), \ldots, (j, i)\}$ in the

upper triangle. The fitness value of σ' (neighbour of σ), $f(\sigma')$, can be calculated by summing to the fitness value of $f(\sigma)$ the difference of the entries exchanged, that is:

$$f(\sigma') = f(\sigma) + \sum_{z=j}^{i-1}(b_{\sigma(i),\sigma(z)} - b_{\sigma(z),\sigma(i)}) \tag{7.3}$$

- $i < j$ (see Fig. 7.5b). The entries $\{b_{\sigma(i),\sigma(i+1)}, b_{\sigma(i),\sigma(i+2)}, \ldots, b_{\sigma(i),\sigma(j)}\}$ inside the upper triangle (light line pattern cells) are moved to positions $\{(j,i),(j,i+1),\ldots,(j,j-1)\}$ in the lower triangle. Alternatively, the entries $\{b_{\sigma(i+1),\sigma(i)}, b_{\sigma(i+2),\sigma(i)}, \ldots, b_{\sigma(j),\sigma(i)}\}$ outside the upper triangle (dark line pattern cells) are moved to positions $\{(i,j),(i+1,j),\ldots,(j-1,j)\}$ in the upper triangle. Similarly to the previous case, the fitness value of σ', $f(\sigma')$ can be calculated as:

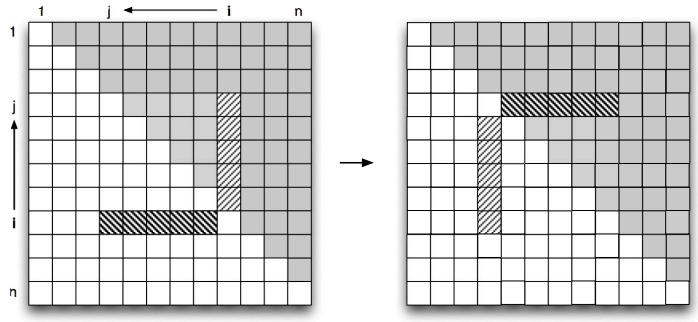
$$f(\sigma') = f(\sigma) + \sum_{z=i+1}^{j}(b_{\sigma(z),\sigma(i)} - b_{\sigma(i),\sigma(z)}) \tag{7.4}$$

As Fig. 7.5 illustrates, the pairs of entries in line pattern cells (dark and light), are the only entries that are exchanged because of the insert operation. The rest of the entries remain on the same side of the main diagonal as they were before the insert operation. As described in Section 7.3, every $\sigma(i)$ has associated $\{b_{\sigma(i),1}, b_{1,\sigma(i)}\}, \ldots, \{b_{\sigma(i),n}, b_{n,\sigma(i)}\}$ pairs of entries, and so we introduce the term *vector of differences* $(b_{\sigma(i),1} - b_{1,\sigma(i)}, \ldots, b_{\sigma(i),n} - b_{n,\sigma(i)})$ where each value in the vector describes the fitness variation that a specific pair of entries produces when it is exchanged because of a movement in $\sigma(i)$. In what follows, we will see how the vector of differences associated to each item will be essential in order to determine whether an item can produce local optima at a given position or not.

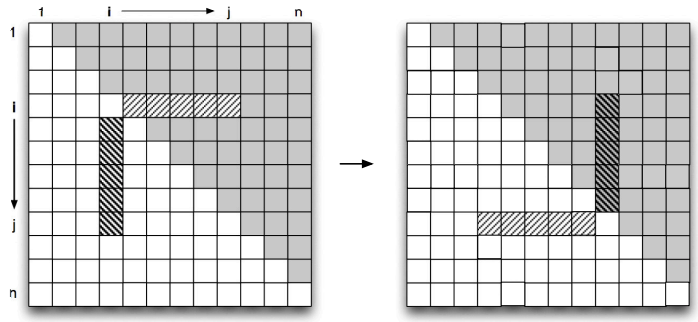
Definition 7.2 *A solution σ^* is a local optimum for the insert neighbourhood if all neighbouring solutions σ have a lower fitness value.*

$$\forall \sigma \in N_I(\sigma^*) \;\; f(\sigma^*) > f(\sigma)$$

Therefore, in the insert neighbourhood system, a solution is considered local optima if and only if among all the possible insert operations, there is no movement that outperforms the current solution. Taking into account what was exposed in Definition 7.1, given a solution σ and the neighbouring solution σ' obtained moving item k, $f(\sigma')$ can be computed by recalculating the fitness contribution of item k in the new position, and summing the variation to $f(\sigma)$. This way, we state that a solution is local optima in the insert neighbourhood if any insert operation performed over $\sigma(i)$, being $i = 1, \ldots, n$, does not increase the contribution of $\sigma(i)$.



(a) $i > j$. Before and after the insert operation.



(b) $i < j$. Before and after the insert operation.

Fig. 7.5: The exchange of entries produced by moving item $\sigma(i)$ to position j. Two different scenarios are considered: $i > j$ and $i < j$. Figures on the left illustrate the cells implicated in the insert operation. Figures on the right show the new arrangement of the entries because of the insert operation. Cells in grey denote the entries of the matrix that sum to the fitness function.

Example 7.5 Let us consider the entries associated to the item at position 2, $\sigma(2) = 2$ with $c(\sigma, 2) = 54$ (see Fig. 7.4). Fig. 7.6 illustrates the vector of differences $(b_{\sigma(1),2} - b_{2,\sigma(1)}, \ldots, b_{\sigma(n),2} - b_{2,\sigma(n)})$ of item 2 when performing all the possible insert operations. Besides, the contribution of item 2 is also given for each case.

As we can see, the insert operation that moves item 2 to position 5 is the one that maximises the contribution of this item, from 54 to 89. Note that in order to find the position at which the contribution of an item k is maximised, we need to find the arrangement of the vector of differences associated to item

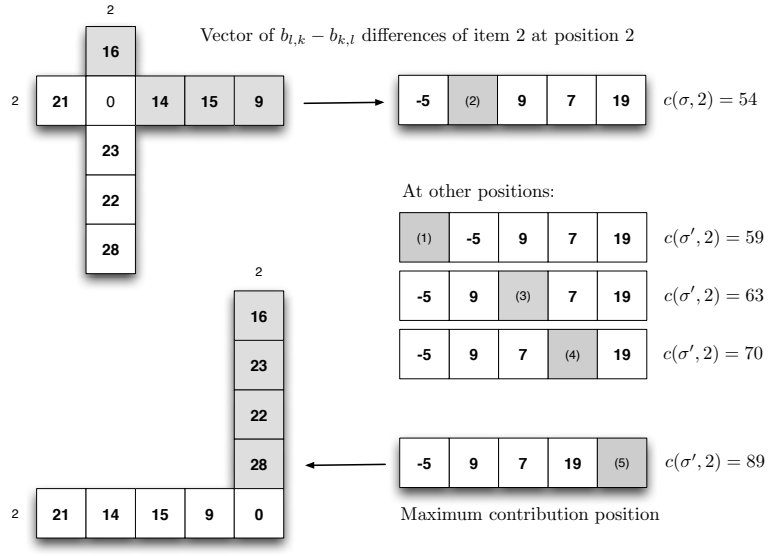


Fig. 7.6: The vector of differences of item 2, and the contribution of this item to the fitness function for each possible insert operation given a specific σ.

k such that the sum of the values in the positions $\{1, \ldots, i-1\}$ is maximised, and the sum of the values in the positions $\{i+1, \ldots, n\}$ is minimised[3].

The property below studies the arrangement of the vector of differences of each item when σ is a local optimal solution.

Property 7.1 *Given a local optimal solution σ^* for the insert neighbourhood then, for every item $\sigma^*(i)$, $i = 1, \ldots, n$, all the partial sums of the differences between the associated entries located before i are positive:*

$$\sum_{j=i-1}^{z} (b_{\sigma^*(j),\sigma^*(i)} - b_{\sigma^*(i),\sigma^*(j)}) \geq 0, \qquad z = i-1, \ldots, 1 \tag{7.5}$$

and all the partial sums of the differences between the associated pairs located after i are negative:

$$\sum_{j=i+1}^{z} (b_{\sigma^*(j),\sigma^*(i)} - b_{\sigma^*(i),\sigma^*(j)}) \leq 0, \qquad z = i+1, \ldots, n \tag{7.6}$$

[3] Note that the vector of differences is calculated by subtracting the entries in the column from the entries in the row.

Back in Example 7.5, it can be observed in Fig. 7.6 that only the insert operation that moves item 2 to position 5 organises the vector of differences in the way that complies with Eq. 7.5 and 7.6: all the partial sums from position 4 to 1 are positive: $19 \geq 0$, $7 + 19 \geq 0$, $9 + 7 + 19 \geq 0$, $-5 + 9 + 7 + 19 \geq 0$.

Due to the vector of differences induced by the ordering of the remaining items in σ, item 2 only complies with Eq. 7.5 and 7.6 at position 5. However, if we change the ordering of σ, the ordering of the vector of differences changes too, and thus, Eq. 7.5 and 7.6 might no longer hold. For illustrative purposes, let us consider a solution $\sigma' = 53421$ that has been obtained exchanging the position of the items 1 and 5 in σ. The vector of differences associated to item 2, according to σ', is $(19, 9, 7, *, -5)$[4], which indeed complies with Eq. 7.5 and 7.6. Moreover, in the case of σ', item 2 complies with the previous equations at either position 4 or position 5.

Nonetheless, there exist positions at which item 2 cannot generate a local optimum independently of the position of the remaining items, as is the case of the positions 1, 2 and 3. Looking (see Fig. 7.6) at the vectors of differences associated to item 2 in positions 1, 2 and 3, it can be seen that no ordering of the values in the vector complies with Eq. 7.5 and 7.6 at the same time.

In view of this property, in the next section, we propose a more efficient insert neighbourhood where we discard those insert operations that move items to positions at which they cannot generate local optimal solutions independently of the rest of the items.

7.5 The restricted insert neighborhood

In the previous section, we described the properties that the items within a solution σ need to comply with in order for σ to be a local optimum (Property 7.1). The next obvious step would consist of identifying the specific positions where the items generate local optimal solutions. An in-depth analysis in this sense, however, suggests that such an approach is NP-hard, since for each item at each position that complies with Property 7.1, we need to check (n-1)! permutations. Nevertheless, there are some positions at which items do not generate a local optima regardless of the ordering of the rest of the items, and contrary to the previous case, to detect these positions is straightforward.

Based on the vector of differences associated to the items, in this section we analyse the basis for discarding the positions at which items cannot produce local optimal solutions. As a result of the analysis, we propose the *restricted insert neighbourhood*, which discards some insert operations that move items to the positions where local optima are not generated.

In order to illustrate the process of identifying the positions where an item cannot generate a local optima, we start studying the trivial cases, i.e. the boundary cases at which an item k is located either first or last:

[4] * denotes the position at which item 2 was placed, the position 4.

- In order to discard the first position for item k, $\sigma(1) = k$, we need to demonstrate that no arrangement of the vector of differences associated to item k complies with Eq. 7.6. In order to prove that item k does not comply with that condition, independently of the order of the rest of items, it is enough to see that Eq. 7.6 is false when $\sigma(1) = k$ and $z = n$. Note that the result of the sum operator in the previous equation is independent of any ordering of the $\{\sigma(2), \ldots, \sigma(n)\}$ items, since the full vector of differences $(b_{\sigma(1),k} - b_{k,\sigma(1)}, \ldots, b_{\sigma(n),k} - b_{k,\sigma(n)})$ is considered in the sum as a result of $z = n$.
- Similarly to the previous case, in order to discard the last position for item k, $\sigma(n) = k$, we need to demonstrate that no arrangement of the vector of differences associated to item k complies with Eq. 7.5. In order to prove that item k does not comply with that condition, independently of the order of the rest of items, it is enough to see that Eq. 7.5 is false when $\sigma(n) = k$ and $z = 1$.

Beyond the boundary cases, in order to state whether an item k does not generate a local optima at a given position i, $\sigma(i) = k$, we need to check that a partition of items does not exist which locates a group of values of the vector of differences in the positions $\{1, \ldots, i-1\}$ and another group in $\{i+1, \ldots, n\}$, such that the arrangement of the vector complies with Eq. 7.5 and Eq. 7.6 at the same time.

In this regard, we propose a simple algorithm that starts sorting in descending order the vector of differences associated to the item k. Since our aim is to discard positions, the second step consists of checking whether the most favourable partitioning of the vector of differences complies with Eq. 7.5 and Eq. 7.6 when $\sigma(i) = k$. The allocation procedure consists of placing the largest value in the vector at position $i-1$, the second largest at position $i-2$ and so on. On the other hand, the lowest value is placed at position $i+1$, the next lowest at $i+2$, following the same procedure as for the largest values in the vector of differences. Let us denote as σ' the solution induced by the new ordering of the vector of differences. Then, item k does not generate a local optimal solution at position i if one of the following inequalities:

$$\sum_{z=i-1}^{1} (b_{\sigma'(z),k} - b_{k\sigma'(z)}) < 0 \text{ or } \sum_{z=i+1}^{n} (b_{\sigma'(z)k} - b_{k\sigma'(z)}) > 0 \qquad (7.7)$$

is true.

Extending this procedure to the whole set of items in σ for all the positions, we calculate a binary matrix, called *restrictions matrix* $\mathbf{R}$, where the entries with 0 represent the positions at which items do not produce local optimal solutions. The computational complexity of the algorithm used to calculate the *restrictions matrix* is, in the worst case, $O(n^3)$. Algorithm 7.1 summarises the pseudocode of the proposed algorithm.

```
1  input: The matrix B of entries.
2  for k = 1 to n
3     diffVector = CalculateVectorDifferences(k,B);
4     sortedDifferences = SortDescendingOrder(diffVector);
5     for i = 1 to n
6        beforeSum=0;
7        for z = i − 1 to 1
8           beforeSum+=sortedDifferences[z];
9        end for
10       afterSum=0;
11       for z = i + 1 to n
12          afterSum+=sortedDifferences[z];
13       end for
14       if (beforeSum≥ 0 and afterSum≤ 0)
15          R[k][i] = 1;
16       else
17          R[k][i] = 0;
18       fi
19    end for
20 end for
21 output: The restrictions matrix R.
```

Alg. 7.1: Pseudocode of the algorithm to compute the *restrictions matrix* $\mathbf{R}$.

