Intelligent Systems Group

informatika fakultatea facultad de informática

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

Instances of Combinatorial Optimization Problems: Complexity and Generation

by

Leticia Hernando Rodríguez

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, May 22, 2015

To David
and my parents

ACKNOWLEDGEMENTS

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 have been essential in the development of this dissertation. Thank you very much.

I thank my colleagues in the Intelligent Systems Group with whom I have shared these years, and also the members of other research groups at the Faculty of Computer Science. Especially, I would like to thank Itzi and Borja because of their help with the design of this dissertation cover; Jose, because of his patience teaching me; Unai, because of the "A8s" ; and those who were in my lab 309 when I started this long journey: Carlos, Juan, Jonathan and Ekhiñe.

I also owe a debt of gratitude to Prof. Jing Liu, for allowing me to complete a stay as a visitor in her research group at the Key Lab of Intelligent Perception and Image Understanding of Ministry of Education of China at Xidian University.

I am grateful to the Basque Government for the financial support during these years, and to the funding institutions during the years of this thesis.

I would like to thank my friends in Bilbao for being next to me since we were children, and especially, during each day of this thesis. Also Itsaso, thank you very much. I lived with your peculiar laugh, your constant good humor and your energy that helped me and cheered me up day after day. You are part of this work.

Quiero dedicar unas líneas a ellas dos, que han estado y están presentes no solo en mi tesis, sino en la de todo doctorando de esta facultad: Josune y Amparo. Porque en cada descansito estáis ahí, por esos ratos de risas y de ayudarme a desconectar, por esa mermelada caducada y ese chorrito de licor en la CocaCola. Eskerrik asko Josune! Obrigado Amparo!

Special thanks go to my family: my *Tía Gema*, my brother and sister, and, of course, my parents. *Gracias por todo vuestro apoyo, preocupación y por creer en mi. A mis padres, sobre todo, gracias por poder contar con vosotros en todo momento y por la educación que me habéis dado, sin ella no habría llegado hasta aqui.*

I devote these last lines to give thanks to the person who, without any doubt, has suffered most this thesis. Thank you for listening and encouraging me, thank you for having patience and for believing in me, thank you for being with me in the sad as well as the happy moments of this thesis. I know these have been the most difficult years of your life, and beyond everything, you have been here, always next to me, bearing me... Everything that I could say is not enough. Every word of this thesis, from the first to the last one, has been written thinking of you, especially, the chapter that mysteriously disappeared... Thank you very much David!!

Para terminar, solo me falta nombrar a un pequeño ser al que agradezco esta tesis. Gracias a ti, por tu compañía, por lo cariñoso, lo simpático y lo guapo que eres, gracias por tu RONroneo...

Contents

1 Introduction

1.1 Preliminaries

Combinatorial optimization is related to operations research, algorithm theory, and computational complexity theory. It has important applications in several fields, including artificial intelligence, mathematics, and software engineering. A Combinatorial Optimization Problem (COP) consists of finding the points that minimize (or maximize) a function f, subject to a set of constraints, where the solutions are in a finite or countable infinite search space [69]. That is, find π such that:

$$\begin{array}{ll} min\ (max) & f(\pi), \ \ \pi \in \Omega \\ subject\ \ to & \\ \quad g_i(\pi) \geq b_i, & i = 1, 2, \ldots, m \\ \quad h_j(\pi) = c_j, & j = 1, 2, \ldots, s \end{array}$$

where f, g_i, and h_j are general functions of π, and Ω is a finite or countable infinite set. In particular, in this dissertation we focus on permutation-based COPs. So, from now on, we refer to Ω as the space of permutations of size n.

Although this is the most common definition of a COP, there exist three different versions for these problems, depending on what we are looking for. Without loss of generality, we detail these three versions assuming minimization. First, the optimization problem itself is what is called the *optimization version*. We simplify its definition describing the objective as finding an optimal solution π^* such that:

$$\pi^* = \arg\min_{\pi \in \Omega} f(\pi).$$

Secondly, if the aim of the problem is to know the cost of the optimal solution, that is, the value of the objective function in the optimal point, it is referred to as the *evaluation version*. Given a COP, the goal is to find $f(\pi^*)$ such that:

$$f(\pi^*) \leq f(\pi), \qquad \forall \pi \in \Omega.$$

The third version of a COP is particularly important in studying the complexity of the problem. This is the *recognition version*. Given an instance and a fixed integer L, the *recognition version* consists of determining if there is a feasible solution such that the objective function evaluated in that point is lower than or equal to L. That is, given a COP and $L \in \mathbb{Z}$, it consists of answering the following question:

$$\text{Is there any } \pi \in \Omega \text{ such that } f(\pi) \leq L?$$

Unlike the two previously introduced versions, the *recognition version* is in fact a question which has a yes or no answer. Obviously, solving the *recognition version* of the problem is not harder than solving the *evaluation version*. If we know the solution of the *evaluation version*, we just have to check if the value is lower than or equal to L and the question of the *recognition version* will be answered. It is also possible to show that the *evaluation version* can be solved efficiently whenever the *recognition version* can. If we solve the *recognition version* for many values of L, we can find the exact value for which the solution of the instance changes from yes to no, and that value of L will be the solution of the evaluation version. Moreover, it is known that solving the *evaluation version* is not harder than solving the *optimization* one. But, although there are problems for which the three versions are equivalent, there is no known general method for solving the *optimization version* of the problems by making use of an algorithm that efficiently solves the *evaluation version*.

During the last 30 years, the analysis and solution of COPs has been considered as one of the biggest challenges in mathematics. A first step in this analysis was the classification of the recognition version of these problems in terms of its algorithmic complexity [38]. Roughly speaking, two class of problems were detected. On the one hand, a set of problems (the P class) was detected, which can be solved by means of a polynomial-time algorithm. This class can be defined in terms of any mathematical formalism for algorithms, such as the Turing machine. On the other hand, inside the NP-complete class we find the problems for which there is no known polynomial-time algorithm for solving all their instances and, if there was an efficient algorithm that solved all the instances of one of the problems in the NP-complete class, then all the instances of the NP-complete problems could be solved by an efficient algorithm. Some famous examples of NP-complete problems are the Traveling Salesman problem, the Satisfiability problem, the Knapsack problem or the Graph Coloring problem [69].

Although this complexity classification is very relevant, it has its limitations. Firstly, it is a classification based on the worst case scenario. This means that, for the problems in the NP-complete class, there is no known algorithm that solves all the instances of the problem in polynomial time. However, it can be very easy to solve instances for which algorithms find a solution very quickly. Secondly, these results of complexity are not applied to the optimization problem directly, but to the recognition version of the problem.

Although, as previously mentioned, the recognition problem is not harder than the optimization problem, some of the concepts and techniques used to analyze the recognition version are not easy to transfer to the optimization version.

A large group of researchers has carried out a more detailed complexity analysis of the recognition problem, trying to assign a complexity measure to particular instances, instead of assigning it to the complete problem [1, 41, 45, 49, 64, 65]. These studies focused on the Satisfiability and on the Graph Coloring problem. One of the characteristics observed in these works is a phenomenon called phase transition. This term comes from the field of Physics but it has been moved to other disciplines such as Mathematics, and particularly, combinatorial optimization. In general, a phase transition in a system means a dramatic change in its properties when a control parameter exceeds a critical value. In COPs, this phenomenon means that there is a parameter of the problem for which, for a certain value of it, there is a dramatic change in the complexity of the instances. That is, for very small (or large) values of the parameter, we find that the instances are very easy, while for large (or small) values, they are very hard. Instances with values of that control parameter close to the phase transition threshold can not easily be proved easy or hard. This approach is very important, as it can help to guess the difficulty of a given instance before trying to solve it. As an example, for the recognition version of the Traveling Salesman problem, and using random instances, the authors in [40] represented in Figure 1.1 the probability of finding a tour according to different values of a control parameter, and the phase transition phenomenon was observed.

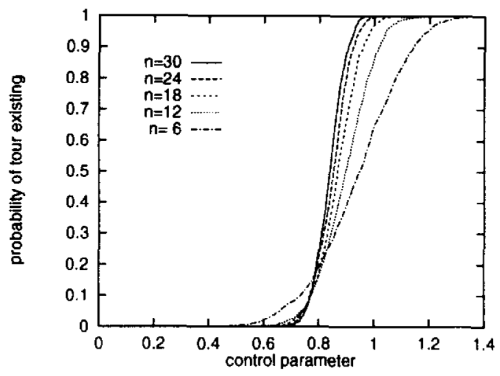


Fig. 1.1. Figure taken from [40], where the authors represented, for the instances of the recognition version of the Traveling Salesman Problem, the phase transition in the probability of finding a tour when varying the values of a control parameter, for different permutation sizes n.

An interesting study would be to move these concepts of complexity of instances and phase transition from the recognition to the optimization version of the problem. Unfortunately, there is a huge difference between determining the complexity for instances of both versions. For example, to determine complexity for an instance of the recognition problem, we do not need to consider a particular algorithm, because the complexity of the instance is well determined by answering the yes-no question of the problem. However, in the optimization problem, the complexity criterion established for an instance directly depends on the algorithm chosen to solve it. This would mean that we should establish one complexity criterion for each algorithm, but this would be tedious due to the huge amount of algorithms available in the literature. A way to approach this issue would be to associate a structure to the search space in the hope that every algorithm that uses the same structure of the space will have the same (or almost similar) behavior, and therefore the complexity of the instances will be determined by this structure, independently of the algorithm that solves it.

In this sense, the research community has studied the complexity of COPs once equipped with a neighbor system. A neighborhood N in a search space Ω is a mapping that assigns a set of neighbor solutions $N(\pi) \in \mathcal{P}(\Omega)$ to each solution $\pi \in \Omega$:

$$\begin{aligned} N : \Omega &\longrightarrow \mathcal{P}(\Omega) \\ \pi &\longmapsto N(\pi) \, . \end{aligned}$$

As an example, in Figure 1.2 we represent all the space of permutations of size 4, structured under the adjacent swap neighborhood. That is, two permutations are connected if they differ from one and only one adjacent swap.

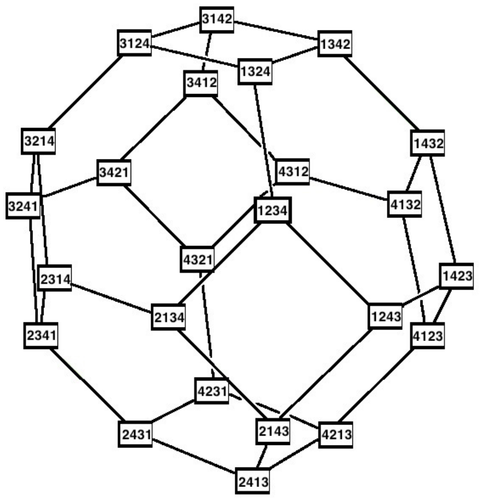


Fig. 1.2. Space of permutations of size 4 connected according to the adjacent swap neighborhood.