Example 7.6 Fig. 7.7 illustrates the algorithm we propose to identify the positions where an item, in this example item 2, cannot generate a local optimal solution. As can be observed in Fig. 7.7, a solution is local optima if and only if item 2 is ranked at positions 4 or 5, since in the rest of the positions, even with the most favourable arrangement of the vector of differences, Eq. 7.7 is not complied. By repeating the same algorithm with the rest of the items, we obtain the restrictions matrix $\mathbf{R} = [r_{i,j}]_{n \times n}$ (see Fig. 7.8).

In view of the restrictions matrix in Fig. 7.8, solving the toy LOP example used throughout the paper is trivial, since some items can generate local optima only at one position, as is the case of items 1 and 5. In addition, once these items are fixed, items 2 and 3 are left with only one position, 2 and 4. Finally, item 4 must be placed in position 3. The global optimal solution for the example is $\sigma^* = 53421$.

For non-toy instances, the number of restricted positions is lower, but still significant. For this reason we propose to improve the performance of local search based methods by introducing a restricted version of the insert neighbourhood, called the *restricted insert neighbourhood*. This neighbourhood discards the insert operations that move items to positions at which they do not generate local optimal solutions.

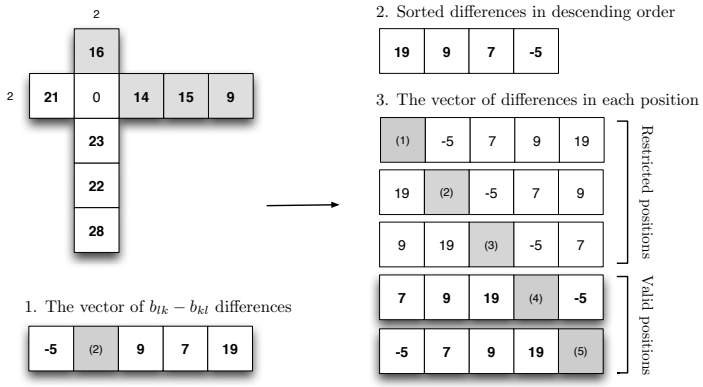


Fig. 7.7: The vector of differences associated to item 2, the sorted vector of differences, and the arrangement of the differences in order to identify the positions for which item 2 cannot generate local optimal solutions.

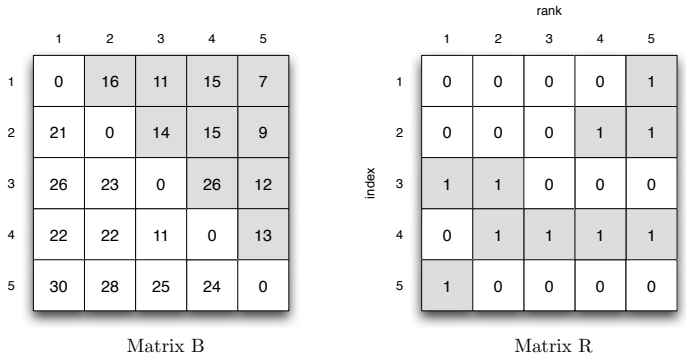


Fig. 7.8: Figure on the left shows the $n = 5$ instance used throughout the paper, and the corresponding restrictions matrix $\mathbf{R}$ is introduced on the right. Each position i, k of $\mathbf{R}$ indicates whether item k can generate a local optima when located at i; 1 if true, and 0 if not.

At first glance, to restrict some insert movements may alter the path followed by a greedy local search, since the neighbourhood operation under which the largest improvement is given could be discarded. However, in the case of the LOP, we discovered that the insert operation that is chosen in a greedy local search is never a restricted operation according to the matrix $\mathbf{R}$. The theorem below formalises this result.

Theorem 7.1 *Given a non local optimal solution σ, for every item $\sigma(i)$, $i = 1, \ldots, n$, the insert movement that maximises its contribution to the fitness function is not given in a restricted position of $\sigma(i)$.*

Proof In order to demonstrate that the theorem is held, we will prove that the inverse scenario cannot be true, i.e., let us assume that there exists an insert operation that moves an item $\sigma(i)$ to a restricted position j which improves the solution the most. This means that the maximum contribution of the item $\sigma(i)$ is given at position j. If the position that has the maximum contribution of item $\sigma(i)$ is restricted, then Property 7.1 should hold, implying that Eq. 7.7 is true for $\sigma(i)$ at position j. Therefore, position j cannot be restricted to item $\sigma(i)$ if it is the maximum fitness contribution position. □

As a result of Theorem 7.1, the restricted insert neighbourhood is a subsystem of the insert neighbourhood, which has some meaningful properties:

- Given a solution σ and the restrictions matrix $\mathbf{R}$, the size of the restricted insert based neighbourhood $N_R(\sigma)$ is reduced to

$$|N_R(\sigma)| = (n-1)^2 - \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} 1_{[r_{i,j}==0]}$$

- Due to the reduction of the neighbourhood, a greedy local search performed on this neighbourhood will check fewer insert operations than on the classical insert neighbourhood. Note that, as seen in Theorem 7.1, the restricted movements are useless, since they will never be the movements under which the largest improvement is made, and therefore, they will never be selected in a greedy search.
- When running a greedy local search on the classical and the restricted neighbourhoods, the solutions obtained at each step of the search are the same (both follow the same search path).

7.6 Experimentation

In order to demonstrate the improvement of using the *restricted insert neighbourhood*, we have applied our neighbourhood proposal to the best performing algorithms proposed in the literature: the Memetic Algorithm (MA) and the Iterated Local Search (ILS) (proposed in [155]). Both algorithms include an optimised implementation of a greedy local search algorithm for the LOP in their procedure. The goal in this experimentation is to analyse the improvement obtained using the restricted insert neighbourhood instead of the classical insert neighbourhood.

Due to the lack of challenging benchmarks in the literature, Schiavinotto and Stützle [155] proposed a new benchmark, the *extended* LOLIB (xLOLIB),

which was generated by randomly sampling the instances of the LOLIB benchmark. Particularly, they generated 39 instances of size 150 and 39 instances of size 250. In addition to these instances, we generated an extra benchmark to include in this experimentation, xLOLIB2, with 200 instances of sizes 300, 500, 750 and 1000 (50 instances of each size) following the same procedure used for generating the xLOLIB benchmark.

Both source codes, MA and ILS, were obtained from the authors, and so the restricted insert neighbourhood was directly implemented on the original code (written in C), adding only the necessary code to calculate the restrictions matrix and implement the restricted insert neighbourhood. The experimentation was conducted on a cluster of 20 nodes, each of them equipped with two Intel Xeon X5650 CPUs and 48GB of memory.

7.6.1 Performance comparison based on the quality of solutions obtained for a given number of evaluations

In order to analyse the contribution of the restricted insert neighbourhood as fairly as possible, we ran the original implementations of MA and ILS, and their *restricted* versions MA_r and ILS_r for three different maximum numbers of evaluations, $1000n^2$, $5000n^2$ and $10000n^2$ (n denotes the size of the instance). The evaluation numbers were set without performing any previous experimentation.

Each *algorithm-instance* pair was run 20 times and the average fitness of the best solutions obtained was calculated. Due to the large size of the results-tables obtained from the conducted experiments, the results have been summarised in Table 7.1, divided in three groups according to the different stopping criterion and the size of the instances. Besides, the results have been presented as the number of times the restricted versions of the algorithms beat the classical versions. Next to the results, within parentheses, we show the number of instances for which the same results on both versions, restricted and non-restricted, were obtained[5]. Remember that the restricted versions will at least equal the results of the classical proposal.

In view of the results, the restricted algorithms, MA_r and ILS_r, outperform the classical implementations in almost all the evaluated cases. With respect to MA_r, it outperforms MA in $81.2\%, 94.6\%$ and 95.3% of the instances for $1000n^2, 5000n^2$ and $10000n^2$ maximum numbers of evaluations. Similarly, ILS_r outperforms ILS in $97.4\%, 94.2\%$ and 88.4% of the instances.

In order to assess whether there exist statistical differences among the results, we applied a nonparametric Wilcoxon test to the average results obtained by the pair MA - MA_r, and the pair ILS - ILS_r for each size of the instances. A level of significance $\alpha = 0.05$ was set. The statistical test reported

[5] Supplementary results, original source codes, instances, and extended material of the experiments can be obtained from `http://www.sc.ehu.es/ccwbayes/members/jceberio/LOP.html`.

Table 7.1: Comparison of the results of MA with MA_r, and ILS with ILS_r for the xLOLIB and xLOLIB2 benchmark LOP instances (278 instances). The results have been grouped with respect to the maximum number of evaluations used, i.e. $1000n^2, 5000n^2$ and $10000n^2$.

$1000n^2$ evals.	xLOLIB		xLOLIB 2				Total
	150	250	300	500	750	1000	
MA_r vs. MA	35 (4)	31 (8)	39 (11)	43 (7)	41 (9)	37(13)	226 (52)
ILS_r vs. ILS	37 (2)	37 (2)	49 (1)	48 (2)	50 (0)	50(0)	271 (7)

$5000n^2$ evals.	xLOLIB		xLOLIB 2				Total
	150	250	300	500	750	1000	
MA_r vs. MA	37 (2)	39 (0)	50 (0)	49 (1)	44 (6)	44(6)	263 (15)
ILS_r vs. ILS	38 (1)	36 (3)	50 (0)	45 (5)	46 (4)	47(3)	262 (16)

$10000n^2$ evals.	xLOLIB		xLOLIB 2				Total
	150	250	300	500	750	1000	
MA_r vs. MA	39 (0)	34 (5)	43 (7)	50 (0)	50 (0)	49(1)	265 (13)
ILS_r vs. ILS	33 (6)	37 (2)	46 (4)	42 (8)	43 (7)	45(5)	246 (32)

significant differences between the algorithms for all the sets of instances and for the three maximum number of evaluations. The p-values obtained for the pair MA - MA_r were in the worst cases 1.23×10^{-06}, and 5.6×10^{-07} for ILS - ILS_r.

7.6.2 Performance comparison based on execution time

In addition to evaluating the performance of the algorithms in terms of the quality of the solutions obtained for a fixed number of evaluations, we have also carried out an evaluation in terms of running time. Basically, we measure the computation time spent by both pairs of algorithms, MA - MA_r and ILS - ILS_r, to perform c iterations[6] (recall that both approaches, the classical and the restricted, follow the same path, and therefore, after c iterations the same solution will be reached). Particularly, we compare the computation time of the basic approaches with respect to the computation time of the restricted approaches that include: (1) the computation of the restrictions matrix at the beginning of the algorithm, and (2) a greedy local search under the restricted insert neighbourhood which, for each item, reads in the restrictions matrix the range of the positions to which the item can be moved.

It is clear that the execution time needed to perform a given number of iterations with the restricted approaches, ILS_r and MA_r, depends on the sparsity

[6] As regards MA, we consider as iteration a generation of the algorithm, while in the ILS, we count an iteration every time that the algorithm reaches a local optimum and applies the perturbation procedure.

of the restrictions matrix associated to each particular instance. That is, the higher the number of zero-valued entries, the lower the checked neighbours, and therefore, the faster the greedy local search. In order to obtain a more general picture of the influence of the sparsity in the execution time, in this experiment, apart from the previously used benchmarks, xLOLIB and xLOLIB2, we have included additional instances with different sparsities. Particularly, we have considered the most common benchmarks for the LOP: LOLIB, LMC and MB. The number of iterations was set to $c = 10000$ without any previous experimentation. Results are displayed in Figures 7.9 and 7.10.

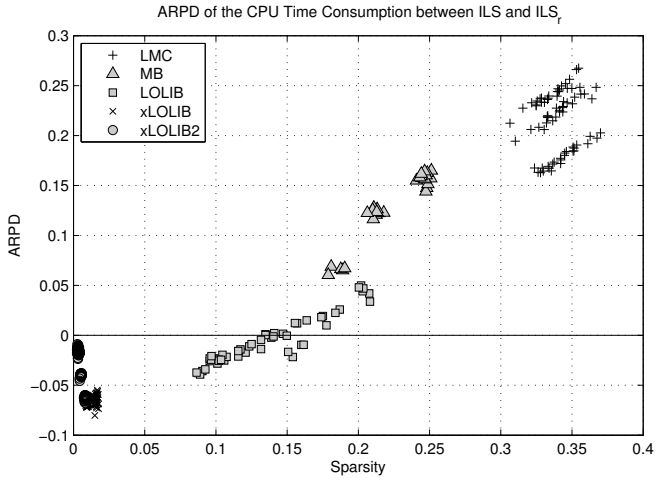


Fig. 7.9: ARPD of the CPU time consumption between ILS and ILS$_r$. Positive ARPD results indicate that ILS$_r$ is faster, while negative values point out that ILS is faster.

From these figures, we can observe that sparsity varies between (almost) zero and 0.37. When sparsity is about 0.15 or higher, our restricted approaches run faster than the original versions (including the calculation of the restrictions matrix). In the best cases (highest sparsities), ILS$_r$ is 27% faster than ILS (22% in the case of MA$_r$). In the worst case, ILS$_r$ runs 8% slower than ILS (7% for MA$_r$). Therefore, in view of the results, in most of the instances, our restricted versions require shorter execution times than the original versions.

7.7 Conclusions and future work

In this chapter we have introduced a detailed theoretical study of the LOP in the context of local search algorithms. Based on this study we presented

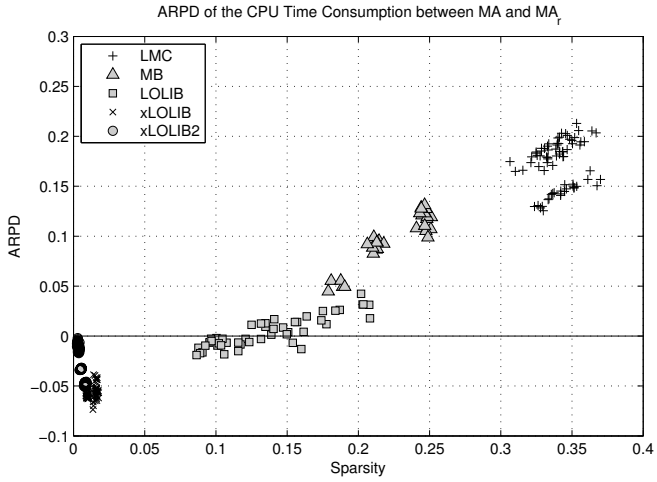


Fig. 7.10: ARPD of the CPU time consumption between MA and MA_r. Positive ARPD results indicate that MA_r is faster, while negative values point out that MA is faster.

a method that allows to extract static information about the problem and incorporate it to improve, for example, the performance of local search algorithms. Particularly, we developed a method to detect the positions where the items cannot appear in local optimal solutions of the insert neighbourhood. As a result of this study, an improved version of the insert neighbourhood system, called *restricted insert neighbourhood* was proposed, in which the insert operations that lead items to restricted locations are discarded.

In order to demonstrate the efficiency of the restricted insert neighbourhood, we applied this neighbourhood to the best performing state-of-the-art algorithms for the LOP: the Memetic Algorithm and the Iterated Local Search. Average fitness values of the best solutions from 20 repetitions of MA, ILS, MA_r, and ILS_r were calculated for three different maximum numbers of evaluations on a benchmark of 278 instances.

Conducted experiments showed that the restricted version of the algorithms systematically outperforms the classical versions when a maximum number of evaluations is set as stopping criterion. From 278 instances tested (xLOLIB - 78 instances) and (xLOLIB2 - 200 instances), MA_r outperformed MA in 90% of the cases, and ILS_r improved ILS in 93.3%. The Wilcoxon statistical test confirmed the behaviour reported by the experiments, indicating that MA_r and ILS_r are significantly better than the classical version of the algorithms for all the studied sets of instances.

In addition, the execution time of the restricted and the classical algorithms was measured. The results reported showed that, in most of the cases,

the restricted versions run faster than their classical counterparts. In particular, in the best cases, ILS_r and MA_r were 27% and 22% faster than the classical approaches. Alternatively, results showed that, for instances with almost no sparsity, ILS_r and MA_r are in the worst cases $7 - 8\%$ slower.

Studying a problem and extracting information that can be used to guide the optimisation process of an algorithm is an interesting task that, in most of the cases, requires thorough research. In this work, we just scratched the surface of the linear ordering problem with the *restricted insert neighbourhood*, and therefore, there are many issues that deserve deeper analysis. An obvious extension of the theoretical study presented in this chapter is that of the relative ordering of the items. For instance, it is relatively easy to prove that certain pairs of items, due to the associated entries that they share, cannot generate local optima when located in consecutive positions.

Finally, how to exploit the extracted knowledge in the most advantageous way is another challenging task that, in this case, has been addressed by discarding solutions within a neighbourhood, and proposing a restricted version of the neighbourhood. However, taking into account that the restrictions matrix partially describes the global optimum solution, such information could be used in exact, heuristic and metaheuristic algorithms in order to guide the search towards better solutions. Particularly, we find it interesting to include the restrictions matrix to Branch and Bound algorithms in order to discard the branches that do not comply with the restrictions. Alternatively, the implementation of constructive heuristics to initialise metaheuristic procedures, or the implementation of guided crossover and mutations within Evolutionary Algorithms are other interesting applications to explore.

8

Understanding instance complexity in the LOP

8.1 Introduction

As stated previously, Garey and Johnson [63] demonstrated that the linear ordering problem is an *NP-hard* problem, which means that there is no known algorithm able to solve up to optimality all LOP instances in polynomial time. However, as seen for most of the combinatorial optimization problems, the difficulty of solving a given instance is not limited to the size of the instance, but other additional (and sometimes unknown) parameters influence too. In the particular case of the LOP, it easy to find an instance of size 100 where the optimal is known, while instances of size 50 are still to be solved.

In this regards, the research community has also tried to identify the aspects of the problem that influence its optimisation for specific algorithms, or alternatively, extract information from the problem that could be used to guide the search. For instance, Schiavinotto and Stutzle [155] analyzed LOP instances from different benchmarks, extracting features that were used later to rank the benchmarks according to their difficulty for local search schemes, and proposed new algorithms for optimizing the LOP.

In this chapter, we investigate in this direction trying to identify features or characteristics of the LOP instances that influence the complexity of solving them with local search type algorithms. Particularly, we introduce two new metrics: *insert ratio* and *ubiquity ratio* that give a measure of the difficulty of achieving the optimal solution with an insert neighborhood system.

In this chapter, we will use the definition and notation about the LOP introduced in Chapter 7. The rest of the chapter is structured as follows: in the following section, the instance complexity of the LOP is studied by introducing two new metrics: the insert ratio and ubiquity ratio. Some experiments are run in order to evaluate the validity of the proposed metrics, afterwards, in Section 8.3. Finally, some conclusions and ideas for future work are presented in Section 8.4.

8.2 Analysis of complexity

As previously mentioned, in this chapter we aim to study the properties of the instances that affect the complexity of local search type algorithms. Since many works in the literature [60, 155] clearly state that the insert neighborhood is the most appropriate system for solving the LOP with a local search algorithm, our study will be carried out for this neighborhood.

In the following lines, the *insert ratio* and *ubiquity ratio* are introduced in detail. The examples, notations and equations have been set following the contributions of the previous chapter.

8.2.1 Insert Ratio

The insert ratio measures the intercalation between the entries of the column and of the row associated to an item. It is calculated as follows:

1. Put all row and column entries associated to an item i in a vector, and order these numbers from the highest to the lowest.
2. For each pair of consecutive column entries calculate the distance in the vector.
3. Repeat the same process as before for the entries belonging to the row.
4. The insert ratio is calculated by summing the obtained distances for the column entries and the row entries, and dividing by $n-1$.

The overall insert ratio of the instance is calculated by averaging the insert ratio of each of the items. Fig. 8.1 illustrates the insert ratio calculated for item 4 of the matrix.

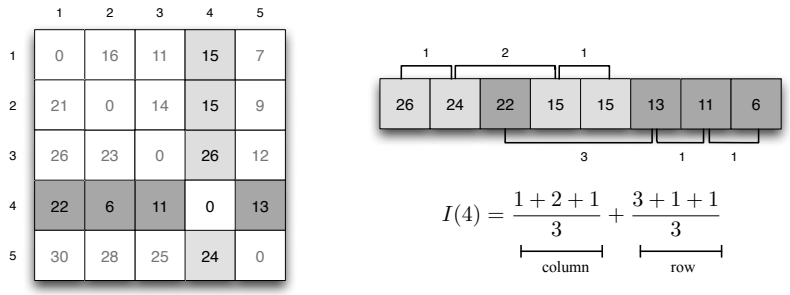


Fig. 8.1: Insert ratio of item 4.

Given a specific item i, when all the entries of the column i are higher than the entries of the row i, the item i can only generate local optima in the last position of σ, and only in the first position when the opposite scenario occurs. The items that agree with the exposed cases receive the lowest possible

ratios, suggesting that this type of items are easy to locate. When the overall insert ratio associated to an instance is low, the instance is supposed to be easy to solve. On the contrary, a high insert ratio suggests that the items are more susceptible to generate local optima in several positions of σ, and thus, the difficulty of the instance is higher. Note that when we mention the complexity we refer to the difficulty of the instance to be solved with a local search algorithm based on a specific neighbourhood system.

8.2.2 Ubiquity Ratio

The ubiquity ratio measures the percentage of locations where the items are susceptible to generate local optima using the insert neighborhood. An item may produce local optima in a certain position if all the possible insert operations over the item decreases its contribution. Note that for each item i, the entries $b_{i,j}$ and $b_{j,i}$ for $j = 1, \ldots, n$ always appear in symmetric positions with respect to the main diagonal, i.e., they never appear together in the upper triangle. Therefore, an item i may generate a local optima when it is located in position j, i.e., $\sigma(j) = i$, if there exists an ordering σ such that the following inequalities holds.

$$\sum_{z=1}^{j-1} b_{\sigma(z),\sigma(j)} - b_{\sigma(j),\sigma(z)} > 0 \wedge \sum_{z=j+1}^{n} b_{\sigma(z),\sigma(j)} - b_{\sigma(j),\sigma(z)} < 0$$

In order to determine the positions at which the items may produce local optima, we count the positions where the item located at position j cannot produce local optima, that is to check whether the sum of all the differences in positions $\{1, \ldots, j-1\}$ is negative and the sum of all the differences in positions $\{j+1, \ldots, n\}$ is positive. The ubiquity ratio of an item is calculated by dividing the number of positions where the item may generate local optima, by the total number of positions. The global ubiquity ratio is then calculated by averaging the ratios of each of the items. Fig. 8.2 illustrates the procedure of calculating the ubiquity ratio of item 4.

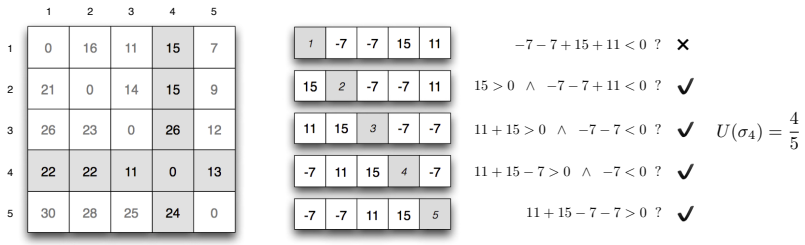


Fig. 8.2: Ubiquity ratio of item 4.

8.3 Experimentation

In order to confirm the validity of the proposed metrics, we introduce two different sets of experiments. Additionally, two artificially generated benchmarks of instances are used.

8.3.1 Benchmark instances.

540 instances of sizes $\{8, 9, 10, 11, 12, 13\}$ (benchmark A) and 600 instances of sizes $\{20, 30, 40, 50, 60, 70\}$ (benchmark B) have been generated according to the following procedure. We start by generating the entries associated to the first item, then the second, and so on. Note that, when we generate the entries for the first item, we have also indirectly generated some of the entries for the rest of the items, and the same for the second item, and so on. In order to generate the entries for item i, we sample uniformly at random a vector of $2(n-i-1)$ entries in the range [0,999]. Next, we order these entries. After that, following the order of the entries, we make groups of size t (t is a parameter of the procedure that ranges from 1 to 10). Then, we uniformly at random decide where, column or row, to place each pair of groups (one to the row and the other to the column).

8.3.2 Insert and ubiquity ratios vs. estimated number of local optima

We analyze the correlation of the insert and ubiquity ratios with respect to the estimated number of local optima for the benchmark B of LOP instances. Following the recommendations of a recently published review on local optima estimation methods [75], we have chosen two methods to estimate the number of local optima of the instances: *ChaoBunge* and *ChaoLee2*.

Fig. 8.3 and 8.4 introduce the results for the proposed metrics according to the average of 10 repetitions of *ChaoLee2* and *ChaoBunge*. Results show a good correlation between the *ubiquity* and *insert* ratios and the number of local optima. In fact, the higher the value of the metric, the higher the number of local optima.

8.3.3 Number of evaluations required to achieve optimal solutions

In this experiment, we run a muti-start local search greedy algorithm (MLS) with the insert neighborhood, and we compute the number of evaluations needed to achieve the optimal solution for the benchmark A instances.

Fig. 8.5 and 8.6 introduce the results for the proposed metrics according to the average of 10 repetitions of the MLS. Although not as explicitly as in the previous case, results show that for the highest *ubiquity* and *insert* ratios, the variance of the number of performed evaluations increases. This is clearer for instances of size $\{11, 12, 13\}$.

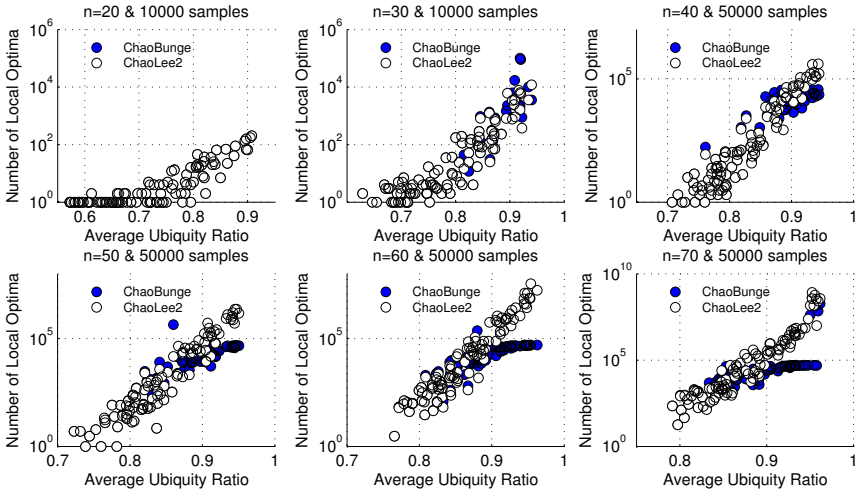


Fig. 8.3: Estimated number of local optima in relation to the average *ubiquity* ratio.

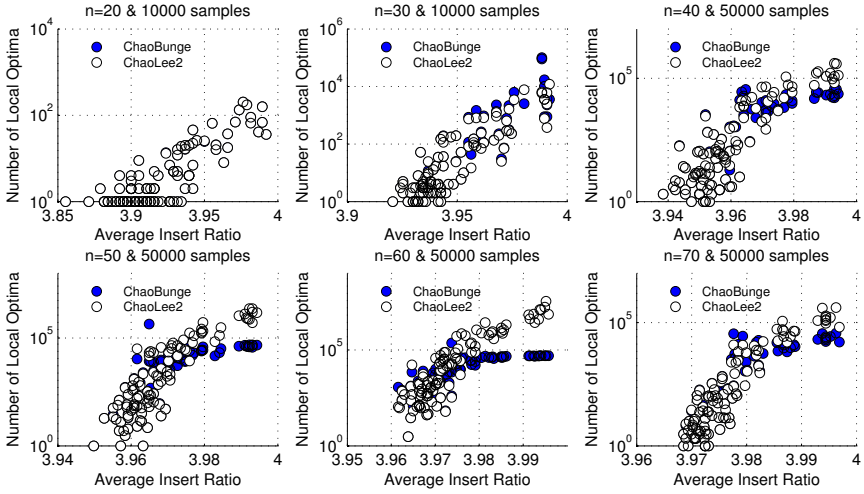


Fig. 8.4: The estimated number of local optima in relation to the average *insert* ratio.

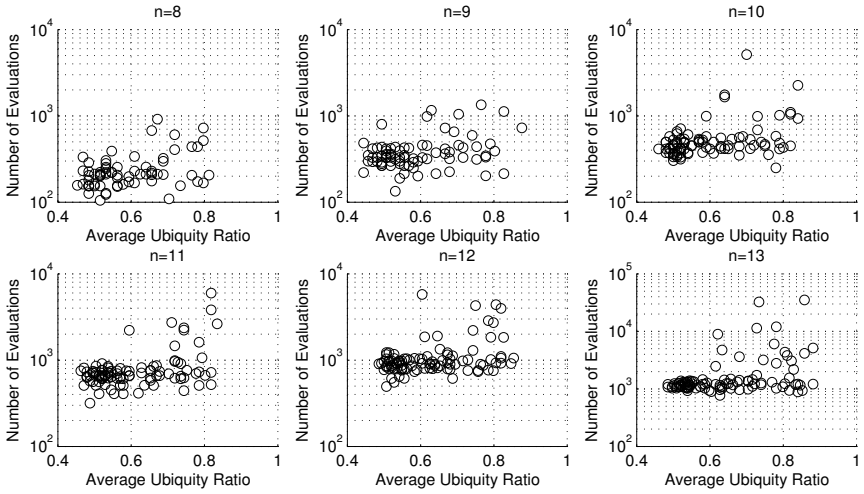


Fig. 8.5: Average number of evaluations performed by MLS in relation to the average *ubiquity* ratio.