Based on the definition of neighborhood, we say that a solution $\pi^* \in \Omega$ is a local optimum if

$$f(\pi^*) \leq f(\pi), \; \forall \pi \in N(\pi^*) \text{ (local minimum).}$$

Obviously, one solution $\pi^* \in \Omega$ can be a local optimum under a neighborhood N_1, but not when considering a different neighborhood N_2. If $\sigma^* \in \Omega$ is a solution such that

$$f(\sigma^*) \leq f(\sigma), \; \forall \sigma \in \Omega,$$

then σ^* is a global optimum. Clearly, the global optima are local optima for any neighborhood.

In general, the triple (Ω, f, N) is called a landscape, where Ω is the search space, f is the objective function of the optimization problem and N is the neighbor system. Obviously, there exist different landscapes (with different complexities) for the same problem, depending on the neighborhood chosen.

Local search algorithms are metaheuristics commonly used to solve instances of permutation-based COPs, which make use of this concept of neighborhood. Particularly, in this dissertation, we use a deterministic best-improvement local search. The steps followed by this algorithm are specified in Algorithm 1, where it is assumed that we are dealing with a minimization problem. In the case of a maximization problem, we just replace "**if** $f(\sigma_i) < f(\pi^*)$" (step 6) with "**if** $f(\sigma_i) > f(\pi^*)$". It is important to notice that the neighbors are evaluated in a specific order, so that, in the case of two neighbors having the same function value, the algorithm will always choose that which was encountered first. We denote by $\mathcal{H}$ the operator that associates, to each solution π, the local optimum π^* obtained after applying the algorithm ($\mathcal{H}(\pi) = \pi^*$).

Algorithm 1 Deterministic best-improvement local search algorithm (assuming a minimization problem)

```
 1: Choose an initial solution $\pi \in \Omega$
 2: repeat
 3:     $\pi^* = \pi$
 4:     for $i = 1 \to |N(\pi^*)|$ do
 5:         Choose $\sigma_i \in N(\pi^*)$
 6:         if $f(\sigma_i) < f(\pi)$ then
 7:             $\pi = \sigma_i$
 8:         end if
 9:     end for
10: until $\pi = \pi^*$
```

The number of local optima that the neighborhood imposes on the search space has attracted much attention because it also seems to be related to the difficulty of finding the global optima [2, 3, 14, 30, 31, 39, 43, 71, 72]. That is why, the number of local optima has been considered as an indirect complexity measure of an instance when solving it using a local search algorithm. Moreover, as the number of local optima depends on the neighborhood, it would be useful to take it into account in order to choose a priori the most suitable neighborhood to solve a particular problem instance efficiently.

Unfortunately, in practice, given an instance of a COP and a neighborhood, we do not know in advance the number of local optima. Thus, the development of methods that efficiently estimate the number of local optima seems to be a requirement in order to design algorithms that work in the right neighborhood. While several approaches have been proposed to estimate and bound the expected number of local optima for particular class of instances of COPs [2, 3, 43], the literature is not so extensive when it is about estimating the number of local optima of an individual instance. A key aspect to take into account when designing a method to estimate the number of local optima of an instance is the distribution of the sizes of the attraction basins.

Roughly speaking, an attraction basin $\mathcal{B}(\pi^*)$ is composed of all the solutions that, after applying a local search algorithm starting from these solutions, finishes in π^*. So, the attraction basin $\mathcal{B}(\pi^*)$ of a local optimum π^* is the set that can be defined in the following way:

$$\mathcal{B}(\pi^*) = \{\pi \in \Omega \mid \mathcal{H}(\pi) = \pi^*\}.$$

The basin of attraction of a local optimum clearly depends on the neighborhood, however, we omit it in order to simplify the notation. Supposing that $\pi_1^*, \pi_2^*, \ldots, \pi_v^*$ are all the local optima of the landscape, three important properties that the attraction basins fulfill are the following:

1. $\mathcal{B}(\pi_i^*) \neq \emptyset, \forall i$
2. $\mathcal{B}(\pi_i^*) \cap \mathcal{B}(\pi_j^*) = \emptyset, \forall i \neq j$
3. $\bigcup_{i=1}^{v} \mathcal{B}(\pi_i^*) = \Omega$

So, the set of attraction basins of the local optima defines a partition of Ω. The relative size of the attraction basin $\mathcal{B}(\pi^*)$ with respect to the search space Ω, i.e., the proportion of solutions of the whole search space that belong to the basin $\mathcal{B}(\pi^*)$, denoted as $p = \frac{|\mathcal{B}(\pi^*)|}{|\Omega|}$, is of special relevance.

Having information about all these properties that the neighborhood imposes on the instance: the number of local optima, their distribution along the search space, the sizes of their attraction basins, etc., is relevant for studying the performance and efficiency of local search algorithms. So, it could happen that an algorithm that uses a specific neighborhood works well when it is applied to an instance, but the same algorithm with the same neighborhood is not efficient for a different instance. In the first case, we say that the instance is easy for the algorithm, while in the second case we say that it becomes hard. Therefore, when comparing different algorithms, or when proposing a new technique that solves an instance, it is interesting to have at our disposal instances with diverse characteristics, and, thus, different levels of complexity for the algorithms.

However, it is difficult to find sets of instances of COPs and neighborhoods whose properties are known beforehand. Furthermore, there are well-known benchmarks of instances of different problems whose characteristics, such as the number of local optima and their attraction basins, are not known. Therefore, it is difficult to a priori guess the performance of the algorithms without the knowledge of the characteristics of the instances. In order to solve this deficiency, there exist proposals of generators of instances, where the aim is to create instances with specific and fixed properties that help to study the behavior of the algorithms.

Unfortunately, the portfolio of models that generate customized optimization problem instances is not very extensive. Most of the proposals found in the literature for this topic are for the continuous domain [37, 66, 67, 75, 95]. In the discrete domain, and particularly for binary spaces, we found a proposal in [25]. In the space of permutations, we can find the following generators of instances [7, 23, 70, 87], and only in some of them are the authors able to create instances where the global optimum is known. Moreover, this is the only information provided, and these generators lack the flexibility to generate instances with controlled properties. So, there is no clue about how easy or difficult it is to solve the instance. Precisely, in [70], the authors state that the toughest technical challenge to evaluate how close heuristic algorithms come to the optimum is finding (or generating) suitable test instances where applying those algorithms.

1.2 Permutation-based Combinatorial Optimization Problems

Throughout this dissertation, three different permutation-based COPs will be used. We denote by Ω the set of permutations of size n, and define a permutation $\pi \in \Omega$ as a bijection of the set of integers $\{1, 2, \ldots, n\}$ onto itself. A permutation is understood as an order of the items $\{1, 2, \ldots, n\}$, this is:

$$\pi = (\pi(1)\pi(2) \cdots \pi(n))$$

where $\pi(i) \in \{1, 2, ..., n\}$ is the item in the i-th position and $\pi(i) \neq \pi(j), \forall i \neq j$.

1.2.1 Traveling Salesman Problem

Given a list of cities and their pairwise distances, the aim of the TSP is to find the shortest tour that visits each city exactly once, returning to the initial city. In particular, we work with instances of the Symmetric Traveling Salesman

Problem, where the distance from the city A to the city B is considered the same as from B to A.

Taking into account that the problem has n cities, the search space Ω comes specified by the set of permutations of n elements, and the objective function to minimize is:

$$F(\pi) = \textstyle\sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)},$$

where $d_{\pi(i)\pi(j)}$ represents the distance between the cities $\pi(i)$ and $\pi(j), i \neq j$.

Note that, in this problem, one solution can be represented by $2n$ different permutations. Therefore the search space is of size $n!/2n$. For example, in Figure 1.3, for $n = 5$, the 10 permutations $\pi_1, \ldots, \pi_{10}$ represent the same tour.

$$\begin{array}{ll}
\pi_1 = (14325) & \pi_6 = (52341) \\
\pi_2 = (43251) & \pi_7 = (23415) \\
\pi_3 = (32514) & \pi_8 = (34152) \\
\pi_4 = (25143) & \pi_9 = (41523) \\
\pi_5 = (51432) & \pi_{10} = (15234)
\end{array}$$

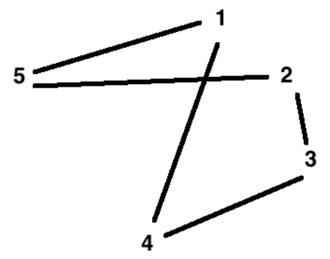


Fig. 1.3. Tour of length 5 represented by the permutations $\pi_1, \pi_2, \ldots, \pi_{10}$.

1.2.2 Permutation Flowshop Scheduling Problem

In the Permutation Flowshop Scheduling Problem (PFSP), n jobs have to be scheduled on m machines in such a way that a criterion is minimized. A job consists of m operations, and the j-th operation $(j = 1, \ldots, m)$ of each job must be processed on machine j for a given specific processing time without interruption. The processing times are fixed non-negative values, and every job is available at time zero. At a given time, a job can start on the j-th machine when its $(j-1)$-th operation has finished on the machine $(j-1)$, and machine j is idle.

The makespan is the total length of the schedule and, traditionally, has been the criterion to be optimized in the PFSP. However, recently, Total Flow Time (TFT) has captured the attention of the scientific community since it is more meaningful for the current industry, and thus, this criterion will be used in this dissertation. The following formula expresses mathematically the concept of TFT for a permutation π of jobs, where $c_{\pi(i),m}$ stands for the completion time of job $\pi(i)$ $(i = 1, \ldots, n)$ at machine m:

$$F(\pi) = \sum_{i=1}^{n} c_{\pi(i),m}.$$

Being $p_{\pi(i),j}$ the processing time required by job $\pi(i)$ on machine j, the completion time of job $\pi(i)$ on machine j can be recursively calculated as:

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

Note that the search space in this case and in the next problem has size $n!$.

1.2.3 Linear Ordering Problem

Given a matrix $B = [b_{ij}]_{n\times n}$ of numerical entries, the Linear Ordering Problem (LOP) consists of finding a simultaneous permutation π 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:

$$F(\pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{\pi(i)\pi(j)}$$

where $\pi(i)$ ($\pi(j)$) denotes the index of the row (column) ranked at position i (j) in the solution π.