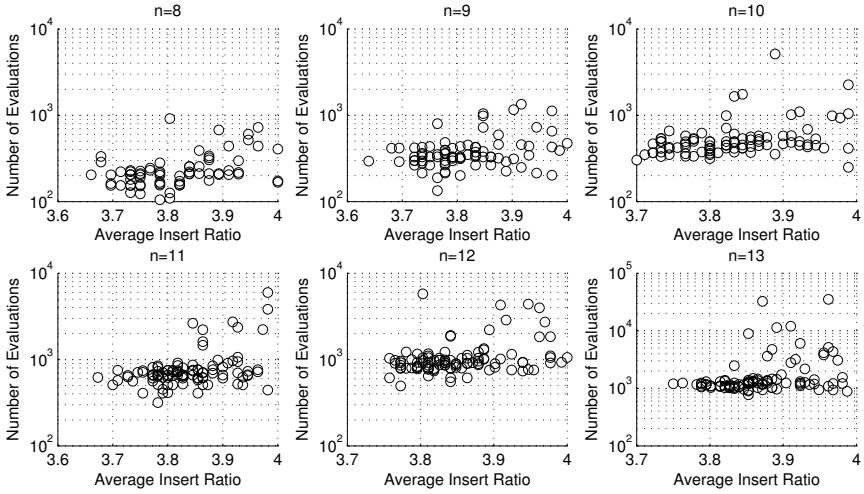


Fig. 8.6: Average number of evaluations performed by MLS in relation to the average *insert* ratio.

8.4 Conclusions and future work

In this chapter, a preliminary analysis of the instance complexity of the LOP was presented. Particularly, we focused on identifying the characteristics of the instances that influence their complexity when being solved by local search type algorithms with the insert neighborhood. Characterizing the conditions required by each item to generate local optima for the insert neighborhood, we presented two new metrics, *insert ratio* and *ubiquity ratio*. Conducted experiments showed a correlation between the proposed metrics and the complexity of solving an artificially generated set of LOP instances with multistart local search algorithms.

The proposed metrics took into account univariate information of the items, however, as future work, it would be interesting to extend these metrics to bivariate or multivariate information associated to the items. Alternatively, the influence of ties of pairs of entries to the number of local optima should be studied, which, at first sight, introduce a high redundancy in the search space.

Part III

Spectral landscape theory on permutation problems

9

Multi-objectivizing the quadratic assigment problem by means of an elementary landscape decomposition

9.1 Introduction

A combinatorial optimisation problem consists of finding an optimal solution of a function,

$$\begin{aligned} f : X &\longrightarrow \mathbb{R} \\ x &\longmapsto f(x) \end{aligned}$$

such that the search space X is a finite or countable infinite set. Usually f is considered as a single-objective function. However, in many real-world problems, the optimisation process may involve multiple objectives (functions) simultaneously. These problems are known as multi-objective optimisation problems (MOPs). Formally, MOPs can be formulated as [192]:

$$\text{maximize} \quad F(x) = [f_1(x), \ldots, f_m(x)] , \quad x \in X \tag{9.1}$$

where $F : X \longrightarrow \mathbb{R}^m$ consists of m real-valued objective (fitness) functions and $\mathbb{R}^m$ denotes the *objective space*.

During the last decades, multi-objective evolutionary algorithms (MOEAs), such as *Non-Dominated Sorting Genetic Algorithm* [50, 159], *Strength Pareto Evolutionary Algorithm II* [199] and *Multi-objective Evolutionary Algorithm based on Decomposition* [192], have demonstrated their competitiveness when solving MOPs. Considering the unique ability of MOEAs to enhance the diversity of the population, authors have claimed that multi-objective algorithms might be helpful for single-objective optimisation as well [1]. In fact, according to a recent survey on MOEAs for single-objective optimisation [157], a number of papers [91, 132, 154] have proposed transforming single-objective problems into MOPs by transforming their fitness landscapes. This procedure, known as *multi-objectivization*, was first used by Knowles *et al.* [91], where the authors distinguished between two types of schemes: *decomposition* and *aggregation*. The first scheme proposes decomposing the original function f into several components in such a way that the original optimum is in the

Pareto optimum in the new formulation. The second scheme, instead, considers some additional objectives that are used in combination with the original function f.

Papers on this topic [72, 91] have demonstrated that multi-objectivization by decomposition increases the plateau regions of incomparable solutions in number and size, which might negatively influence the search. On the contrary, they have also showed that the introduction of plateaus reduces the number of local optima, which should ease the search in some cases.

In this chapter, we present a novel scheme to multi-objectivize single-objective problems into MOPs by means of the decomposition of the fitness function. Particularly, we propose using *elementary landscape decomposition* techniques in order to decompose the fitness function f as a sum of a set of elementary functions.

In landscape theory, *elementary landscapes* [163] are a class of landscapes whose main characteristic is that they can be modelled using the Groover's [69] *wave equation* (see Eq. 9.2). Based on this equation, it is possible to compute the average value of the fitness function in the neighbourhood of a solution with a closed form expression. Moreover, it also permits to compute landscape ruggedness measures more accurately [7]. Unfortunately, for most of the combinatorial optimisation problems, with the exception of the traveling salesman problem [187], and a few others, we do not know a neighbourhood system that is able to produce an elementary landscape.

In the same research line, further research by Chicano *et al.* [40] demonstrated that, for any fitness landscape, if the neighbourhood system has certain characteristics, then it is possible to decompose f as a sum of some elementary landscapes. And thus, f can be expressed as:

$$f(x) = \sum_{i=1}^{t} f_i(x)$$

where t is the number of landscapes in the decomposition, and f_i is the function related to the i-th landscape.

Works on elementary landscape decomposition have shown that the landscape produced by the MAX-k-SAT under the Hamming neighbourhood is a superposition of k elementary landscapes [166]. Similarly, [40, 41] demonstrated that the subset sum problem and the frequency assignment problem under this neighbourhood are superpositions of two elementary landscapes. In addition, Chicano *et al.* [39] demonstrated that the general case of the Quadratic Assignment Problem (QAP) under the *interchange* neighbourhood[1] is at most the superposition of three elementary landscapes [39].

In order to illustrate the multi-objectivization scheme proposed in this chapter, we will consider the QAP as a case study. As aforementioned, the

[1] The *interchange* neighbourhood considers that two solutions (permutations) are neighbours if one is obtained by interchanging two elements in the other.

landscape produced by the general formulation of the QAP (see Eq. 9.4) under the interchange neighbourhood can be decomposed at most as a sum of three elementary landscapes. However, when translating this decomposition to the standard formulation of the QAP (the one introduced in Chapter 1), only two of the elementary landscapes are meaningful since the third one is a constant function. As a result, we propose to reformulate the QAP as a 2-objective problem, one for each elementary landscape in the decomposition.

For the sake of demonstrating the validity of this multi-objectivization method, we compare the NSGA-II on the multi-objectivized QAP with its single-objective version genetic algorithm (SGA). The performed experiments on two different benchmarks (random and real-life like) show that solving the QAP in its 2-objective form obtains better results than with the single-objective approach.

The remainder of the chapter is organised as follows: in Section 9.2, the elementary landscape decomposition of the QAP is described. Section 9.3 presents a modified version of the NSGA-II for the 2-objective approach of the QAP. In Section 9.4, we compare the performance of NSGA-II and SGA on two benchmarks: random and real-life like. Finally, general conclusions and ideas for future work are summarised in Section 9.5.

9.2 Elementary landscape decomposition of the QAP

In combinatorial optimisation, a *fitness landscape* is understood as a triple $(\mathbb{S}_n, N, f)$ where $\mathbb{S}_n$ denotes the search space of solutions, $f : \mathbb{S}_n \to \mathbb{R}$ defines the fitness function, and the neighbourhood operator N assigns a set of neighbouring solutions $N(\sigma) \in \mathbb{S}_n$ to each solution σ.

In 1992, Grover [69] and Codenotti and Margara [45] demonstrated that some fitness landscapes arising from certain classes of combinatorial optimisation problems could be modelled using the so called Groover's *wave equation*:

$$\underset{\pi \in N(\sigma)}{avg} \{f(\pi)\} = f(\sigma) + \frac{k}{|N(\sigma)|}(\bar{f} - f(\sigma)) \tag{9.2}$$

where $\bar{f}$ denotes the average fitness value of all the solutions in the search space, $|N(\sigma)|$ is the size of the neighbourhood, and k is a constant value. Furthermore, Groover demonstrated that if a landscape satisfies Eq. 9.2, then all the local maxima solutions are greater than $\bar{f}$ and all the local minima are lower than $\bar{f}$. This class of landscapes was called by Stadler [161] *elementary landscapes*. According to Stadler, a landscape $(\mathbb{S}_n, N, f)$ is elementary if the function f is an eigenvector of the Laplacian matrix Δ that describes the neighbourhood N up to an additive constant k.

In general, an arbitrary landscape is not elementary. However, Chicano *et al.* [40] proved that any landscape can be decomposed as a sum of elementary landscapes if the neighbourhood system considered is *regular* ($|N(\sigma)| = d > 0$,

for all $\sigma \in \mathbb{S}_n$) and *symmetric* (for all $\sigma, \pi \in \mathbb{S}_n, \pi \in N(\sigma) \iff \sigma \in N(\pi)$). As the authors stated, we know that, if a square real matrix Q of size l is symmetric, then there exists an orthogonal basis in the vector space $\mathbb{R}^l$ that is composed of eigenvectors of Q. And thus, every vector of $\mathbb{R}^l$ can be written as the weighted sum of the vectors in the orthogonal basis. Since the Laplacian matrix Δ is defined as a symmetric square real matrix, then it can be deduced that there exists an orthogonal basis of eigenvectors associated to it. As a consequence, f can be decomposed as the weighted sum of a set of elementary functions.

As previously mentioned, Chicano *et al.* [39] demonstrated that the landscape produced by the general formulation of the QAP (see Eq. 9.4) under the interchange neighbourhood can be written as a superposition of three elementary landscapes. Since this decomposition is an essential part of this work, in this section we provide a general overview. The explanation below has been extracted from [39, 40], and thus, for proofs and detailed descriptions, we recommend the interested readers to address the original works.

In order to analyse the elementary components of the fitness function of the QAP, it is useful to separate the instance-related part and the problem-related part. Therefore, we start the decomposition procedure by rewriting the fitness function of the QAP (see Chapter 1) in this way:

$$f(\sigma) = \sum_{i,j=1}^{n} \sum_{p,q=1}^{n} d_{i,j} h_{p,q} \delta^{p}_{\sigma(i)} \delta^{q}_{\sigma(j)} \tag{9.3}$$

where $\delta^{p}_{\sigma(i)}$ denotes the Kronecker's delta and returns 1 if $\sigma(i) = p$, and 0 otherwise. At this point, we extend the decomposition of f to a more general function g at which the instance-related part of Eq. 9.3, the product $d_{i,j} h_{p,q}$, is replaced with a new variable ψ_{ijpq}. Alternatively, the problem-related part is rewritten as the parameterised function $\varphi_{(i,j)(p,q)}(\sigma) = \delta^{p}_{\sigma(i)} \delta^{q}_{\sigma(j)}$. Thus, the generalised QAP function is defined as:

$$g(\sigma) = \sum_{i,j,p,q=1}^{n} \psi_{ijpq} \varphi_{(i,j)(p,q)}(\sigma) \tag{9.4}$$

where Eq. 9.3 is a particular case in which $\psi_{ijpq} = d_{i,j} h_{p,q}$.

As far as we know, g does not produce elementary landscapes. However, as the interchange neighbourhood is regular and symmetric, it is possible to find an orthogonal basis composed of elementary functions in which to decompose g. To this end, Chicano *et al.* [39] focus exclusively on the decomposition of the problem-related part, since any result on $\varphi_{(i,j)(p,q)}$ can be extended to any linear combination of it, and subsequently to g.

First, we distinguish two cases of the function φ under the interchange neighbourhood: 1) $i = j \wedge p = q$ and 2) $i \neq j \wedge p \neq q$ [2]. We rewrite Eq. 9.4 as follows:

$$g(\sigma) = \sum_{i,p=1}^{n} \psi_{iipp}\varphi_{(i,i)(p,p)}(\sigma) + \sum_{\substack{i,\,j,\,p,\,q\,=\,1 \\ i \neq j \\ p \neq q}}^{n} \psi_{ijpq}\varphi_{(i,j)(p,q)}(\sigma) \tag{9.5}$$

When $i = j \wedge p = q$, we can easily prove that the function φ is elementary and complies with Eq. 9.2 (when $k = n$) by demonstrating that the equation below holds for two constants a and b:

$$\underset{\pi \in N(\sigma)}{avg}\ \{\varphi(\pi)\} = a\varphi(\sigma) + b \tag{9.6}$$

Nonetheless, it is not elementary when $i \neq j \wedge p \neq q$. In order to decompose it as a sum of elementary functions, in [39] the authors introduced a set of some auxiliary functions that provide the required orthogonal basis in the decomposition. Particularly, the authors demonstrate that the right-hand term $\varphi_{(i,j)(p,q)}$ (see Eq. 9.5) can be decomposed as the sum of three elementary functions/landscapes. As a result, the fitness function of the general QAP, g, is finally decomposed as follows:

$$\begin{aligned} g(\sigma) = & \sum_{i,p=1}^{n} \psi_{iipp}\varphi_{(i,i)(p,p)}(\sigma) \\ & + \sum_{\substack{i,\,j,\,p,\,q\,=\,1 \\ i \neq j \\ p \neq q}}^{n} \psi_{ijpq} \left(\frac{\Omega^1_{(i,j)(p,q)}(\sigma)}{2n} + \frac{\Omega^2_{(i,j)(p,q)}(\sigma)}{2(n-2)} + \frac{\Omega^3_{(i,j)(p,q)}(\sigma)}{n(n-2)} \right) \end{aligned} \tag{9.7}$$

where $\Omega^1_{(i,j)(p,q)}$, $\Omega^2_{(i,j)(p,q)}$ and $\Omega^3_{(i,j)(p,q)}$ stand for the elementary functions. An extended description of the decomposition and definitions of the Ω functions can be found in [39].

The decomposition above is for the general form of the QAP (Eq. 9.4). If we focus on the standard version ($\psi_{ijpq} = d_{i,j}h_{p,q}$), we can observe the following differences:

- The diagonal values in $\mathbf{D} = [d_{i,j}]$ and $\mathbf{H} = [h_{k,l}]$ are 0 by default, and therefore the variable ψ_{iipp} is 0. Thus, the first summing term in Eq. 9.7 can be discarded.

[2] Note that cases of φ such as $i = j \wedge p \neq q$ are impossible ($\sigma(i) = \sigma(j) = p = q$), since under the interchange neighbourhood $i = j \iff p = q$, and also $i \neq j \iff p \neq q$.

- Due to the symmetry of the entries in the matrix $\mathbf{D} = [d_{i,j}]$, the first landscape of the decomposition, $\Omega^1_{(i,j)(p,q)}(\sigma)$, is constant.

Therefore, the general form of the QAP, g, is transformed to the classical formulation of the QAP, f, as:

$$f(\sigma) = \lambda + \sum_{\substack{i,j,p,q=1 \\ i \neq j \\ p \neq q}}^{n} \psi_{ijpq} \frac{\Omega^2_{(i,j)(p,q)}(\sigma)}{2(n-2)} + \sum_{\substack{i,j,p,q=1 \\ i \neq j \\ p \neq q}}^{n} \psi_{ijpq} \frac{\Omega^3_{(i,j)(p,q)}(\sigma)}{n(n-2)} \quad (9.8)$$

where λ is the constant value associated to the first elementary landscape.

The time complexity of calculating the decomposition of any solution in the search space is at most $O(n^3)$.

9.3 A non-dominated sorting genetic algorithm for the multi-objectivized QAP

In the previous section, we showed that the QAP function can be decomposed as the sum of two elementary landscapes and a constant value. Taking this decomposition as a basis, we reformulate the standard QAP as a problem with two objectives, where each term of the sum is a function to optimise (except λ which is constant).

In order to solve the multi-objectivized QAP, we propose a modified version of the Non-Dominated Sorting Algorithm II (NSGA-II) [50]. As stated in the introduction, NSGA-II is one of the most referred algorithms for multi-objective optimisation. Presented as an improvement of NSGA, this algorithm introduces two innovations in the selection of the solutions that survive at each generation. Firstly, at every generation t, NSGA-II combines the population of the parents and offsprings in a population P_t, and ranks the solutions according to a fast *non-dominance* sorting algorithm. This algorithm is an iterative scheme that, at each step, finds the set of non-dominated solutions in the population P_t, also known as *front*, and moves them into the next generation population P_{t+1}[3]. If the size of the front is smaller than the defined population size, all the members of the front are chosen. The remaining members of the new population will be chosen from the subsequent non-dominated fronts. This procedure is continued until no more fronts can be accommodated. In general, the last front accommodated will be partially chosen, since, presumably, it will be larger than the remaining space in P_{t+1}. In order to choose the solutions that will be selected from that last front, NSGA-II implements

[3] A solution x *dominates* a solution y when there is no function in the MOP for which x has a worse value than y, and there is at least one function for which x has a better value than y.

the *crowded*-comparison operator which, based on a density-estimation metric called *crowding*-distance, selects the solutions that are spread out in that front.

In a standard MOP, all the solutions within a front are considered equally *fit*. However, the proposed formulation of the QAP is not purely a MOP, since, ultimately, f measures the fitness of a solution. Therefore, we adapt the original design of NSGA-II to use on the multi-objectivized QAP. Particularly, we replace the diversity preservation algorithm applied to the last accommodated front with a simple sorting algorithm, which ranks from the best to the worst the solutions in relation with the original fitness function f.

As regards the other operators of the NSGA-II, we have set an adapted version of the one-point crossover (OPX) for permutations [103], and the one-interchange mutation scheme. Both operators have been selected without performing previous experiments.

9.4 Experiments

In order to demonstrate the validity of the multi-objectivization scheme proposed in this chapter, we compare the performance of the adapted NSGA-II for the multi-objectivized QAP with a simple genetic algorithm (SGA) for the standard QAP. The SGA is a simplified version of the adapted NSGA-II where the non-dominance sorting procedure has been replaced by a standard sorting algorithm that ranks the solutions in the population with respect to f (from the best to the worst).

We have selected all the instances up to size 100 from the Taillard's QAP benchmark [169] (which includes random instances, denoted as 'a', and real-life like instances, denoted as 'b', 48 instances in total), and instances of sizes 27, 45 and 75 from the $Taixxeyy$ benchmark of instances [54] (60 real-life like instances).

9.4.1 Parameter settings

We have selected the parameter settings detailed below without performing previous experiments:

- Population size: $8n$.
- Selected individuals: $8n$.
- Stopping criterion: maximum number of $1000n^2$ evaluations.

Both algorithms, NSGA-II and SGA, have been implemented in the C++ programming language. The experimentation was conducted on a cluster of 20 nodes, each of them equipped with two Intel Xeon X5650 CPUs and 48GB of memory.

9.4.2 Results

Each *algorithm - instance* pair was run 10 times. The performance measure employed in this study is the average relative percentage deviation (ARPD) [26]:

$$ARPD = \frac{|AvgRes - Best|}{Best}$$

where $AvgRes$ denotes the average result obtained throughout 10 repetitions of the algorithm, and $Best$ stands for the best known solution of the instances obtained from Eric Taillard's web page [169][4]. So, the lower the ARPD value, the better the performance. The ARPD results of the executions for the instances from the Taillard's and the $Taixxeyy$ benchmarks are summarised in Tables 9.1 and 9.2 respectively. Results in bold highlight the algorithm that obtained the best result.

Table 9.1: ARPD results of NSGA-II and SGA with respect to the best known solution for the Taillard's benchmark. In instances marked with (*) the best known solutions were not specified in the literature, and thus, we calculated it from the best fitness value obtained by NSGA-II or SGA at any execution.

Inst.	Best	NSGA-II	SGA	Inst.	Best	NSGA-II	SGA
10a	135028	**0.0063**	0.0099	29a	1669394	**0.0326**	0.0332
10b	1183760	**0.0091**	0.0146	30a	1818146	0.0328	**0.0324**
11a	188368	0.0319	**0.0152**	30b	637117113	**0.0287**	0.0375
12a	224416	**0.0339**	0.0356	31a	1945072	**0.0325**	0.0348
12b	39464925	0.0199	**0.0055**	32a	2033652	**0.0326**	0.0337
13a	270750	0.0283	**0.0247**	33a	2117042	**0.0362**	0.0409
14a	339524	**0.0142**	0.0238	34a	2254262	0.0318	**0.0310**
15a	388214	**0.0184**	0.0220	35a	2422002	0.0367	**0.0366**
15b	51765268	0.0022	**0.0021**	35b	283315445	**0.0170**	0.0382
16a	436316	0.0289	**0.0274**	36a*	2606790	**0.0111**	0.0115
17a	491812	**0.0282**	0.0289	37a*	2781888	**0.0112**	0.0120
18a	568250	**0.0289**	0.0331	38a*	2910360	**0.0087**	0.0148
19a	646208	**0.0362**	0.0385	39a*	3046888	0.0132	**0.0122**
20a	703482	**0.0333**	0.0384	40a	3139370	**0.0334**	0.0349
20b	122455319	**0.0053**	0.0354	40b	637250948	**0.0330**	0.0455
21a	781678	**0.0361**	0.0366	50a	4941410	**0.0344**	0.0367
22a	894546	0.0337	**0.0335**	50b	458821517	**0.0219**	0.0319
23a	1005738	**0.0346**	0.0358	60a	7208572	0.0374	**0.0336**
24a	1101310	**0.0338**	0.0352	60b	608215054	**0.0285**	0.0337
25a	1167256	**0.0367**	0.0381	64c	1855928	**0.0027**	0.0036
25b	344355646	**0.0135**	0.0206	80a	13557864	0.0289	**0.0287**
26a	1287596	**0.0302**	0.0306	80b	818415043	**0.0351**	0.0449
27a	1398568	**0.0350**	**0.0350**	100a	21125314	**0.0271**	0.0283
28a	1542998	0.0318	**0.0303**	100b	1185996137	**0.0308**	0.0448

[4] Best known solutions have been provided with reproducibility purposes.

Table 9.2: ARPD results of NSGA-II and SGA with respect to the best known solution of instances of size 27, 45 and 75 from the benchmark $Taixxeyy$.

	$n = 27$			$n = 45$			$n = 75$		
ID	*Best*	NSGA-II	SGA	*Best*	NSGA-II	SGA	*Best*	NSGA-II	SGA
1	2558	**0.207**	0.635	6412	**0.459**	0.623	14488	**0.470**	0.599
2	2850	**0.166**	0.666	5734	**0.685**	0.740	14444	**0.482**	0.601
3	3258	**0.559**	0.655	7438	**0.476**	0.705	14154	**0.476**	0.550
4	2822	**0.249**	0.597	6698	**0.624**	0.722	13694	**0.412**	0.729
5	3074	**0.197**	0.673	7274	**0.434**	0.755	12884	**0.468**	0.643
6	2814	**0.226**	0.640	6612	**0.370**	0.612	12534	**0.419**	0.583
7	3428	**0.167**	0.473	7526	**0.539**	0.623	13782	**0.544**	0.654
8	2430	**0.184**	0.688	6554	**0.276**	0.756	13948	**0.599**	0.702
9	2902	**0.198**	0.612	6648	**0.491**	0.796	12650	**0.588**	0.693
10	2994	**0.206**	0.682	8286	0.599	**0.547**	14192	**0.363**	0.720
11	2906	**0.217**	0.571	6510	**0.259**	0.773	15250	**0.408**	0.645
12	3070	**0.187**	0.405	7510	**0.616**	0.784	12760	**0.379**	0.677
13	2966	**0.128**	0.581	6120	**0.556**	0.621	13024	**0.515**	0.644
14	3568	**0.162**	0.647	6854	**0.607**	0.669	12604	**0.491**	0.638
15	2628	**0.550**	0.724	7394	**0.427**	0.739	14294	**0.260**	0.604
16	3124	**0.378**	0.488	6520	**0.664**	0.669	14204	**0.396**	0.638
17	3840	**0.172**	0.527	8806	**0.381**	0.556	13210	**0.546**	0.603
18	2758	**0.212**	0.654	6906	**0.437**	0.626	13500	**0.534**	0.606
19	2514	**0.178**	0.616	7170	**0.609**	0.723	12060	**0.623**	0.666
20	2638	**0.450**	0.733	6510	**0.530**	0.743	15260	**0.388**	0.643

Results show that NSGA-II is more competitive than SGA in 34 instances out of 48 in the Taillard's benchmark, and 59 out of 60 in the $Taixxeyy$ instances. In order to assess whether there exist statistical differences among the algorithms, we applied a non-parametric Wilcoxon test to the average results obtained by NSGA-II and SGA, for each benchmark of instances. A level of significance $\alpha = 0.05$ was set in both cases. The statistical test reported significant differences between the algorithms for both benchmarks. The p-value obtained for the Taillard's benchmark was 4×10^{-4}, and 2.04×10^{-11} for the $Taixxeyy$ benchmark. Therefore, the statistical analysis confirms that solving the QAP in its 2-objective form is preferred to the single-objective form in most of the evaluated cases.

Beyond these general conclusions, going deeper in the analysis of the results we see that NSGA-II is unquestionably better than SGA in real-life like instances ($Taixxeyy$ benchmark and 'b' type instances in the Taillard's benchmark): 69 out of 72 instances. Moreover, in these cases the ARPD difference between NSGA-II and SGA is remarkably larger than in the Taillard's benchmark (see Table 9.1).

9.5 Conclusions and future work

In this chapter, we presented a novel method to multi-objectivize single-objective problems based on the elementary landscape decomposition of the fitness function. In order to illustrate this procedure, we considered the QAP, whose decomposition into three elementary landscapes was introduced in [39]. Based on the analysis of the elementary functions, we transformed the QAP into a 2-objective problem. In order to demonstrate the validity of the proposed multi-objectivization scheme, we compared the performance of an adapted version of NSGA-II for the multi-objectivized QAP, with a simple genetic algorithm, SGA, for the standard QAP.

Experiments on two benchmarks of instances showed that NSGA-II is more competitive than SGA in 86% of the instances. In addition, experiments showed that NSGA-II is remarkably efficient on real-life like instances, this option being preferred in 95.8% of the instances. These results reveal that the multi-objectivization of single-objective problems by means of elementary landscape decomposition is an efficient scheme to be considered in other problems in which the decomposition of the landscape is already known.

As future work, we think that it is interesting to study the shapes of the Ω^2 and Ω^3 elementary functions in relation to the real-life like instances in order to understand the outstanding performance of the 2-objective approach.

In addition, beyond the QAP, we also find it interesting to extend the procedure presented in this paper to other problems such as the Linear Ordering Problem (LOP). The LOP under the interchange neighborhood is also a special case of Eq. 9.4, in which Ω^2 becomes a constant function. However, as reported in [28], the most efficient neighbourhood to solve the LOP is the *insert*, and therefore, finding an orthogonal basis of functions that permits to decompose the landscape under the insert neighbourhood is another challenging task that deserves further study.

10

Conclusions and future work

This chapter summaries the main contributions and conclusions provided in this dissertation. Moreover, this chapter addresses a set of general directions for further research. More specific conclusions and future work lines have been exposed in each corresponding chapter. At the end of the chapter we present the publications that result from the work in this dissertation.

10.1 Conclusions

In essence, this dissertation has been devoted to studying permutation-based combinatorial optimisation problems, and to proposing efficient procedures for solving them. To this end, we have dealt with permutation problems from three different research approaches: i) we proposed a new set of EDAs that implement probability models that explicitly define probability distributions in the domain of permutations, ii) we carried out an analysis of the LOP in relation to the fitness landscape produced under the insert neighborhood, and proposed a new neighbourhood system that allows to perform the search more efficiently, and iii) based on spectral landscape theory, and particularly on the elementary landscape decomposition, we studied the landscape decomposition associated to the QAP, and proposed an innovative scheme to reformulate it as a multi-objective problem.

As stated in previous chapters, it is dangerous to extrapolate the conclusions drawn on a problem to the whole class of problems. For that reason, in order to provide a solid framework for designing and testing the contributions proposed in this dissertation, we selected a representative benchmark of four permutation problems: TSP, LOP, PFSP and QAP.

In the following paragraphs, we elaborate on the main topics and conclusions of this thesis.