1.2.4 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is the problem of allocating a set of facilities to a set of locations, with a cost function associated to the distance and the 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 $n \times n$ input matrices with real values $\mathbf{H} = [h_{ij}]$ and $\mathbf{D} = [d_{kl}]$, where h_{ij} is the flow between facility i and facility j and d_{kl} is the distance between location k and location l. Given n facilities, the solution of the QAP is codified as a permutation $\pi = (\pi(1)\pi(2)\cdots\pi(n))$ where each $\pi(i)$ $(i = 1, \ldots, n)$ represents the facility that is allocated to the i-th location. The fitness of the permutation is given by the following objective function:

$$F(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} \cdot d_{\pi(i)\pi(j)}$$

1.3 Neighborhoods

The experiments carried out in this dissertation are based on three different neighborhoods that have been commonly used in the COPs literature. These are: the adjacent swap, the swap and the insert neighborhoods.

1.3.1 Adjacent swap

Given a solution $\pi = (\pi(1)\pi(2)\cdots\pi(n))$, its adjacent swap neighbors are those permutations obtained as the result of swapping two adjacent elements of such solution π:

$$\begin{aligned} N_A(\pi(1)\pi(2)\cdots\pi(n)) = \{(\pi'(1)\pi'(2)\cdots\pi'(n)) \mid \pi'(k) = \pi(k), \forall k \neq i, i+1, \\ \pi'(i) = \pi(i+1), \pi'(i+1) = \pi(i)\}\,. \end{aligned}$$

The number of adjacent swap neighbors of a permutation of size n is $n-1$.

For example, in the space of permutations of size $n = 5$, the set of adjacent swap neighbors of the permutation $\pi = (12345)$ is:

$$N_A(12345) = \{(21345), (13245), (12435), (12354)\}.$$

1.3.2 Swap

The swap or 2-exchange neighborhood considers that two solutions are neighbors if one is generated by swapping two elements of the other, not necessarily adjacent:

$$\begin{aligned} N_S(\pi(1)\pi(2)\cdots\pi(n)) = \{(\pi'(1)\pi'(2)\cdots\pi'(n)) \mid \pi'(k) = \pi(k), \forall k \neq i, j, \\ \pi'(i) = \pi(j), \pi'(j) = \pi(i), i \neq j\}\,. \end{aligned}$$

Under this neighborhood, a solution has $n(n-1)/2$ neighbors.

Taking the same permutation $\pi = (12345)$ as in the previous case, the set formed by its neighbors under the swap neighborhood is:

$$\begin{aligned} N_S(12345) = \{\, & (21345), (32145), (42315), (52341), (13245), \\ & (14325), (15342), (12435), (12543), (12354)\}. \end{aligned}$$

1.3.3 Insert

Two solutions are neighbors under the insert neighborhood if one is the result of moving an element of the other one to a different position:

$$\begin{aligned} N_I(\pi(1)\pi(2)\cdots\pi(n)) = & \big\{(\pi'(1)\pi'(2)\cdots\pi'(n)) \mid \pi'(k) = \pi(k), \forall k < i \text{ and } \forall k > j, \\ & \quad \pi'(k) = \pi(k+1), \forall i \leq k < j, \pi'(j) = \pi(i)\big\} \\ & \cup \big\{(\pi'(1)\pi'(2)\cdots\pi'(n)) \mid \pi'(k) = \pi(k), \forall k < i \text{ and } \forall k > j, \\ & \quad \pi'(i) = \pi(j), \pi'(k) = \pi(k-1), \forall i < k \leq j\big\}\,. \end{aligned}$$

The number of neighbors of a solution under the insert neighborhood is $n(n-1) - (n-1) = (n-1)^2$.

Following with the same example, the set composed with the insert neighbors of the permutation $\pi = (12345)$ is the following:

$$
\begin{aligned}
N_I(12345) = \{ & (21345), (23145), (23415), (23451), \\
& (13245), (13425), (13452), (31245), \\
& (12435), (12453), (41235), (14235), \\
& (12354), (51234), (15234), (12534)\}.
\end{aligned}
$$

1.4 Outlook of the dissertation

This thesis is organized in three parts: (i) defining and estimating complexity measures of instances of combinatorial optimization problems, (ii) designing a tunable generator of instances of permutation-based combinatorial optimization problems, and (iii) general conclusions, future work and publications derived from the thesis.

The first part of the dissertation, divided in 3 chapters, is devoted to establish and estimate complexity criteria for the instances of COPs. These measures do not depend on the type of problem or the algorithm used to solve them, but just on the chosen structure for the search space, and would be useful to know, in advance, if the structure chosen is suitable for efficiently solving a given instance. In Chapter 2, we define two complexity measures: the attraction basin size of the global optimum and the number of local optima. Particularly, for the TSP under the swap neighborhood, we study the evolution of these descriptors as the size of the problem grows, and we find a phase transition phenomenon in the complexity of the instances. While studying these complexity criteria, we noticed the difficulty of calculating the exact number of local optima when the size of the problem is medium-large. So, in Chapter 3, we review the existing methods for estimating the number of local optima, as well as we bring to the optimization field some of methods that come from the statistics arena and that have been used by biologists and ecologists to estimate the number of species in a population. We test the different estimation methods in different instances: artificial, randomly generated, real and those obtained from well-known benchmarks. We find that the performance of the methods depends on properties of the instances, such as the distribution of the sizes of the attraction basins. As a general conclusion the estimation methods that come from the statistics arena, provide better results than those already known in optimization field, particularly, in the case of small sample sizes. From this analysis, we learn that the relative sizes of the attraction basins of the local optima are a main component in the estimation of the number of local optima of the instances. Therefore, in order to improve them, the development of accurate methods that estimate these sizes of the attraction basins is fundamental. In Chapter 4, we propose two methods to estimate the attraction basin sizes of the local optima. The results obtained with both of them for instances of different problems and considering different neighborhoods are analyzed.

In the second part of the dissertation, Chapter 5 proposes a generator of instances of permutation-based COPs. Our main concern was to provide a highly tunable tool. In this sense in our generator the researcher can choose the set of permutations she/he wants to be the local optima (including the global optima) together with additional parameters that influence the sizes of the basin of attraction of the local optima. The generator is defined as a linear programing problem, where the function to optimize also helps to obtain qualitative properties of the instances to generate. We also prove that, with our generator, we are able to create instances comparable to benchmark instances in terms of locality, and we also study the influence of the different parameters, as well as give examples of the utility of the instances produced by the generator when analyzing the behavior of different algorithms. We remark that the most outstanding characteristic of our generator is that, by tuning the parameters involved in it, we are able to control the properties of the output instances. This allows us to conduct a deep analysis of the performance of the optimization algorithms.

Finally, the third part draws the general conclusions of the dissertation, points out possible future works and lists the publications produced during this thesis.

Part I

Measuring complexity of instances of combinatorial optimization problems

2

Complexity measures: Attraction basin size of the global optimum and number of local optima

2.1 Introduction

Permutation-based combinatorial optimization considers problems where the objective is to find the permutation (or permutations) that maximizes or minimizes an objective function [69]. The solution of these problems is of great importance, because they naturally appear in different fields such as science, engineering or industry. One of the most paradigmatic examples of this type of problems is the Traveling Salesman Problem (TSP): given a list of n cities and a matrix with the distances between each pair of cities, the objective of the TSP is to find a tour that visits every city exactly once, with minimal total length.

Metaheuristic algorithms have been proved as efficient methods for solving hard permutation-based COPs. Most of these methods are based on, or use a kind of local search that relies on the neighborhood structure over the search space. The properties of this neighborhood can cause dramatical differences in the performance of those local search methods [35, 48, 55, 63, 74, 86]. Thus, these properties determine the complexity that the algorithms using that neighborhoods will find when solving a specific instance.

Literature gathers several papers related to complexity measures for instances of COPs. First, we find papers that propose complexity measures for particular algorithms or specific operators used inside a given algorithm. A first study about the complexity of TSP instances was developed in [40] where, starting from random instances, the number of nodes that appeared when it was solved with a Branch & Bound algorithm was assigned as a complexity measure. This type of complexity measure depends on the algorithm, but there are also studies in which the measures deal with the operators. In [24, 68] fitness distance correlation was used as a complexity measure, mainly related to Genetic Algorithms. This function measures the correlation between the value of the objective function and the distance to the nearest global optimum. In [27], the point quality is proposed as a local complexity measure, where distance is measured as a function of the neighborhood and it is analyzed for five different neighborhoods [77]. Another paper related to the analysis of complexity is [86], where the authors proposed a network characterization of NK landscapes, constructing a graph whose vertices represent the local maxima in the landscape, and the edges account for the transition probabilities between their corresponding basins of attraction. Such networks were exhaustively extracted on representative NK landscape instances, and statistical characterization of their properties were performed. For the TSP, it has been proved that with the swap neighbor system it is an elementary landscape [22, 81, 94]. So, we can obtain the autocorrelation coefficient and the autocorrelation length of the landscape, which are parameters that measure the ruggedness of a landscape [6]. These parameters are useful to compare landscapes of the same problem in terms of its ruggedness and to choose a suitable neighborhood. Unfortunately, these measures give characteristics of the whole landscape, and therefore are not valid to compare specific instances of the same landscape.

In this study we focus on local search algorithms. As we have previously mentioned, the same local search algorithm can produce different results in the same instance depending on the neighborhood chosen. So, the difficulty that a local search finds when solving an instance of a COP, comes specified by the properties that the neighborhood used in the algorithm provokes. Of course, different neighborhoods draw different shapes (ruggedness) in the landscapes, but, in addition, the same neighborhood can cause very different properties for distinct instances of the same problem. Thus, before solving an instance of a COP with a local search algorithm, we should decide which neighborhood is the most convenient for such a specific instance.

In order to measure the complexity of an instance for local search algorithms, we establish different criteria according to the characteristics that the neighborhood imposes on the search space. Mainly, we consider two properties as complexity measures. The first one is the proportion of the size of the basin of attraction of the global optimum over the size of the search space (from now on, we denote this proportion as *ABsize-GO*). We suppose that, if *ABsize-GO* is large, the instance is easy, because the probability of ending at the global optimum is high. Secondly, we choose the proportion of the number of different local optima that appear in an instance over the size of the search space (denoted as *Num-LO*). In this case we assume that, the higher *Num-LO* is, the harder the instance. It is well known that *Num-LO* is a very important property that makes an instance easy or difficult to solve when using a local search algorithm. However, it is possible to find instances with many local optima whose basins of attraction are small in size, while the global optimum has a very large basin. So, the distribution of the sizes of the basins of attraction must also be considered [39].