A large number of papers have demonstrated the validity of EDAs for solving optimisation problems. Nonetheless, according to the literature review in

Chapter 2, little work has been done with regard to permutation-based problems in the context of EDAs. With the exception of a few permutation-oriented algorithms, most of the approaches are adaptations of EDAs designed to solve integer or continuous domain problems. In both cases, the proposed EDAs do not learn explicit probability distributions over the space of permutations, and thus, in order to guarantee the generation of permutations, they are forced to include additional operators in the sampling step. Although these operators lead to permutations, the information kept by the probabilistic model is somehow denaturalised in the sampled solutions. The review of 11 EDAs for permutation problems revealed that two permutation-oriented EDAs, EHBSA and NHBSA, are clearly the most competitive algorithms among those compared. However, an exhaustive analysis of the models implemented by these two approaches, suggests that these models only capture a minimal part of the information, univariate and bivariate, about the permutation nature of the solutions.

Motivated by the absence of EDAs that implement efficient schemes to model probability distributions over the space of permutations, we proposed using distance-based exponential probability models such as the Generalised Mallows model in the context of EDAs. Defined by a distance on permutations, a central permutation and a set of spread parameters, Generalized Mallows explicitly assigns to every permutation in the search space a probability that depends on its distance to the central permutation. In order to demonstrate the validity of this model in the context of EDAs, in Chapter 4, we presented the Generalized Mallows EDA (GMEDA) under the Kendall's-τ distance, and the Hybrid Generalized Mallows EDA (HGMEDA). Taking the PFSP as a case study, we conducted experiments demonstrating that HGMEDA is the new state-of-the-art algorithm for this problem. Moreover, the experimental study supported the original intuition which indicated that probability models specifically designed for permutation spaces could take a qualitative leap with respect to the performance of EDAs.

Despite the success of the Generalized Mallows model under the Kendall's-τ distance on the PFSP, it is obvious that the ability of Generalized Mallows to efficiently capture the relevant information from a sample of solutions of a given problem is clearly characterised, as demonstrated in Chapter 5, by the distance used. Therefore, in order to obtain competitive EDAs, it is essential to choose a distance that shows certain correlation between the locality notion induced by the distance in the search space of solutions, and the characteristics of the fitness value associated to this. In this sense, we extended our study to investigate the performance of GMEDA on two other distances for permutations: Cayley and Ulam. A sound analysis of the models on four benchmark problems, showed that the performance of the EDAs is clearly influenced by the distance for permutations used inside the model. Furthermore, the experimental study showed that, for most of the problems, there is a distance that outperforms the rest almost systematically. In order to better understand the behaviour of the distances, and have a priori knowledge that

allows us to predict the most suitable one for any given problem, we moved our analysis into the framework of local search algorithms (LSs). Based on the explicit relation between the Kendall's-τ, Cayley and Ulam distances, and the *swap*, *interchange* and *insert* neighbourhood systems, we investigated whether such correspondence could be extrapolated to the performance of EDAs and LSs. According to our study, there exist strong performance correlations when the distance and neighbourhood used in each framework hold that correspondence.

In addition to distance-based probability models, in Chapter 6, we investigated a generalisation of the well-known Bradley Terry model in the context of EDAs: the Plackett-Luce model. Defined by a vector of scores, this model decomposes the process of generating a permutation of n items into n sequential stages: at each stage i, an item is assigned to the position i according to the probability based on the scores of the unassigned elements. Conducted experiments revealed that the Plackett-Luce EDA clearly outperforms the results of the Generalized Mallows EDA under the Kendall's-τ distance, and also that of EHBSA and NHBSA, when optimising the LOP. An analysis of the results point out that the Plackett-Luce model is able to capture the characteristics of the fitness function associated to the LOP more efficiently than the rest of the models included in the experimental study.

Leaving EDAs aside, in the second research line we approached permutation problems from the perspective of fitness landscape analysis. Particularly, in Chapter 7 we introduced a detailed theoretical study of the LOP in relation to the landscape produced under the insert neighbourhood. Based on this study, we presented a method that allows to extract static information about the problem and incorporate it to improve the performance of local search algorithms. Moreover, we developed a method to detect the neighbourhood operations that cannot generate local optima solutions. As a result, an improved version of the insert neighbourhood system, called *restricted insert neighbourhood* was proposed. Conducted experiments showed that when the proposed neighbourhood is applied to two state-of-the-art algorithms, it outperforms the classical designs systematically when a maximum number of evaluations are permitted, and in most cases, when a maximum execution time is proposed.

In Chapter 8, we presented an analysis of the instance complexity of the LOP, focusing especially on identifying the characteristics of the instances that influence their complexity. Characterizing the conditions required by each item to generate a local optima for the insert neighbourhood, we presented two new metrics, *insert ratio* and *ubiquity ratio*. Experimental results showed that there exists a relation between the proposed metrics and the number of local optima in the fitness landscape.

Going deeper into the landscape theory, the third research line of the dissertation (Chapter 9) has been devoted to investigating a more complex aspect of fitness landscapes: the elementary landscape decomposition. Particularly, we presented a novel method to multi-objectivise single-objective problems

based on the elementary landscape decomposition of the fitness function. In order to illustrate this procedure, we considered the QAP and the decomposition of the landscape produced under the interchange neighbourhood. Based on the analysis of the elementary functions, we proposed to multi-objectivize the QAP by decomposing its fitness function into two subfunctions. The experiments performed with a NSGA-II (for the multi-objective approach) and a GA (for the classical approach) revealed that the multi-objectivization of single-objective problems by means of elementary landscape decomposition arises as an efficient optimization strategy.

10.2 Future work

The contributions of this dissertation have led multiple open paths for future research. In the following paragraphs, we present a relation of topics.

Despite the successful performance shown by the probability models designed specifically for permutation domains, we believe that there are still many aspects that deserve further research with respect to EDAs for permutation domains. In what follows, we present four theoretical/methodological research lines of EDAs for future work:

- As described in Chapter 4, the Mallows and Generalized Mallows models are unimodal whenever the spread parameters are different from 0. However, the configuration space of solutions is usually multimodal. In this sense, similarly to [140], mixtures of models [129] or kernels of Generalized Mallows models [99] could be investigated in order to take a step forward in terms of performance for EDAs.
- The review of distances for the Mallows and Generalized Mallows models presented in Chapter 5, demonstrated that the performance of EDAs with these models is strongly correlated with that of MLS algorithms when the correspondence between the distance (in the model) and the neighbourhood system (in the local search) is held. In fact, preliminary experiments already suggested that these algorithms frequently converge to local optima of the distance used inside the probability model. In this sense, studying the dynamics of the Mallows and Generalized Mallows EDAs with regard to their convergence is a topic that deserves further research.
- Throughout the first part of the dissertation, we conducted experiments and performed statistical tests in order to demonstrate the validity of a specific model-distance for a given problem. Nevertheless, a more theoretical approach consists of calculating the probability distribution over the whole search space of solutions induced by the probability model at hand, i.e. either Mallows, Generalized Mallows or Plackett-Luce, and comparing it to the Boltzmann distribution associated to the problem. Assuming that the Boltzmann distribution is the ideal, although infeasible, distribution

to optimise a given problem, choosing the model with the lowest Kullback-Leibler divergence with respect to this distribution could be an interesting criterion to select the most suitable probability model.
- In this dissertation, distance-based exponential models have been studied for the Kendall's-τ, Cayley and Ulam distances, nonetheless, other distances on permutations such as Hamming or Spearman could be considered in future research.

From an application perspective of EDAs, the experiments in Chapter 5 showed that the Mallows model under the Ulam distance, or the Generalized Mallows model under the Cayley distance, behave competitively for the LOP and QAP respectively. Alternatively, we drew similar conclusions in Chapter 6 with the Plackett-Luce model and the LOP. In view of the successful performance obtained by the GMEDA on the PFSP (see Chapter 4), we suggest that similar strategies could be developed in the cases mentioned above in order to outperform state-of-the-art algorithms in each problem.

In relation to the study of the LOP carried out in Chapters 7 and 8, we suggest two research lines for the future:

- The proposed contributions considered univariate information associated to the items in order to i) define the restricted insert neighbourhood in Chapter 7, and ii) propose new complexity metrics in Chapter 8. However, we believe that to consider bivariate or even multivariate information about the items could provide more insights on the contributions of both chapters.
- An alternative application of the restrictions matrix described in Chapter 7 could be that of branch and bound algorithms. Taking into account that this matrix describes the positions at which local optima are not generated, and hence global optimum, this information could be used as a bounding rule to discard those branches that locate items in restricted positions.

The multi-objectivisation by means of elementary landscape decomposition presented in Chapter 9 leaves also many trends that are worth extending in future works.

- In the particular case of the QAP, we saw that the landscape produced under the interchange neighbourhood could be formulated as the superposition of three elementary landscapes. However, the shapes of this landscapes were not studied in relation to the parameters of the problem, which could provide new insights into the difficulty of solving a given instance.
- Taking into account that the LOP and TSP are particular cases of the QAP, the elementary landscape decomposition under the interchange neighbourhood is also extendable to these problems. Therefore, it could be interesting to study how the shapes of the landscapes vary with respect to the QAP.
- Despite the suitability of the interchange neighbourhood on the QAP, it is not the case for the LOP, where the insert neighbourhood is the one that

performs the best. Extending the same methodology described in Chapter 9, and based on the work of Chicano et al. [40], an interesting research line would be to find the orthogonal basis of functions in which to decompose the landscape produced by the LOP under the insert neighbourhood.

In order to conclude this section, we propose a more general research line for future work. We believe that it is worth investigating towards a theory on the parameterised complexity of permutation problems. Our aim in the future is to be able to, based on the parameters of a given instance, predict the difficulty of solving that instance with a specific algorithm. In this sense, we think that the contribution of items in the permutation to the fitness function, or the number of parameters of the problem affected by an operation on a specific item have outstanding relevance and should be investigated in detail.

10.3 Publications

The research work carried out during this thesis has produced the following publications and submissions:

10.3.1 Referred journals

- **J. Ceberio**, E. Irurozki, A. Mendiburu, J.A. Lozano (2012) A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1(1):103-117.
- **J. Ceberio**, E. Irurozki, A. Mendiburu, J.A. Lozano (2014) A Distance-based ranking model estimation of distribution algorithm for the flow-shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286-300.
- **J. Ceberio**, A. Mendiburu, J.A. Lozano (2014) The linear ordering problem revisited. *European Journal of Operational Research.* In press.
- **J. Ceberio**, E. Irurozki, A. Mendiburu, J.A. Lozano (2014) A review of distances for the Mallows and Generalized Mallows estimation of distribution algorithms. *Journal of Computational Optimization and Applications.* Submitted.
- **J. Ceberio** , A. Mendiburu, J.A. Lozano (2014) Multi-objectivizing the quadratic assignment problem by means of a elementary landscape decomposition. *Natural Computing.* Submitted.

10.3.2 Conference communications

- **J. Ceberio**, A. Mendiburu, J.A. Lozano (2011) A preliminary study on EDAs for permutation problems based on marginal-based models. *In Proceedings of 2011 Genetic and Evolutionary Computation Conference (GECCO-2011), Dublin, Ireland, 12-16, July 2011. Pp. 609-616.*

- **J. Ceberio**, A. Mendiburu, J.A. Lozano (2011) Introducing The Mallows model on estimation of distribution algorithms. *In Proceedings of 2011 International Conference on Neural Information Processing (ICONIP-2011), Shanghai, China, 23-25, November 2011. Pp. 461-470.*
- **J. Ceberio**, A. Mendiburu, J.A. Lozano (2013) The Plackett-Luce ranking model on permutation-based optimization problems. *IEEE Congress on Evolutionary Computation (CEC-2013), Cancun, Mexico, 20-23 June 2013.*
- **J. Ceberio**, L. Hernando, A. Mendiburu, J.A. Lozano (2013) Understanding instance complexity in the linear ordering problem. *The International Conference on Intelligent Data Engineering and Automated Learning (IDEAL-2013), LNCS 8206, pp. 479-486, Hefei, Anhui, China, 20-23 October 2013*
- **J. Ceberio**, E. Irurozki, A. Mendiburu, J.A. Lozano (2014) Extending distance-based ranking models in estimation of distribution algorithms. *IEEE Congress on Evolutionary Computation (CEC-2014), Beijing, China, 6-11 July 2014.*

10.3.3 Collaborations

- E. Irurozki, **J. Ceberio**, B. Calvo, J.A. Lozano (2014). Sampling and learning the Mallows model under the Ulam distance. *Department of Computer Science and Artificial Intelligence. University of the Basque Country UPV/EHU. EHU-KZAA-TR;2014-04.*
- E. Irurozki, **J. Ceberio**, B. Calvo, J.A. Lozano (2014). Mallows model under the Ulam distance: a feasible combinatorial approach. *Neural Information Processing Systems (NIPS) - Analysis of Rank Data.* Accepted for publication.

References

[1] Abbass, H. A. and Deb, K. (2003). Searching under multi-evolutionary pressures. In *Proceedings of the 4th Conference on Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science*, pages 391–404. Springer Berlin Heidelberg.

[2] Agrawal, S., Wang, Z., and Ye, Y. (2008). Parimutuel Betting on Permutations. In Papadimitriou, C. and Zhang, S., editors, *Internet and Network Economics*, volume 5385 of *Lecture Notes in Computer Science*, pages 126–137. Springer Berlin Heidelberg.

[3] Aledo, J. A., Gámez, J. A., and Molina, D. (2013). Tackling the rank aggregation problem with evolutionary algorithms. *Applied Mathematics and Computation*, 222:632–644.

[4] Ali, A. and Meila, M. (2011). Experiments with Kemeny ranking: What works when? *Mathematical Social Sciences*.

[5] Allahverdi, A. and Aldowaisan, T. (2002). New heuristics to minimize total completion time in m-machine flowshops. *International Journal of Production Economics*, 77(1):71 – 83.

[6] Allahverdi, A., Ng, C., Cheng, T., and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985 – 1032.

[7] Angel, E. and Zissimopoulos, V. (2000). On the classification of np-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99(1–3):261 – 277.

[8] Aujac, H. (1960). La hiérarchie des industries dans un tableau des échanges interindustriels. *Revue économique*, 11(2):169–238.

[9] Baker, K. (1974). *Introduction to sequencing and scheduling*. Wiley.

[10] Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon Report, CMU-CS-94-163.

[11] Bartholdi, J., Tovey, C. A., and Trick, M. A. (1989). Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165.

[12] Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *INFORMS Journal on Computing*, 6(2):154–160.

[13] Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., and Abramson, D. (2000). Scheduling aircraft landings—the static case. *Transportation Science*, 34(2):180–197.

[14] Becker, O. (1967). Das helmstädtersche reihenfolgeproblem – die effizienz verschiedener näherungsverfahren -. In *Computers Uses in the Social Sciences, Berichteiner Working Conference*, Vienna.

[15] Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2002). Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867–2880.

[16] Betzler, N., Guo, J., Komusiewicz, C., and Niedermeier, R. (2011). Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4):774–789.

[17] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

[18] Borda, J. C. (1781). Memoire sur les elections au scrutin. *Histoire de l'Academie Royale des Science*.

[19] Borwein, J. M. and Borwein, P. B. (1987). *Pi and the AGM: A Study in the Analytic Number Theory and Computational Complexity*. Wiley-Interscience, New York, NY, USA.

[20] Bosman, P. A. N. and Thierens, D. (2000). Expanding from Discrete to Continuous Estimation of Distribution Algorithms: The IDEA. In *In Parallel Problem Solving From Nature - PPSN VI*, pages 767–776. Springer.

[21] Bosman, P. A. N. and Thierens, D. (2001). Crossing the road to efficient IDEAs for permutation problems. In et al., L. S., editor, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, GECCO 2001, pages 219–226, San Francisco, California. Morgan Kaufmann.

[22] Box, G. E. P. and Jenkins, G. (1970). *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated.

[23] Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4).

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

[25] Campos, V., Glover, F., Laguna, M., and Martí, R. (2001). An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization*, 21(4):397–414.

[26] Ceberio, J., Irurozki, E., Mendiburu, A., and Lozano, J. A. (2014a). A Distance-based Ranking Model Estimation of Distribution Algorithm for the Flowshop Scheduling Problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286 – 300.

[27] Ceberio, J., Irurozki, E., Mendiburu, A., and Lozano, J. A. (2014b). Extending distance-based ranking models in estimation of distribution algorithms. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*. IEEE.

[28] Ceberio, J., Irurozki, E., Mendiburu, A., and Lozano, J. A. (2014c). A review of distances for the mallows and generalized mallows estimation of distribution algorithms. *Computational Optimization and Applications*.

[29] Ceberio, J., Mendiburu, A., and Lozano, J. A. (2011). Introducing the Mallows Model on Estimation of Distribution Algorithms. In Lu, B.-L., Zhang, L., and Kwok, J. T., editors, *Proceedings of International Conference on Neural Information Processing (ICONIP)*, Lecture Notes in Computer Science, pages 461–470. Springer.

[30] Ceberio, J., Mendiburu, A., and Lozano, J. A. (2013). The Plackett-Luce Ranking Model on Permutation-based Optimization Problems. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, pages 494 – 501.

[31] Ceberio, J., Mendiburu, A., and Lozano, J. A. (2014d). The Linear Ordering Problem Revisited. *European Journal of Operational Research*.

[32] Chanas, S. and Kobylanski, P. (1996). A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6(2):191–205.

[33] Charon, I. and Hudry, O. (1998). Lamarckian genetic algorithms applied to the aggregation of preferences. *Annals of Operations Research*, 80(0):281–297.

[34] Charon, I. and Hudry, O. (2006). A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15):2097 – 2116.

[35] Charon, I. and Hudry, O. (2007). A survey on the linear ordering problem for weighted or unweighted tournaments. *4or*, 5(1):5–60.

[36] Chen, S. and Chen, M. (2011). Bi-Variate Artificial Chromosomes with Genetic Algorithm for Single Machine Scheduling Problems with Sequence-Dependent Setup Times. In *Proceedings of the Congress on Evolutionary Computation*.

[37] Chenery, H. B. and Watanabe, T. (1958). International comparisons of the structure of production. *Econometrica*, 26(4):487–521.

[38] Cheng, W. and Hullermeier, E. (2009). A Simple Instance-Based Approach to Multilabel Classification Using the Mallows Model. In *Workshop Proceedings of Learning from Multi-Label Data*, pages 28–38, Bled, Slovenia.

[39] Chicano, F., Luque, G., and Alba, E. (2010). Elementary landscape decomposition of the quadratic assignment problem. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 1425–1432.

[40] Chicano, F., Whitley, L. D., and Alba, E. (2011a). A methodology to find the elementary landscape decomposition of combinatorial optimization problems. *Evolutionary Computation*, 19(4):597–637.

[41] Chicano, F., Whitley, L. D., Alba, E., and Luna, F. (2011b). Elementary landscape decomposition of the frequency assignment problem. *Theoretical Computer Science*, 412(43):6002 – 6019.

[42] Chira, C., Pintea, C. M., Crisan, G. C., and Dumitrescu, D. (2009). Solving the linear ordering problem using ant models. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1803–1804, New York, NY, USA. ACM.

[43] Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

[44] Chung, C.-S., Flynn, J., and Kirca, O. (2002). A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems. *International Journal of Production Economics*, 79(3):185–196.

[45] Codenotti, B. and Margara, L. (1992). *Local Properties of Some NP-complete Problems*. TR: International Computer Science Institute. International Computer Science Institute.

[46] Cohen, W. W., Schapire, R. E., and Singer, Y. (1998). Learning to order things. In *Proceedings of the 1997 conference on Advances in neural information processing systems 10*, NIPS '97, pages 451–457, Cambridge, MA, USA. MIT Press.

[47] Costa, W. E., Goldbarg, M. C., and Goldbarg, E. G. (2012). New vns heuristic for total flowtime flowshop scheduling problem. *Expert Systems with Applications*, 39(9):8149 – 8161.

[48] Critchlow, D. E., Fligner, M. A., and Verducci, J. S. (1991). Probability models on rankings. *Journal of Mathematical Psychology*, 35(3):294 – 318.

[49] De Bonet, J. S., Isbell, C. L., and Viola, P. (1997). MIMIC: Finding Optima by Estimating Probability Densities. In M. Mozer, M. J. and eds., T. P., editors, *Proceedings of the 1997 conference on Advances in Neural Information Processing Systems (NIPS)*, volume 9.

[50] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[51] Deza, M., Supérieure, L.-e. N., and Huang, T. (1998). Metrics on permutations, a survey. In *Journal of Combinatorics, Information and System Sciences*.

[52] Diaconis, P. (1988). *Group representations in probability and statistics*. Institute of Mathematical Statistics Lecture Notes—Monograph Series, 11. Institute of Mathematical Statistics, Hayward, CA.

[53] Dong, X., Huang, H., and Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research*, 36(5):1664 – 1669. Selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X).

[54] Drezner, Z., Hahn, P., and Taillard, É. (2005). Recent advances for the quadratic assignment problem with special emphasis on instances that

are difficult for meta-heuristic methods. *Annals of Operations Research*, 139(1):65–94.

[55] Fligner, M. and Verducci, J. (1988). Multistage ranking models. *Journal of the American Statistical Association*, 83(403):892–901.

[56] Fligner, M. A. and Verducci, J. S. (1986). Distance based ranking Models. *Journal of the Royal Statistical Society*, 48(3):359–369.

[57] Framinan, J. M. and Leisten, R. (2003). An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *Omega*, 31(4):311 – 317.

[58] Framinan, J. M., Leisten, R., and Ruiz-Usano, R. (2005). Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research*, 32:1237–1254.

[59] Gajpal, Y. and Rajendran, C. (2005). An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. *International Journal of Production Economics*, 101(2):259–272.

[60] Garcia, C. G., Pérez-Brito, D., Campos, V., and Martí, R. (2006). Variable neighborhood search for the linear ordering problem. *Computers & Operations Research*, 33(12):3549 – 3565.

[61] Garcia, S. and Herrera, F. (2008). An Extension on "Statistical Comparisons of Classifiers over Multiple Data Set" for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

[62] Garcia, S., Molina, D., Lozano, M., and Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15(6):617–644.

[63] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[64] Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flow shop and jobshop scheduling. *Mathematics of Operations Research*, 2:117–129.

[65] Glover, F., Klastorin, T., and Klingman, D. (1972). *Optimal weighted ancestry relationships*. Management science report series. Business Research Division, Graduate School of Business Administration, University of Colorado.

[66] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison/Wesley, Reading MA.

[67] Goldberg, D. E. and Jr., R. L. (1985). Alleles Loci and the Traveling Salesman Problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ.

[68] Grötschel, M., Jünger, M., and Reinelt, G. (1984). A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220.

[69] Grover, L. K. (1992). Local search and the local structure of np-complete problems. *Operations Research Letters*, 12(4):235 – 243.

[70] Guiver, J. and Snelson, E. (2009). Bayesian inference for Plackett-Luce ranking models. In *International Conference on Machine Learning (ICML 2009)*, ICML '09, pages 377–384. ACM.

[71] Gupta, J. and Stafford, J. E. (2006). Flow shop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699–711.

[72] Handl, J., Lovell, S., and Knowles, J. (2008). Multiobjectivization by decomposition of scalar cost functions. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *Parallel Problem Solving from Nature – PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 31–40. Springer Berlin Heidelberg.

[73] Hejazi, S. R. and Saghafianz, S. (2004). Flowshop-scheduling problems with makespan criterion: a review.

[74] Henrion, M. (1986). Propagating uncertainty in Bayesian networks by Probabilistic Logic Sampling. In *Uncertainty in Artificial Intelligence (UAI)*, pages 149–164.

[75] Hernando, L., Mendiburu, A., and Lozano, J. A. (2013). An evaluation of methods for estimating the number of local optima in combinatorial optimization problems. *Evolutionary Computation*, 21(4):625–658.

[76] Huang, J., Guestrin, C., and Guibas, L. (2009). Fourier theoretic probabilistic inference over permutations. *Journal of Machine Learning Research (JMLR)*, 10:997–1070.

[77] Hunter, D. R. (2004). MM Algorithms for Generalized Bradley-Terry Models. *The Annals of Statistics*, 32(1):384–406.

[78] Inza, I., Larrañaga, P., and Sierra, B. (2002). Feature Subset Selection by Estimation of Distribution Algorithms. In Larrañaga, P. and Lozano, J. A., editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 269–294. Kluwer Academic Publishers.

[79] Irurozki, E., Calvo, B., and Lozano, J. A. (2014a). Sampling and learning mallows and generalized mallows models under the cayley distance. Technical report, University of the Basque Country UPV/EHU.

[80] Irurozki, E., Calvo, B., and Lozano, J. A. (2014b). Sampling and learning the mallows model under the ulam distance. Technical report, University of the Basque Country UPV/EHU.

[81] Jarboui, B., Eddaly, M., and Siarry, P. (2009). An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Computers & Operations Research*, 36(9):2638–2646.

[82] Jarboui, B., Ibrahim, S., Siarry, P., and Rebai, A. (2008). A combinatorial particle swarm optimisation for solving permutation flowshop problems. *Computers & Industrial Engineering*, 54(3):526 – 538.

[83] Jiang, S., Ziver, A., Carter, J., Pain, C., Goddard, A., Franklin, S., and Phillips, H. (2006). Estimation of Distribution Algorithms for nuclear reactor fuel management optimisation. *Annals of Nuclear Energy*, 33(11-12):1039–1057.

[84] Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

[85] Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In Eshelman, L. J., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Mateo, CA. Morgan Kaufmann Publishers.

[86] Kaas, R. (1981). A branch and bound algorithm for the acyclic subgraph problem. *European Journal of Operational Research*, 8(4):355 – 362.

[87] Kalczynski, P. J. and Kamburowski, J. (2005). On the NEH heuristic for minimizing the makespan in permutation flow shops. *Omega*.

[88] Kemeny, J. G. (1959). Mathematics without numbers. *Daedalus*, 88:577–591.

[89] Kernighan, B. W. and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307.

[90] Knjazew, D. and Goldberg, D. E. (2000). Omega - ordering messy ga: Solving permutation problems with the fast genetic algorithm and random keys. In *GECCO*, pages 181–188.

[91] Knowles, J., Watson, R., and Corne, D. (2001). Reducing local optima in single-objective problems by multi-objectivization. In Zitzler, E., Thiele, L., Deb, K., Coello Coello, C., and Corne, D., editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 269–283. Springer Berlin Heidelberg.

[92] Koopmans, T. C. and Beckmann, M. J. (1955). Assignment Problems and the Location of Economic Activities. Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University.

[93] Laguna, M., Marti, R., and Campos, V. (1999). Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & Operations Research*, 26:1217–1230.

[94] Larrañaga, P., Etxeberria, R., Lozano, J. A., and Peña, J. M. (2000a). Combinatorial optimization by learning and simulation of Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI 2000*, pages 343–352, Stanford, CA, USA.

[95] Larrañaga, P., Etxeberria, R., Lozano, J. A., and Peña, J. M. (2000b). Optimization in continuous domains by learning and simulation of Gaussian networks. In *Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the 2000 Genetic and Evolutionary Computation Conference, GECCO 2000*, pages 201–204, Las Vegas, Nevada, USA.

[96] Larrañaga, P. and Lozano, J. A. (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

[97] Lau, K. K., Yuen, P. C., and Tang, Y. Y. (2005). Directed connection measurement for evaluating reconstructed stroke sequence in handwriting images. *Pattern Recognition*, 38(3):323–339.

[98] Lebanon, G. and Lafferty, J. (2002). Cranking: Combining rankings using conditional probability models on permutations. In *International Conference on Machine Learning (ICML 2002)*, pages 363–370.

[99] Lebanon, G. and Mao, Y. (2008). Non-Parametric Modeling of Partially Ranked Data. *Journal of Machine Learning Research (JMLR)*, 9:2401–2429.

[100] Leontief, W. (2008). *Input-Output Economics*. Cambridge University Press.

[101] Li, X., Wang, Q., and Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega*, 37(1):155 – 164.

[102] Liao, C.-J., Tseng, C.-T., and Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research*, 34:3099–3111.

[103] Lim, M. H., Yuan, Y., and Omatu, S. (2000). Efficient genetic algorithms using simple genes exchange localsearch policy for the quadratic assignment problem. *Computational Optimization and Applications*, 15(3):249–268.

[104] Liu, H., Gao, L., and Pan, Q. (2011). A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. *Expert Systems with Applications*, 38:4348–4360.

[105] Liu, J. and Reeves, C. R. (2001). Constructive and composite heuristic solutions to the p//[summation operator]ci scheduling problem. *European Journal of Operational Research*, 132(2):439 – 452.

[106] Lozano, J. A., Larrañaga, P., Inza, I., and Bengoetxea, E. (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.