In order to test the validity of our descriptors, we have conducted some experiments, focusing on the TSP and the swap neighborhood. We have chosen the swap neighbor system because many successful algorithms have been proposed based on it [8, 29, 46, 62, 88]. We are based on [40, 96], where, for the recognition problem of the TSP considering random instances, it was proved that an instance-dependent parameter exists which makes the instance hard or easy. In this case, this parameter is related to the length of the tour L. There also exists a value for this parameter for which the complexity changes rapidly, appearing the phase transition phenomenon. In more recent works [5, 51], the authors also state that for problems such as the Number Partitioning problem and the Satisfiability problem the properties of the instances change as the values of different parameters vary. Therefore, the complexity of the instances varies with these parameters and also the phase transition phenomenon appears. The main difference of our proposed complexity measures with the existing ones is that we establish complexity measures for instances of the optimization problem that do not depend on the algorithm that solves them, and in using them, we find phase transition patterns in the complexity of the instances.

The rest of this chapter is organized as follows. We evaluate the complexity measures in Section 2.2. First, we carry out an analysis of the evolution of our descriptors as the dimension of the problem increases. Next, given a random instance of the TSP optimization problem, we study the probability of this instance having values of the descriptors higher than or equal to certain values, and a phase transition behavior is observed similar to what had been seen in the recognition version [40] but taking our complexity measures as parameters. Finally, in Section 2.3 we conclude and explain some ideas derived from this work, and that are developed in the following chapters.

2.2 Evaluation of the complexity measures

2.2.1 Experimental setup

The goal of the experimentation is to analyze the evolution of our complexity measures and to try to find patterns of phase transition for the TSP under the swap neighbor system. To do this, we base our work on similar ones done for the recognition version of the problem [1, 40, 41, 45, 49, 64, 65]. We work with random instances of the TSP. Based on [40], the instances were created by placing n cities (for $n = 6, 7, ..., 25$) uniformly at random on a square of area 100 in an Euclidean space and calculating the matrix that gives the distance between every pair of cities.

For each n we randomly created 500 instances. The number of local optima and their attraction basins are exhaustively calculated for each instance. In order to do that, Algorithm 1, which chooses the best neighbor solution at each step, is applied starting from each point of the search space. Although we would have liked to calculate our complexity measures (*ABsize-GO* and *Num-LO*) exactly for each instance, this is very time consuming for values of n higher than 15 because of the exponential growth of the size of the search space ($(n-1)!/2$ [1]). This is the reason why the exact descriptors have been calculated only for $n \leq 15$. For $n \geq 16$ we have estimated the values of the sizes of the attraction basins of the local optima by proceeding in the following way: for each instance, we have taken $10!$ random permutations as initial solutions and, for each of these initial solutions, the local search algorithm has been applied. For $n \geq 16$, *ABsize-GO* has been estimated as the proportion of the attraction basin size of the global optimum over the $10!$ permutations. In addition, using an integer programming formulation we checked that the minimum local optimum obtained with the local search in each instance for $n \geq 16$ is exactly the global optimum of the instance.

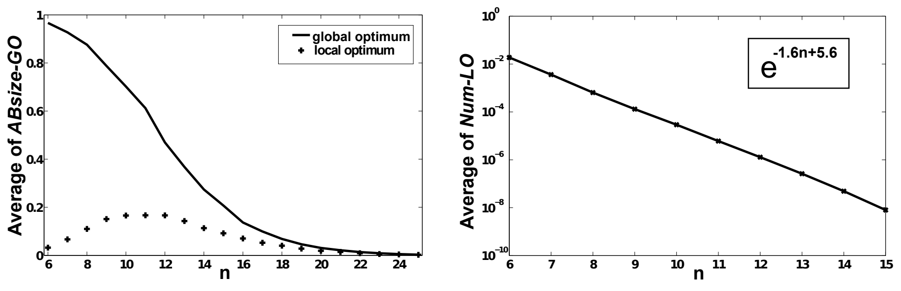
However, for *Num-LO*, we have only worked with exact values, that is, instances with n up to 15. This is because there are no reliable estimators for this parameter. Although there exist many works that explain techniques to estimate the number of local optima [14, 30, 31, 39, 43, 57, 82, 83], these techniques are not applicable in our problem because they can not give us a precise estimation of our parameter for a particular instance. In fact, these estimations are made for the expected number of local optima of all instances of the problem, and not for a specific instance. In Chapter 3, we delve into this matter, and present a deep comparison of methods that estimate the number of local optima of instances of COPs.

2.2.2 Evolution of the complexity measures

In this section the evolution of the complexity descriptors is analyzed in relation to the size of the problem. Figure 2.1(a) shows the average of *ABsize-GO* of the 500 instances for different values of n. It can be observed that, as n increases, this value decreases. While for n = 6 the average of *ABsize-GO* is higher than 0.96, for n = 15 the average is 0.2075, and for n = 25 it is 0.0031. These values show the important decrease of this parameter. The exact values of the averages for each n can be consulted in Table 2.1, as well as the average numbers of local optima that are found for the different permutation sizes. As previously mentioned, we only report this number for $n \leq 15$.

Figure 2.1(b) shows the average of *Num-LO* that are obtained from each of the 500 instances for $n = 6, 7, ..., 15$. Taking into account that the y axis is represented at logarithmic scale, it can be observed that the average *Num-LO* decreases exponentially as n increases. This graph approximates to the following function: $e^{-1.6n+5.6}$. Table 2.2 gives the exact values for this average of *Num-LO*. The important decrease of *Num-LO* can lead us to think that, according to this complexity descriptor, instances are easier as n increases, but this is not true. It must be taken into account that *ABsize-GO* also decreases.

[1] Notice that we are working with cyclic permutations and considering the symmetric TSP. This is why the size of the search space is $(n-1)!/2$ instead of $n!$.



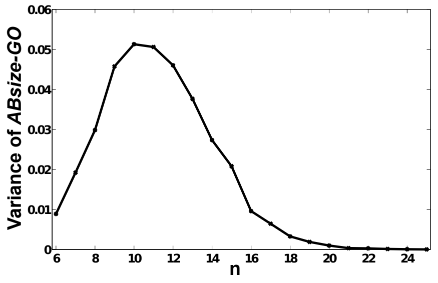
(a) (b)

Fig. 2.1. (a) Average of *ABsize-GO* and the proportions of the maximum basin of attraction of the local optima, which are not the global optima, obtained from each of the 500 instances, against the number of cities n. (b) Average of *Num-LO* that appear in each of the 500 instances, at logarithmic scale, against the number of cities n.

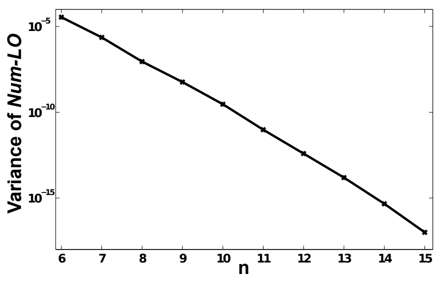
This means that the local optima (which are not the global optimum), have a larger basin of attraction as n grows. This fact leads us to represent, in Figure 2.1(a) plotted with '+', the proportion of the size of the largest basin of attraction of a local optimum that appears in the instance, that is, the maximum basin of attraction of the local optima (not being the global optimum).

Observing the plot, it can be seen that the proportion of the size of the largest basin of attraction of the local optima of an instance decreases for n higher than 11, but much more slowly than that of the global optimum. Table 2.1 (fourth and fifth columns) shows the values that correspond to the proportion of the size of the biggest basin of attraction of the local optima. If we compare the average values (second and fourth columns), it can be observed that, for $n = 6$, there is a huge difference between *ABsize-GO* and the proportion of the maximum attraction basin of a local optimum. However, this difference becomes smaller as n increases, so the size of the basin of attraction of the local optimum approaches to *ABsize-GO* as n grows.

Figure 2.2 represents the variances of our complexity descriptors. For *ABsize-GO* (Figure 2.2(a)), the variance increases for n up to 10, but from then on, it decreases rapidly. Notice that, for low n, no more than two or three local optima are found on average, so this number is considerably low, and this means that some instances have just one optimum (the global one) but other instances have the global optimum plus one or two more local optima. Therefore, there is a huge difference in *ABsize-GO* of these different instances, and thus the variance is also considerably high. As can be observed in Table 2.1, the variances of *ABsize-GO* are, for each n, higher than those of the proportion of the size of the maximum basin of attraction of the local optima. In Figure 2.2(b), an exponential decrease at the variance of *Num-LO* is observed. We have to take into account that we have represented the y axis at logarithmic scale so as to better appreciate it.



(a)



(b)

Fig. 2.2. (a) Variance of *ABsize-GO* obtained from each of the 500 instances, against the number of cities n. (b) Variance of *Num-LO* that appear in each of the 500 instances, at logarithmic scale, against the number of cities n.

Table 2.1. Averages and variances of *ABsize-GO*, and the same for the proportions of the maximum basin of attraction of the local optima. The last column shows the average number of local optima.

	Global optimum		**Local optimum**		**Average number of**
n	**Mean**	**Variance**	**Mean**	**Variance**	**local optima**
6	0.9665	0.008888	0.0324	0.008085	1.13
7	0.9273	0.019201	0.0667	0.015521	1.30
8	0.8760	0.029832	0.1102	0.022005	1.62
9	0.7888	0.045748	0.1512	0.022505	2.65
10	0.7020	0.051290	0.1659	0.015503	5.31
11	0.6123	0.050583	0.1669	0.011993	11.02
12	0.4697	0.046021	0.1665	0.008734	25.68
13	0.3681	0.037620	0.1429	0.004959	62.99
14	0.2740	0.027339	0.1134	0.003004	152.56
15	0.2075	0.020804	0.0920	0.002675	348.71
16	0.1368	0.009592	0.0705	0.001166	—
17	0.0996	0.006441	0.0526	0.000800	—
18	0.0685	0.003271	0.0404	0.000614	—
19	0.0464	0.001889	0.0282	0.000258	—
20	0.0310	0.000988	0.0192	0.000117	—
21	0.0215	0.000340	0.0147	0.000092	—
22	0.0139	0.000272	0.0093	0.000052	—
23	0.0091	0.000131	0.0059	0.000017	—
24	0.0055	0.000050	0.0041	0.000012	—
25	0.0031	0.000021	0.0025	0.000003	—

Table 2.2. Averages and variances of *Num-LO*.

n	**Mean (*e-4)**	**Variance (*e-10)**
6	188.33333	347500.0000000
7	36.16667	23055.2469136
8	6.44444	898.5638700
9	1.31450	57.5919430
10	0.29266	2.9439807
11	0.06074	0.0956951
12	0.01287	0.0039111
13	0.00263	0.0001554
14	0.00049	0.0000046
15	0.00008	0.0000001

2.2.3 Finding phase transitions

Gent and Walsh [40] focused their study on the recognition version of the TSP and found a phase transition behavior when they represented the probability that a tour of length L exists according to a control parameter. We move this concept of complexity from instances of the recognition problem to instances of the optimization problem, taking our complexity descriptors as parameters: *ABsize-GO* and *Num-LO*. First, for each n, we sort in ascending order, the 500 values of *ABsize-GO* obtained from the 500 instances. Secondly, we calculate the empirical distribution function. The same procedure has been used for *Num-LO*.