[107] Lozano, J. A. and Mendiburu, A. (2002). Estimation of Distribution Algorithms applied to the job schedulling problem. In Larrañaga, P. and Lozano, J. A., editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

[108] Lu, T. and Boutilier, C. (2011). Learning mallows models with pairwise preferences. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 145–152, New York, NY, USA. ACM.

[109] Luce R., D. (1959). *Individual Choice Behavior*. Wiley, New York.

[110] Mallows, C. L. (1957). Non-null ranking models. *Biometrika*, 44(1-2):114–130.

[111] Manderick, B., de Weger, M., and Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. In *Proceedings of the fourth international conference on genetic algorithms*, pages 143–150.

[112] Mandhani, B. and Meila, M. (2009a). Tractable search for learning exponential models of rankings. In *Artificial Intelligence and Statistics (AISTATS)*.

[113] Mandhani, B. and Meila, M. (2009b). Tractable Search for Learning Exponential Models of Rankings. *Journal of Machine Learning Research (JMLR)*, 5:392–399.

[114] Mao, Y. and Lebanon, G. (2008). Non-Parametric Modeling of Partially Ranked Data. *Journal of Machine Learning Research (JMLR)*, 9:2401–2429.

[115] Marden, J. I. (1995). *Analyzing and Modeling Rank Data*. Chapman & Hall.

[116] Martí, R. and Reinelt, G. (2011). *The linear ordering problem: exact and heuristic methods in combinatorial optimization*, volume 175. Springer.

[117] Martí, R., Reinelt, G., and Duarte, A. (2012). A benchmark library and a comparison of heuristic methods for the linear ordering problem. *Computational Optimization and Applications*, 51(3):1297–1317.

[118] Maydeu-Olivares, A. (1999). Thurstonian modeling of ranking data via mean and covariance structure analysis. *Psychometrika*, 64(3):325–340.

[119] Maydeu-Olivares, A. (2001). Limited information estimation and testing of thurstonian models for paired comparison data under multiple judgment sampling. *Psychometrika*, 66(2):209–227.

[120] Maydeu-Olivares, A. (2002). Limited information estimation and testing of thurstonian models for preference data. *Mathematical Social Sciences*, 43(3):467 – 483. Random Utility Theory and Probabilistic measurement theory.

[121] McCall, J., Brownlee, A. E. I., and Shakya, S. (2012). Applications of distribution estimation using markov network modelling (DEUM). In Shakya, S. and Santana, R., editors, *Markov Networks in Evolutionary Computation*, pages 193–207. Springer.

[122] Meila, M., Phadnis, K., Patterson, A., and Bilmes, J. (2007). Consensus ranking under the exponential model. In *22nd Conference on Uncertainty in Artificial Intelligence (UAI07)*, Vancouver, British Columbia.

[123] Meilă, M. and Bao, L. (2010). An exponential model for infinite rankings. *Journal of Machine Learning Research*, 11:3481–3518.

[124] Mendiburu, A., Lozano, J. A., and Miguel-Alonso, J. (2005). Parallel Implementation of EDAs Based on Probabilistic Graphical Models. *IEEE Transactions on Evolutionary Computation*, 9(4):406–423.

[125] Mendiburu, A., Miguel-Alonso, J., Lozano, J. A., Ostra, M., and Ubide, C. (2006). Parallel EDAs to create multivariate calibration models for quantitative chemical applications. *Journal of Parallel and Distributed Computing*, 66(8):1002–1013.

[126] Mitchell, J. and Borchers, B. (1996). Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operations Research*, 62(1):253–276.

[127] Mitchell, J. and Borchers, B. (2000). Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In Frenk, H., Roos, K., Terlaky, T., and Zhang, S., editors, *High Performance Optimization*, volume 33 of *Applied Optimization*, pages 349–366. Springer US.

[128] Mühlenbein, H. and Paaß, G. (1996). From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV*, pages 178–187.

[129] Murphy, T. B. and Martin, D. (2003). Mixtures of distance-based models for ranking data. *Computational Statistics & Data Analysis*, 41(3-4):645–655.

[130] Nagano, M. S., Ruiz, R., and Lorena, L. A. N. (2008). A constructive genetic algorithm for permutation flowshop scheduling. *Computers & Industrial Engineering*, 55(1):195–207.

[131] Nawaz, M., Enscore Jr, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

[132] Neumann, F. and Wegener, I. (2006). Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319.

[133] Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175.

[134] Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557.

[135] Parsons, R., Forrest, S., and Burks, C. (1995). Genetic algorithms, operators, and dna fragment assembly. *Machine Learning*, 21(1-2):11–33.

[136] Pelikan, M. and Goldberg, D. E. (2000). Hierarchical problem solving and the Bayesian optimization algorithm. In Whitley, D., Goldberg, D., Cantú-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 267–274, San Francisco, CA. Morgan Kaufmann Publishers.

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

[138] Pelikan, M., Sastry, K., and Cantú-Paz, E. (2006). *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications (Studies in Computational Intelligence)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[139] Pelikan, M., Tsutsui, S., and Kalapala, R. (2007). Dependency Trees, Permutations, and Quadratic Assignment Problem. Technical report, Medal Report No. 2007003.

[140] Peña, J. M., Lozano, J. A., and Larrañaga, P. (2005). Globally multimodal problem optimization via an estimation of distribution algorithm based on unsupervised learning of bayesian networks. *Evolutionary Computation*, pages 43–66.

[141] Pintea, C.-M., Crisan, G. C., Chira, C., and Dumitrescu, D. (2009). A hybrid ant-based approach to the economic triangulation problem for input-output tables. In *Proceedings of the 4th International Conference*

on Hybrid Artificial Intelligence Systems, HAIS '09, pages 376–383, Berlin, Heidelberg. Springer-Verlag.

[142] Plackett, R. L. (1975). The Analysis of Permutations. *Journal of the Royal Statistical Society*, 24(10):193–202.

[143] Pop, P. and Matei, O. (2012). A genetic programming approach for solving the linear ordering problem. In Corchado, E., Snášel, V., Abraham, A., Woźniak, M., Graña, M., and Cho, S.-B., editors, *Hybrid Artificial Intelligent Systems*, volume 7209 of *Lecture Notes in Computer Science*, pages 331–338. Springer Berlin Heidelberg.

[144] Rajendran, C. (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1):65–73.

[145] Rajendran, C. and Ziegler, H. (1997). An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103(1):129–138.

[146] Rajendran, C. and Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426–438.

[147] Reeves, C. R. (1993). Improving the efficiency of tabu search for machine sequencing problems. *Journal of Operations Research Society*, 44(4):375–382.

[148] Reinelt, G. (1990). TSPLIB - A t.s.p. library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg.

[149] Robles, V., de Miguel, P., and Larrañaga, P. (2002). Solving the Traveling Salesman Problem with EDAs. In Larrañaga, P. and Lozano, J. A., editors, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

[150] Romero, T. and Larrañaga, P. (2009). Triangulation of Bayesian networks with recursive Estimation of Distribution Algorithms. *Internation Journal of Approximate Reasoning*, 50(3):472–484.

[151] Sagarna, R. and Lozano, J. A. (2006). Scatter Search in software testing, comparison and collaboration with Estimation of Distribution Algorithms. *European Journal of Operational Research*, 169(2):392–412.

[152] Santana, R., Bielza, C., Larranaga, P., Lozano, J. A., Echegoyen, C., Mendiburu, A., Armananzas, R., and Shakya, S. (2010). Mateda-2.0: Estimation of distribution algorithms in matlab. *Journal of Statistical Software*, 35(7):1–30.

[153] Santana, R., Larrañaga, P., and Lozano, J. A. (2008). Protein folding in simplified models with Estimation of Distribution Algorithms. *IEEE Transactions On Evolutionary Computation*, 12(4):418–438.

[154] Scharnow, J., Tinnefeld, K., and Wegener, I. (2005). The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366.

[155] Schiavinotto, T. and Stützle, T. (2004). The linear ordering problem: instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*.

[156] Schiavinotto, T. and Stützle, T. (2007). A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143 – 3153.

[157] Segura, C., Coello, C. A. C., Miranda, G., and Leon, C. (2013). Using multi-objective evolutionary algorithms for single-objective optimization. *4OR*, 11(3):201–228.

[158] Soto, M. R., Ochoa, A., González-Fernández, Y., Milanés, Y., Álvarez, A., Carrera, D., and Moreno, E. (2012). Vine estimation of distribution algorithms with application to molecular docking. In Shakya, S. and Santana, R., editors, *Markov Networks in Evolutionary Computation*, pages 175–190. Springer.

[159] Srinivas, N. and Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2:221–248.

[160] Staden, R. (1979). A strategy of dna sequencing employing computer programs. *Nucleic acids research*, 6(7):2601–2610.

[161] Stadler, P. F. (1995). Towards a theory of landscapes. In López-Peña, R., Waelbroeck, H., Capovilla, R., García-Pelayo, R., and Zertuche, F., editors, *Complex Systems and Binary Networks*, volume 461-461 of *Lecture Notes in Physics*, pages 78–163. Springer Berlin Heidelberg.

[162] Stadler, P. F. (1996). Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45.

[163] Stadler, P. F. (2002). Fitness landscapes. *Applied Mathematics and Computation*, 117:187–207.

[164] Stanley, R. P. (1986). *Enumerative Combinatorics*. Wadsworth Publ. Co., Belmont, CA, USA.

[165] Stützle, T. (1997). An ant approach to the flow shop problem. *In Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT 98)*, pages 1560–1564.

[166] Sutton, A. M., Whitley, L. D., and Howe, A. E. (2009). A polynomial time computation of the exact correlation structure of k-satisfiability landscapes. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 365–372, New York, NY, USA. ACM.

[167] Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

[168] Taillard, E. (1993a). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

[169] Taillard, E. (1993b). Problem instances.

[170] Tasgetiren, M. F., Liang, Y.-C., Sevkli, M., and Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime

minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3):1930 – 1947.

[171] Tayarani-N., M. and Prugel-Bennett, A. (2014). On the landscape of combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(3):420–434.

[172] Thurstone, L. L. (1994). A law of comparative judgment. *Psychological Review*, 34(4):273–286.

[173] Toth, P. and Vigo, D. (2001). *The vehicle routing problem*. Siam.

[174] Tromble, R. and Eisner, J. (2009). Learning linear ordering problems for better translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2*, EMNLP '09, pages 1007–1016.

[175] Tseng, L.-Y. and Lin, Y.-T. (2009). A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 198(1):84–92.

[176] Tsutsui, S. (2002). Probabilistic Model-Building Genetic Algorithms in Permutation Representation Domain Using Edge Histogram. In *Parallel Problem Solving from Nature—PPSN VII*, pages 224–233. Springer.

[177] Tsutsui, S. (2006). A Comparative Study of Sampling Methods in Node Histogram Models with Probabilistic Model-Building Genetic Algorithms. In *IEEE International Conference on Systems, Man, and Cybernetics. October 8-11, 2006, Taipei, Taiwan*, volume 4, pages 3132–3137.

[178] Tsutsui, S. (2009). Effect of Using Partial Solutions in Edge Histogram Sampling Algorithms with Different Local Searches. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, SMC 2009, pages 2137–2142. IEEE.

[179] Tsutsui, S. and Miki, M. (2002). Solving Flow Shop Scheduling Problems with Probabilistic Model-Building Genetic Algorithms using Edge Histograms. In *4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL 02)*, pages 776–780.

[180] Tsutsui, S., Pelikan, M., and Goldberg, D. E. (2003). Using Edge Histogram Models to Solve Permutation Problems with Probabilistic Model-Building Genetic Algorithms. Technical report, IlliGAL Report No. 2003022.

[181] Tsutsui, S., Pelikan, M., and Goldberg, D. E. (2006). Node Histogram vs. Edge Histogram: A Comparison of PMBGAs in Permutation Domains. Technical report, Medal.

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

[183] Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2000). Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60.

[184] Vempati, V. S., Chen, C.-L., and Bullington, S. F. (1993). An effective heuristic for flow shop problems with total flow time as criterion. *Computers & Industrial Engineering*, 25(1-4):219 – 222.

[185] Wang, C., Chu, C., and Proth, J.-M. (1995). A branch-and-bound algorithm for n-job two machine flow shop scheduling problems. In *Proceedings of the IEEE Symposium on Emerging Technologies and Factory Automation*, volume 2 of *ETFA '95*, pages 375 –383.

[186] Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336.

[187] Whitley, D., Sutton, A. M., and Howe, A. E. (2008). Understanding elementary landscapes. In *Proceedings of the 10th Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 585–592.

[188] Woo, H.-S. and Yim, D.-S. (1998). A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers & Operations Research*, 25(3):175 – 182.

[189] Xu, X., Xu, Z., and Gu, X. (2011). An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. *Expert Systems with Applications*, 38(7):7970 – 7979.

[190] Yao, G. and Böckenholt, U. (1999). Bayesian estimation of thurstonian ranking models based on the gibbs sampler. *British Journal of Mathematical and Statistical Psychology*, 52(1):79–92.

[191] Yuan, B., Orlowska, M. E., and Sadiq, S. W. (2007). Finding the optimal path in 3D spaces using EDAs - the wireless sensor networks scenario. In *Adaptive and Natural Computing Algorithms*, pages 536–545. Springer.

[192] Zhang, Q. and Li, H. (2007). MOEA/D: a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731.

[193] Zhang, Q., Sun, J., Tsang, E., and Ford, J. (2004). Combination of Guided Local Search and Estimation of Distribution Algorithm for Solving Quadratic Assignment Problem. In *Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 42–48.

[194] Zhang, Q., Sun, J., Tsang, E., and Ford, J. (2006). Estimation of Distribution Algorithm with 2-opt Local Search for the Quadratic Assignment Problem. *Studies in Fuzziness and Soft Computing*, 192/2006:281–292.

[195] Zhang, Y., Li, X., and Wang, Q. (2009a). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, 196(3):869–876.

[196] Zhang, Y., Li, X., Wang, Q., and Zhu, J. (2009b). Similarity based ant-colony algorithm for permutation flowshop scheduling problems with total flowtime minimization. *International Conference on Computer Supported Cooperative Work in Design*, 0:582–589.

[197] Zhang, Y., Li, X., Zhu, J., and Wang, Q. (2008). Composite heuristic algorithm for permutation flowshop scheduling problems with total flow-

time minimization. In *Proceedings of the 12th International Conference on Computer Supported Cooperative Work in Design*, CSCWD 2008, pages 903 –907.

[198] Zhigljavsky, A. A. (1991). *Theory of Global Random Search.* Kluwer Academic Publishers.

[199] Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In Giannakoglou, K. et al., editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, pages 95–100. International Center for Numerical Methods in Engineering (CIMNE).


Kaiet Bengoetxea 2014