Figure 2.3 shows, for $n = 6, 7, ..., 25$ from right to left, respectively, the probability of an instance having *ABsize-GO* higher than or equal to the different values represented in the x axis. Notice that this axis is in logarithmic scale. It can be observed that, for high values of n, the probability of the proportion of solutions that reach the global optimum being higher than or equal to the indicated values decreases rapidly. Moreover, this probability tends to be concentrated in smaller values as n increases, showing a slight phase transition phenomenon.

Figure 2.4 shows, for $n = 6, 7, ..., 15$ from right to left, respectively, the probability of an instance having *Num-LO* higher than or equal to the values indicated in x axis. Notice again that this axis is in logarithmic scale. It can be observed that, when our parameter varies, a huge change in probability occurs (it decreases rapidly) and a phase transition phenomenon can be observed. Furthermore, as in the case of *ABsize-GO*, the value for which the transition occurs, becomes smaller as n increases. The higher the value of n is, the sharper the decrease.

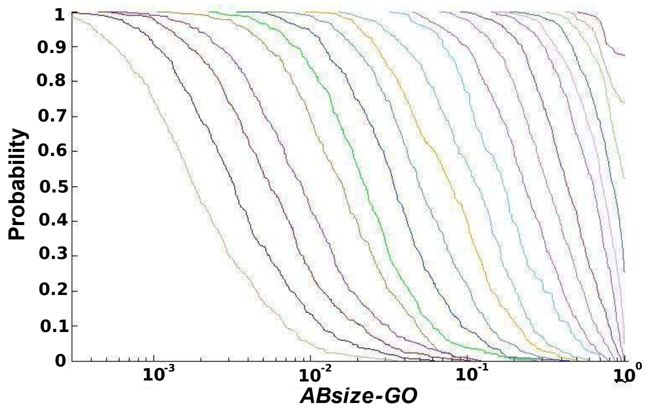


Fig. 2.3. Probability of an instance having *ABsize-GO* higher than or equal to the different values represented in x axis, taking into account that the x axis is in logarithmic scale, for n = 6, 7, ..., 25 from right to left, respectively.

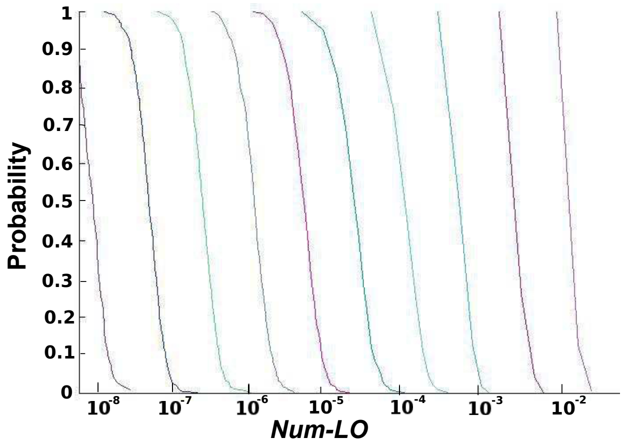


Fig. 2.4. Probability of an instance having *Num-LO* higher than or equal to the values indicated in x axis, taking into account that the x axis is in logarithmic scale, for n = 6, 7, ..., 15 from right to left, respectively.

2.3 Conclusions

In this chapter we have carried out a study about the complexity of random instances of the TSP. The final aim is to find complexity descriptors able to predict the difficulty of the instance for a local search algorithm. In addition, we are also interested in finding a phase transition behavior in the complexity of the instances when varying these complexity descriptors.

Inside combinatorial optimization, there have been many studies focused on the recognition version of different problems trying to find complexity criteria for the instances [1, 40, 41, 45, 49, 64, 65, 96] and, in some of these studies, the phase transition phenomenon is observed. In addition to these works, numerous researchers have worked with metaheuristic algorithms and operators, establishing criteria of complexity [24, 27, 68]. However, in these works the phase transition phenomenon has not been dealt with.

We propose here two measures in order to classify instances of the COPs according to their complexity, particularly for the TSP. First, the proportion of the attraction basin size of the global optimum over the size of the search space: *ABsize-GO*. Secondly, the proportion of the number of local optima over the size of the search space: *Num-LO*. We have studied the evolution of both descriptors as n grows, being n the size of the problem. It has been observed that, as n grows, the average of *ABsize-GO* decreases rapidly. Regarding the variance, an increase up to $n = 10$ has been observed and, for $n > 10$, it decreases very quickly. For values of n close to 10, there is a huge difference between the attraction basin size of the global optimum of the different instances. So, for these values of n some instances will be considered very easy to be solved, and there are also instances that will be considered very hard to be solved. This phenomenon deserves further research as it could be related to the process used for generating the instances.

For *Num-LO*, there is a very fast decrease as n increases, and this also happens for the variance. According to this complexity measure, this would indicate that, as n grows instances are easier. However, it is important to take into account that, as n increases, *ABsize-GO* decreases, so the basins of attraction of the rest of the local optima (that are not the global optimum) have to increase. In fact, as n grows, the average sizes of the basins of attraction of the local optima approach to that of the global optimum.

Finally, in this chapter, and based on experiments carried out in [40], we have found a phase transition on the complexity of the instances, taking as parameters our two complexity measures. This phenomenon has appeared when representing the probability of the instances having values of the complexity measures higher than or equal to certain values. For our first descriptor, and for high values of n, this probability decreases rapidly and tends to be concentrated in smaller values of the descriptor as n increases. In the case *Num-LO*, with a small variation of our parameter, a fast decrease occurs also in probability. Moreover, as n increases, the value of our parameter for which this huge change occurs decreases and the probability decreases more quickly.

In this chapter, we have shown that the proposed measures are in relation with the complexity of the instances. However, it is computationally unaffordable to calculate them even for moderate values of n. Moreover, if we were able to calculate *ABsize-GO*, this would mean that we would know the global optimum. Therefore, we extend this initial work looking for techniques that can be applied to estimate the number of local optima (Chapter 3) and the attraction basin sizes of these local optima (Chapter 4).

3

An evaluation of methods for estimating the number of local optima in combinatorial optimization problems

3.1 Introduction

There are several characteristics of a neighborhood that influence the behavior of a local search algorithm. However, probably, the most relevant is the number of local optima it generates [35, 48, 55, 63, 74, 86]. Therefore, the knowledge about the number of local optima that a neighborhood generates in an instance of a COP can have a high impact on the choice of the local search algorithm used to solve the instance. On the first hand, different neighborhoods generate different number of local optima. Of course, as a general rule, the higher the size of the neighborhood the lower the number of local optima. But, given two neighborhoods of the same size complexity (number of local solutions in the same order of magnitude), the one that generates a lower number of local optima will always be preferred.

Given an instance of a COP and a neighborhood, the exact calculation of the number of local optima is impractical, except for extremely low dimensions ($n \leq 14$). Furthermore, this exact calculation requires, in most of the cases, the exhaustive inspection of every solution in the search space, what makes this approach usefulness. Therefore, we have to resort in statistical estimation methods in order to have an approximation of the number of local optima of a landscape. However, even if we were able to exactly calculate the number of local optima, we still would have to face another challenge, that is, how to represent such number. When the dimension of a problem is high (for instance, $n \geq 200$) and the size of the search space is exponential in n, the number of local optima are usually so high that can not be accurately represented in the computer. The alternative choice to represent the proportion of solutions of the search space that are local optima deals with similar issues as this number is too small to be accurately represented. Therefore, the estimation of the number of local optima of an instance of a COP is only plausible for moderate values of the problem size.

In spite of the useful information that the knowledge of the number of local optima can provide in order to choose the best algorithm for solving a

COP instance, there have not been many proposals in the literature. This can probably be due to the difficulty of the estimation problem. These proposals are mainly divided into three groups. A first group of proposals try to find expected values on the number of local optima, departing from the hypothesis that the instances have been generated uniformly at random [2, 3, 43]. In a second group we mainly include proposals that have been developed in the metaheuristics community. Finally, a third group includes a set of proposals that were not designed with the objective of estimating the number of local optima, but were adapted to this problem.

Most of these approaches assume that the attraction basins are equally-sized [14, 31], and they propose methods to obtain lower bounds for the number of local optima. Under this assumption on the attraction basins, there are works where biased estimators are obtained, and they try to correct this bias to provide an unbiased estimator [30, 71, 72]. On the other hand, there are papers where the sizes of the attraction basins are assumed to fit a certain type of parametric distribution, such as gamma or lognormal. For example, in [39], the authors assume a gamma distribution for the relative sizes of the attraction basins.

So, the estimation of the number of local optima can help, not only to measure the complexity of the instance, but also to choose the most convenient neighborhood to solve it. This work is connected to the area of evolutionary computation as the analysis of the set of local optima in a landscape associated to the problem can be related to the investigation of some properties of evolutionary algorithms, such as properties of the stable steady-state points in Genetic Algorithms [73, 89].

The problem of estimating the number of local optima can be compared with a well-known problem in biology: Estimating the number of different species in a population. In the Statistics field, we can find plenty of algorithms and methods used to estimate the number of species in a population. In fact, [31] noticed the connection between these two problems, and they applied the Schnabel-Census method, already used for estimating species, for estimating the number of local optima. In [11], as well as in [78, 79, 80], an exhaustive classification of methods for estimating the number of species is given. In those works the literature is organized depending on the sampling model, population specification, and philosophy of the estimation. We can also find recent papers [18, 90] that give estimators for the number of classes by assuming that the species abundance follows a Poisson-Gamma model, and others [91, 92] in which estimators are based on the conditional likelihood of a Poisson mixture model. Unfortunately, there are extreme difficulties associated with estimating the population size [59].

In this work, we present an evaluation of methods for estimating the number of local optima, that not only use the methods proposed in the combinatorial optimization field, but also those developed for the species problem in the statistics arena. After describing them in detail, we test their performance under three different scenarios:

1. Simulated instances of combinatorial optimization problems.
2. Random instances of the Traveling Salesman Problem.
3. Instances of the Traveling Salesman Problem with real distances between cities, and instances of the Flowshop Scheduling Problem obtained from the well-known Taillard's benchmark.

The rest of the chapter is organized as follows. The preliminary mathematical background is given in Section 3.2. In Section 3.3 we explain in detail the selected estimate methods. Section 3.4 shows the experimental results when applying the methods to synthetic data as well as to real instances, and discusses the results observed for the different methods, providing clues to help select the most suitable estimation algorithm for a given instance. Finally, the conclusions are presented in Section 3.5.

3.2 Statistics

The problem we are considering in this chapter, that is, estimating the number of local optima, can be considered as a classical estimation problem in statistics. Therefore, the most common way of solving it is by constructing an estimator by collecting a set of statistics from a sample. Due to the fact that we want to estimate the number of local optima, our sample has to be related to the local optima. Basically, most of the presented methods start by taking a uniformly distributed random sample $S = \{\pi_1, \pi_2, ..., \pi_M\} \subseteq \Omega$ of size M. The local search algorithm is applied to each solution in S, and from the M local optima obtained we get the r $(\leq M)$ different ones into the set $S^* = \{\pi_1^*, ..., \pi_r^*\}$.

Two important statistics that will be used for the estimation methods are α_i and β_i. We denote by α_i $(i \in \{1, 2, ..., r\})$ the number of initial solutions of the sample that belong to the attraction basin of the local optimum π_i^*:

$$\alpha_i = \big|\{\pi \in S \mid \mathcal{H}(\pi) = \pi_i^*\}\big| = \big|\{\pi \in S \mid \pi \in \mathcal{B}(\pi_i^*)\}\big| \leq \big|\mathcal{B}(\pi_i^*)\big|.$$

With this information, the following statistic is calculated:

$$\beta_j = \big|\{\alpha_i \mid \alpha_i = j, i \in \{1, 2, ..., r\}\}\big| \ , \quad \forall j \geq 1.$$

So, β_j is the number of local optima that have been seen exactly j times in the sample. In the following section, we call β_0 the number of local optima in the search space that have not been found in the sample. Notice that $\beta_j = 0$, $\forall j > M$. Two interesting relations are the following:

$$\sum_{j=1}^{M} \beta_j = r \qquad \text{and} \qquad \textstyle\sum_{j=1}^{M} j * \beta_j = M.$$

Table 3.1. A classification of the estimate methods selected on both the combinatorial optimization field and the statistics area, according to the sampling model. The methods analyzed in this chapter are in bold and their abbreviations are indicated between brackets.

	SAMPLING MODEL	METHOD (ABBREVIATION)		REFERENCE
COMBINATORIAL OPTIMIZATION	Multinomial	Method based on the Birthday problem		[14]
		Confidence Intervals	**First Repetition Time (*FRT*)**	[31]
			Maximal First Repetition Time (*MFRT*)	
			Schnabel-Census (*Sch-Cen*)	
		Bias Correction	**Jackknife (*Jckk*)**	[30]
			Bootstrap (*Boots*)	
	Gamma	Method based on a Gamma model		[39]
STATISTICS	Multinomial	**Chao 1984 (*Chao1984*)**		[17]
		Chao & Lee 1 (*ChaoLee1*)		[19]
		Chao & Lee 2 (*ChaoLee2*)		
	Poisson - Gamma	**Chao & Bunge (*ChaoBunge*)**		[18]
		Poisson-Compound Gamma Model		[90]
	Mixed Poisson	Penalized Nonparametric Maximum Likelihood Approach		[92]

3.3 Estimation methods

In this section we present a review of the methods proposed in the literature for estimating the number of local optima, as well as the most widespread methods for estimating the number of species in a population. Table 3.1 shows the methods collected from the combinatorial optimization and also from the statistics area. Inside the combinatorial optimization field we find the following methods: a method based on the Birthday problem [14], First Repetition Time, Maximal First Repetition Time and Schnabel-Census [31], Jackknife and Bootstrap [30], and a method based on Gamma distributions [39]. In the statistics field, the selected methods are: Chao 1984 [17], Chao & Bunge [18], Chao & Lee 1 and Chao & Lee 2 [19], a Poisson-Compound Gamma Model [90] and a Penalized Nonparametric Maximum Likelihood Approach [92].

We discarded some of the methods shown in Table 3.1, due to their poor performance [14], high dependence with the sample size [39], or the high computation time [90, 91, 92] observed in preliminary experiments. The methods analyzed in this chapter are highlighted in bold.

The five methods proposed in the field of optimization are explained in detail. First Repetition Time (*FRT*), Maximal First Repetition Time (*MFRT*) and Schnabel-Census are methods that provide lower bounds and that can be used for computing confidence intervals, while Jackknife and Bootstrap are bias correcting non-parametric methods. *FRT* has attracted our interest because it is a parameter-less method, whereas *MFRT* and *Sch-Cen* only depend on one parameter which is the sample size. So, these three methods do not require too much computation time. Jackknife is a method that also depends on the sample size and it is very fast. Bootstrap not only needs the sample size, but also another parameter: the number of repetitions inside the method. The fact of carrying out repetitions causes the method to take more time than the other methods in providing the estimated value.

In a second step, we present methods proposed in the field of statistics used by biologists and ecologists when determining how many different classes of species are in a population of plants or animals. They are non-parametric methods, but they are based on particular sampling models. *Chao1984*, *ChaoLee1* and *ChaoLee2* are based on multinomial sampling, while *ChaoBunge* is based on a mixed Poisson sampling model. The main reason for choosing these methods is that, according to preliminary experiments, they do not require too much computation time and, in general, they give very good estimates.

3.3.1 Methods proposed in the field of optimization

3.3.1.1 Methods used for computing confidence intervals

In this section we describe the methods proposed in [74]: First Repetition Time, Maximal First Repetition Time and Schnabel-Census Procedure. These three methods assume that all the attraction basins of the local optima are equal in size. Under this assumption, and supposing a finite multinomial model, they give (with a high probability) lower bounds for the number of local optima. First, we explain the common aspects of these three methods, and then, the particular details of each of them are given.

Let us start by considering that the continuous distribution function $F_v(t)$ of a random variable T that depends on a parameter v is known, and this distribution function is a strictly monotonically decreasing function on this parameter v, that is:

$$\text{if} \quad v_2 > v_1 \quad \text{then} \quad F_{v_2}(t) < F_{v_1}(t)\ , \ \ \forall t.$$

Now, the (1 - ϵ_1 - ϵ_2)*100% confidence interval for the parameter v is calculated. So, the goal is to find [v_1 , v_2], with $v_1 < v_2$, and such that P($v_1 \leq v \leq v_2$) = 1 - ϵ_1 - ϵ_2, from the known distribution function $F_\mu(t) = P(T \leq t \mid v = \mu)$. Taking into account that $\tau \in \mathbb{N}$ is an observed value sampled from this distribution, we obtain:

$$v_1 = min\left\{\mu \mid 1 - F_\mu(\tau - 1) \geq \epsilon_1\right\} \quad , \quad v_2 = max\left\{\mu \mid F_\mu(\tau) \geq \epsilon_2\right\}.$$

Following the framework of [74], we work with the fixed value $\epsilon_1 = 0.05$. So, these methods will define v_1 as a lower bound with probability 0.95 when $\epsilon_2 = 0$, and consequently v_2 is infinity.

Particularly, in *FRT*, *MFRT*, and Schanbel-Census methods, we will denote v as the number of local optima. Let $\mathbf{p} = (p_1, p_2, ..., p_v)$ be the vector of probabilities of finding the corresponding local optima, that is, the relative sizes of the attraction basins of the local optima. If $\mathbf{p} = \bar{\mathbf{p}} = (\frac{1}{v}, \frac{1}{v}, ..., \frac{1}{v})$ then all the local optima have the same probability of being found. These methods calculate the distribution function $F_{\bar{\mathbf{p}}}(t)$ according to a random variable T, which in each case will determine different concepts.

1. **The First Repetition Time method**
This method starts taking uniformly at random a solution π_1 from the search space Ω, and a local search algorithm (in our case, algorithm $\mathcal{H}$) is applied to π_1, ending at a local optimum π_1^*. This process is repeated until a local optimum is seen twice.
The random variable T denotes, in this case, the number of initial solutions π_i taken until a local optimum is repeated. The distribution function of T corresponding to the vector $\mathbf{p}$ is $F_{\mathbf{p}}(t)$. It can be proved [74] that for any $t \geq 2$, $F_{\mathbf{p}}(t)$ is minimal only at $\mathbf{p} = \bar{\mathbf{p}} = (\frac{1}{v}, \frac{1}{v}, ..., \frac{1}{v})$, where v is the number of local optima.
Now, the distribution function $F_{\bar{\mathbf{p}}}(t)$ is calculated for the variable T.
$$F_{\bar{\mathbf{p}}}(t) = P(T \leq t \mid \mathbf{p} = \bar{\mathbf{p}}) = 1 - P(T > t \mid \mathbf{p} = \bar{\mathbf{p}}),$$
where $P(T > t \mid \mathbf{p} = \bar{\mathbf{p}})$ is the probability of finding none of them repeated in the t first optima:
$$P(T > t \mid \mathbf{p} = \bar{\mathbf{p}}) = \frac{v}{v}.\frac{v-1}{v}...\frac{v-t+1}{v} = \left(\frac{1}{v}\right)^t \binom{v}{t} t!.$$

So,

$$F_{\bar{\mathbf{p}}}(t) = 1 - \left(\frac{1}{v}\right)^t \binom{v}{t} t!.$$

Assuming that τ is the value obtained from the sample for the variable T, the estimate for the number of local optima v_1 is given by the following formula:

$$\hat{v}_{FRT} = v_1 = min\left\{\mu \;\middle|\; \left(\frac{1}{\mu}\right)^{\tau-1} \binom{\mu}{\tau-1} (\tau-1)! \geq 0.05\right\}.$$

2. **The Maximal First Repetition Time method**
In this case a uniformly distributed random sample S of size M is taken from the search space: $S = \{\pi_1, \pi_2, ..., \pi_M\} \subseteq \Omega$. Then, a local search is applied to each solution of the sample, so that M local optima $\{\pi_1^*, \pi_2^*, ..., \pi_M^*\}$ are obtained. Notice that not all of them have to be different. Then, starting from π_1^* and taking the local optima in order of appearance, subsequences $S_i \subseteq S$ are created, where each of them ends with its first re-occurrence of a local optimum. It is as if we were repeating the First Repetition Time procedure many times. If there is no local optima repeated, then the unique subsequence obtained is $S_1 = S$ of size M.
The number of subsequences obtained is denoted by s. The variable that denotes the length of the j-th subsequence is T_j, and $T^{(s)} = max_j T_j$ represents the maximum length of all the subsequences.
The distribution function of the variable $T^{(s)}$ corresponding to the vector of probabilities of the local optima $\mathbf{p}$ is

$$F_{\mathbf{p}}^{(s)}(t) = P(T^{(s)} \leq t \mid \mathbf{p}) = P(T_j \leq t, j = 1, ..., s \mid \mathbf{p}).$$

As in the previous case, for a fixed value of t and a fixed value of s, it can be proved [74] that $F_{\mathbf{p}}^{(s)}$ is minimal when $\mathbf{p} = \bar{\mathbf{p}} = (\frac{1}{v}, \frac{1}{v}, ..., \frac{1}{v})$.
The distribution function $F_{\bar{\mathbf{p}}}^{(s)}(t)$ for the variable $T^{(s)}$ is:

$$F_{\bar{\mathbf{p}}}^{(s)}(t) = \prod_{i=1}^{s} P(T_i \leq t \mid \mathbf{p} = \bar{\mathbf{p}}) = \left[1 - \left(\frac{1}{v}\right)^t \binom{v}{t} t!\right]^s.$$

If τ is the value obtained from the sample for the variable $T^{(s)}$, then the estimate for the number of local optima v_1 is given by the following formula:

$$\hat{v}_{MFRT} = v_1 = min\left\{\mu \;\middle|\; 1 - \left[1 - \left(\frac{1}{\mu}\right)^{\tau-1} \binom{\mu}{\tau-1} (\tau-1)!\right]^s \geq 0.05\right\}.$$

3. **Schnabel Census Procedure**
   This method has also been used in ecology to provide an estimate of the size of a population of animals. It consists of taking a sample of size M and counting the number of distinct animals seen. However, we include it in this section because there are already works [30, 74] that have used this method to estimate the number of local optima in COP. The way to proceed in the case of estimating the number of local optima is similar to the problem of estimating the number of animals.
   Firstly, a uniformly distributed random sample $S = \{\pi_1, \pi_2, ..., \pi_M\} \subseteq \Omega$ of size M is taken from the search space Ω, and a local search algorithm is applied to each solution in S, obtaining r different local optima $\{\pi_1^*, \pi_2^*, ..., \pi_r^*\}$, with $r \leq M$.
   Let R be the random variable that represents the number of different local optima found. The distribution function for the variable R, when a sample of size M has been taken from Ω and when it corresponds to the vector of probabilities of the local optima $\mathbf{p}$ is:

$$F_{\mathbf{p}}(r, v, M) = P(R \leq r \mid M, v, \mathbf{p}) = \sum_{i=1}^{r} P(R = i \mid M, v, \mathbf{p}).$$

   If $\mathbf{p} = \bar{\mathbf{p}}$, then $P(R = i \mid M, v) = \frac{v! S(M,i)}{(v-i)! v^M}$, where $S(M, i)$ is the Stirling number of the second kind, that is, the number of all possible partitions of an M-element set into i non-empty subsets.
   Then,

$$F_{\bar{\mathbf{p}}}(r, v, M) = \sum_{i=1}^{r} \frac{v! S(M, i)}{(v - i)! v^M}.$$

   The estimate for the number of local optima v_1 is given by the following formula:

$$\hat{v}_{Sch-Cen} = v_1 = min \left\{ \mu \,\middle|\, 1 - \sum_{i=1}^{r-1} \frac{\mu! S(M, i)}{(\mu - i)! \mu^M} \geq 0.05 \right\}.$$

3.3.1.2 Bias-correcting nonparametric methods

In this section we describe the application of two commonly used bias-correcting methods to the problem of estimating the number of local optima. These methods are Jackknife and Bootstrap, and their specific use in this context was proposed in [30, 31, 71, 72]. While in the original papers only the mechanic of the algorithm is provided, we have also added the assumptions of the methods, as we consider them relevant for our work.

Jackknife and Bootstrap are nonparametric methods based on ideas of resampling. Moreover, they use the concept of bias of an estimator (the difference between the estimated value and the real value) to improve an initial estimate. There is an important difference that the authors make between the application of Jackknife and the application of Bootstrap. In the Jackknife method, no assumptions about the sampling model are made when calculating the initial biased estimate. However, the Bootstrap method has an underlying assumption about the sampling model because it uses the maximum likelihood estimator as an initial biased estimator.

1. **Jackknife**
   The Jackknife method starts from a biased estimator $\hat{v}$, and assumes that the bias decreases asymptotically as the size of the sample increases. The underlying resampling technique consists of leaving the different points π_i of the initial sample out, and finding estimators that reduce the bias of $\hat{v}$. The mean of these estimates is considered the Jackknife estimator.
   This method, in the same way as Schnabel Census Procedure, was previously proposed for the estimation of population sizes [12]. In the context of estimating the number of local optima [30], a uniformly distributed random sample $S = \{\pi_1, ..., \pi_M\}$ of size M is taken from the search space. After applying a local search algorithm to each solution $\pi_i \in S$, the set of local optima $\mathcal{L}^* = \{\pi_1^*, \pi_2^*, ..., \pi_M^*\}$ is obtained, with $r \leq M$ different local optima.
   Next, one point π_i is left out from the sample S. The subset $\mathcal{L}_i^* \subseteq \mathcal{L}^*$ that contains the local optima that correspond to all the solutions in $S - \{\pi_i\}$ is considered: $\mathcal{L}_i^* = \mathcal{L}^* - \mathcal{H}(\pi_i)$. If this idea is repeated leaving each of the points out from the original sample once each time, we obtain M subsets $\mathcal{L}_1^*, \mathcal{L}_2^*, ..., \mathcal{L}_M^* \subseteq \mathcal{L}^*$, with $r_{-1}, r_{-2}, ..., r_{-M} \leq r$ different local optima.
   The biased estimator $\hat{v} = r$ of v is assumed to be of the form $r = v + \frac{a_1}{M} + \frac{a_2}{M^2} + \frac{a_3}{M^3} + ...$ Thus, the bias is of order $\frac{1}{M}$. The purpose is to find an estimator that reduces the bias to $O(\frac{1}{M^2})$. So, for each $i \in \{1, 2, ..., M\}$, the following estimator is defined:

$$r_i = Mr - (M-1)r_{-i} \ , \tag{3.1}$$

   so that $r_i = v + O\left(\frac{1}{M^2}\right)$.
   Since

$$r_{-i} = r - j_i \ , \quad \text{where } j_i = \begin{cases} 0 \qquad if \qquad \exists\, \pi_k \neq \pi_i \in S \ s.t. \ \mathcal{H}(\pi_k) = \pi_i^* \\ 1 \ otherwise \end{cases} \tag{3.2}$$

   then, from (3.1) and (3.2) the estimator is:

$$r_i = r + (M-1)j_i \ , \quad \forall i \in \{1, 2, ..., M\}.$$

   The Jackknife estimator for the number of local optima is the mean value of r_i:

$$\hat{v}_{Jckk} = \frac{1}{M}\sum_{i=1}^{M} r_i = r + \frac{M-1}{M}\beta_1,$$

where β_1 is the number of local optima which have only been seen once. Notice that when the sample size tends to infinity, β_1 tends to 0, and therefore, the estimate for the number of local optima tends to the real number of local optima. So, this is an unbiased estimator.

2. **Bootstrap**
The Bootstrap method starts from a biased estimator $\hat{v}$, obtained from a sample S. Resamples from S are taken and the biased estimate is calculated in each case. With this information, an estimator of the bias is provided. So, the result of adding the estimated bias to the initial $\hat{v}$ is the Bootstrap estimator.
In the application of this method to the estimation of the number of local optima [30], we start from the set $S^* = \{\pi_1^*, \pi_2^*, ..., \pi_r^*\}$ of r different local optima. Assuming a multinomial model [71], with equally sized attraction basins, the probability distribution of the random variable R that represents the number of different local optima found in the sample S is given by

$$P(R=r) = \tfrac{v!}{(v-r)!}\tfrac{S(M,r)}{v^M}, \quad 1 \leq r \leq min\{M, v\}\ ,$$

where $S(M,r)$ is again the Stirling number of the second kind. From this, the maximum likelihood estimate $\hat{v}_r^{ML}$ of v is obtained by solving the equation:

$$M * log\left(1-\frac{1}{v}\right) - log\left(1-\frac{r}{v}\right) = 0.$$

If r/M is small (lower than 0.3) the best estimate for the number of local optima [71] is actually r, because it is assumed that with small values it is likely that all local optima have been found. So, in this case it is considered that $\hat{v}_r^{ML} = r$.
Afterwards, a resample with replacement of the same size M from S is taken, obtaining r_1 different local optima, and the maximum likelihood estimate of r_1 is considered: $\hat{v}_{r_1}^{ML}$. The same procedure is repeated b times, that is, b resamples with replacement from S are taken, obtaining $\{r_1, r_2, ..., r_b\}$ different local optima, and the maximum likelihood estimate for each r_i is calculated: $\{\hat{v}_{r_1}^{ML}, \hat{v}_{r_2}^{ML}, ..., \hat{v}_{r_b}^{ML}\}$.
With the maximum likelihood estimates for the number of local optima of the resamples $\{\hat{v}_{r_1}^{ML}, \hat{v}_{r_2}^{ML}, ..., \hat{v}_{r_b}^{ML}\}$ and the maximum likelihood estimate for the number of local optima obtained in the original sample $\hat{v}_r^{ML}$, the bias can be estimated and used as a bias correction for $\hat{v}_r^{ML}$. The bias can be calculated as the difference between the maximum likelihood estimate of the number of local optima found with the original sample and

the average maximum likelihood estimates of the number of local optima from the resamples: $bias = \hat{v}_r^{ML} - \frac{1}{b}\sum_{i=1}^{b} \hat{v}_{r_i}^{ML}$.
Hence, the Bootstrap estimator for the number of local optima is $\hat{v} = \hat{v}_r^{ML} + bias$, so:

$$\hat{v}_{Boots} = 2\hat{v}_r^{ML} - \frac{1}{b}\sum_{i=1}^{b} \hat{v}_{r_i}^{ML}.$$

A very important observation is that, as the sample size tends to infinity, the number of local optima found tends to be the real number of local optima. In addition, as r/M is very small (M is very large, and r is constant), the maximum likelihood estimate is the number of local optima found from the sample. Moreover, the bias tends to 0, so the estimate tends to be the real number of local optima.

3.3.2 Methods proposed in the field of statistics

In this section we present four nonparametric methods based on sampling models: *Chao1984* [17], *ChaoLee1*, *ChaoLee2* [19] and *ChaoBunge* [18]. Although they were proposed to estimate the number of species in a population, we explain here their specific application to our problem. An important consideration is that they assume an infinite population, while the common COP have a finite search space. However, we treat the search spaces as if they were infinite because of their large cardinality.

All of the methods presented below start from a sample

$$S = \{\pi_1, \pi_2, ..., \pi_M\} \subseteq \Omega$$

of size M. A local search algorithm is applied to each solution $\pi_i \in S$ and r different local optima $\{\pi_1^*, \pi_2^*, ..., \pi_r^*\}$ are obtained.

1. **Chao 1984**
This is a nonparametric method proposed in [17] based on multinomial sampling that has been used to estimate the number of classes in an infinite population.
The estimator given by this method is the result of adding to the number of local optima obtained from the sample a quantity that depends only on the number of local optima seen once and twice in the sample.
This method is based on the estimate of the expected value of the number of unobserved local optima E_{β_0}. [44] proved that if $j^2 = O(M)$, then $E_{\beta_j} \sim \sum_{i=1}^{v} (Mp_i)^j \frac{e^{-Mp_i}}{j!}$, where p_i is the relative size of the attraction basin of the local optimum π_i^*. The method considers the following distribution function:

$$F(\pi) = \frac{\sum\limits_{Mp_i \leq \pi} (Mp_i)e^{-Mp_i}}{\sum\limits_{i=1}^{v} (Mp_i)e^{-Mp_i}}$$

and it assumes that the number of unobserved local optima is of the following form: $E_{\beta_0} \sim \sum\limits_{i=1}^{v} e^{-Mp_i} \sim (E_{\beta_1}) \int_0^M \pi^{-1}\, dF(\pi)$.
We want to obtain an estimator $\hat{F}(\pi)$ of $F(\pi)$ and thus, find an estimator $\hat{v}$ of v that is the sum of r and the estimated number of unobserved local optima. That is,

$$\hat{v} = r + \beta_1 \int_0^M \pi^{-1}\, d\hat{F}(\pi).$$

The moment estimates were proposed in [17] to obtain an estimator $\hat{F}(\pi)$ of $F(\pi)$, and once attained, a lower bound for v was found. As the sample size M tends to infinity, the lower bound tends to the *Chao1984* estimator:

$$\hat{v}_{chao1984} = r + \frac{\beta_1^2}{2\beta_2}.$$

It is very important to take into account that this estimator works when the information is concentrated on the low order occupancy numbers, that is, when β_1 and β_2 carry most of the information. If $\beta_2 = 0$, that is, if there is no local optima seen exactly twice from the sample, the method does not work. Moreover, β_1 is also very important in this estimator, because if $\beta_1 = 0$, the estimate is just the number of local optima obtained from the sample. We find these situations, for example, when M is much higher than the real number of local optima. In this case, it is unlikely that the local optima will be found only once or twice. Furthermore, we can also find $\beta_1 = 0$ and $\beta_2 = 0$ when the attraction basins are close in size, because the different local optima are probably found the same number of times. Notice that only if we force the method to return r as the estimate for the number of local optima when $\beta_1 = 0$ and $\beta_2 = 0$, we can ensure that when M tends to infinity we obtain the real number of local optima.

2. **Chao & Lee**
[19] proposed two new estimators based on the estimators proposed by [32]. The methods are also nonparametric, used for infinite population, and based on multinomial sampling.
They are the first methods that included the idea of sampling coverage. These estimators are the sum of the number of local optima observed many times, and quantities dependent on the number of local optima found few times in the sample.
Given a sample S, the sample coverage C is defined as the sum of the probabilities of the observed local optima. That is, the sum of the relative

sizes of the attraction basins of the local optima found. Obviously, C is a random variable and an estimator $\hat{C}$ of C [42] is $\hat{C} = 1 - \beta_1/M$. Notice that if all the local optima have the same probability of being chosen, that is

$$p_1 = p_2 = ... = p_v = \frac{1}{v},$$

then

$$C = \frac{r}{v}.$$

So, an initial estimator of v is

$$\hat{v}_1 = \frac{r}{\hat{C}}.$$

Based on [32] and using $\hat{v}_1$, Chao and Lee proposed an estimator $\hat{v}$ of v of the following form:

$$\hat{v} = \hat{v}_1 + \frac{M(1-\hat{C})}{\hat{C}}\gamma^2, \qquad (3.3)$$

where $\gamma = \frac{1}{\bar{p}}\left[\sum_{i=1}^{v}(p_i - \bar{p})^2/v\right]^{1/2}$ is the coefficient of variation (with $\bar{p} = \frac{1}{v}\sum_{i=1}^{v} p_i$).

Notice that $\gamma^2 = v\sum_{i=1}^{v} p_i^2 - 1$. Therefore, they suggested:

$$\gamma^2 = \frac{v\sum_{i=1}^{M} i(i-1)\beta_i}{M(M-1)} - 1.$$

Chao and Lee distinguished between what they call abundant species and rare species. Transferring these concepts to our problem, we will distinguish between easy-to-find local optima and hard-to-find local optima. We define a local optimum π_k^* as hard-to-find if $\alpha_k \leq \delta$ for some cut-off value δ. So, the easy-to-find local optima are the π_k^* such that $\alpha_k > \delta$. One can select this cut-off value in advance, but there are studies [20], that are based on empirical experience set $\delta = 10$. We call r_h the number of hard-to-find local optima, and r_e the number of easy-to-find local optima.

So, taking this distinction into account, the estimators are the following:

$$\hat{v}_{\mathit{ChaoLee1}} = r_e + \frac{r_h}{\hat{C}_h} + \frac{\beta_1}{\hat{C}_h}\hat{\gamma_h}_1^2 \quad , \quad \hat{\gamma_h}_1^2 = max\left\{\frac{r_h}{\hat{C}_h}\frac{\sum_{i=1}^{\delta} i(i-1)\beta_i}{M_h(M_h-1)} - 1, 0\right\}$$

$$\hat{v}_{\textit{ChaoLee2}} = r_e + \frac{r_h}{\hat{C}_h} + \frac{\beta_1}{\hat{C}_h} \hat{\gamma_h}_2^2,$$

$$\hat{\gamma_h}_2^2 = max \left\{ \hat{\gamma_h}_1^2 \left[1 + (1 - \hat{C}_h) \frac{\sum\limits_{i=1}^{\delta} i(i-1)\beta_i}{(M_h - 1)\hat{C}_h} \right], 0 \right\}$$

where $M_h = \sum\limits_{j=1}^{\delta} j\beta_j$ and $\hat{C}_h = 1 - \beta_1 / M_h$.

It is important to take into account that if $\hat{C}_h = 0$, the method does not work. This occurs when $\beta_1 \neq 0$ and $\beta_i = 0, \forall i \in \{2, ..., \delta\}$. Neither does the method work when $M_h = 0$ or $M_h = 1$, that is, when $\beta_i = 0, \forall i \in \{1, ..., \delta\}$, or when $\exists i \in \{1, ..., \delta\}$ such that $\beta_i = 1$ and $\beta_j = 0, \forall j \neq i$. In these cases we redefine the estimators as the number of the different local optima that appear in the sample r. Only under this consideration, we ensure having an unbiased estimator and therefore, obtaining the real number of local optima when M tends to infinity.

3. **Chao & Bunge**
A new estimate technique was proposed by [18]. This estimator is also nonparametric in form, but it has some optimality properties under a particular parametric model. This method is based on a mixed Poisson sampling model. It is closely related to the estimators in [19].
This method bases the estimate of unobserved local optima on

$$(\beta_1, \beta_2, ..., \beta_\delta)$$

for some cut-off value δ, and then they complete the estimate by adding the number of local optima found more that δ times in the sample.
Firstly, they showed that, for the Gamma-Poisson case, the expected proportion of duplicates in the sample, denoted by θ, can be estimated by $\hat{\theta} = 1 - \frac{\beta_1 \sum\limits_{k=1}^{M} k^2 \beta_k}{\left(\sum\limits_{k=1}^{M} k\beta_k \right)^2}$. Secondly, and based on this estimator, they proposed the following estimator for the expected number of unseen optima: $\hat{\beta}_0 = (\hat{\theta}^{-1} - 1) \sum\limits_{k=2}^{M} \beta_k - \beta_1$. Thirdly, with this estimator of the unseen optima, the estimator of the number of local optima was defined as $\hat{v}^* = (\hat{\theta}^{-1} - 1) \sum\limits_{k=2}^{M} \beta_k - \beta_1 + r = \sum\limits_{k=2}^{M} \frac{\beta_k}{\hat{\theta}}$.
Finally, based on the data $(\beta_1, \beta_2, ..., \beta_\delta)$ and the estimator $\hat{v}^*$, and taking into account the distinction between hard-to-find local optima (r_h) and easy-to-find local optima (r_e), their final proposed estimator of v is:

$$\hat{v}_{ChaoBunge} = r_e + \sum_{k=2}^{\delta} \frac{\beta_k}{\hat{\theta}} \quad , \quad \hat{\theta} = 1 - \frac{\beta_1 \sum\limits_{k=1}^{\delta} k^2 \beta_k}{\left(\sum\limits_{k=1}^{\delta} k \beta_k \right)^2} \tag{3.4}$$

where $\hat{\theta}$ takes only into account the hard-to-find local optima, and not all the local optima found in the sample.
Notice that, if $\delta > max\{j \mid \beta_j \neq 0\}$, then all the local optima are considered as hard-to-find, and therefore

$$\hat{v}_{ChaoBunge} = \sum_{k=2}^{M} \frac{\beta_k}{\hat{\theta}} \, , \hat{\theta} = 1 - \frac{\beta_1}{M^2} \sum_{k=1}^{M} k^2 \beta_k .$$

It is important to take into account that if we have a sample in which $\beta_1 \neq 0$ and $\beta_i = 0, \forall i \in \{2, ..., \delta\}$, then $\hat{\theta} = 0$ and therefore the method does not work. In these cases, we consider that $\hat{\theta} = 1$ and so, the estimate is just r. Considering this point we have that, as M tends to infinity, we obtain the real number of local optima.

3.4 Experiments

The accuracy of the different estimators presented in the previous section has been tested on three different datasets: simulated instances of COPs, instances of the Traveling Salesman Problem (TSP) taking random distances, and instances of the TSP with real distances between different cities, as well as instances of the Flow Shop Scheduling Problem (PFSP) obtained from the well-known Taillard's benchmark. Using these datasets we can first test the methods over a wide set of instances with different characteristics (the dataset with simulated instances) and then check whether these conclusions can be generalized for artificial and real instances (second and third datasets). In these three scenarios, we work with problems for which we already know the number of local optima, which allows us to evaluate the accuracy of the different estimates. We report a comparison of the different methods, giving recommendations of the methods that provide the best estimates.

3.4.1 Synthetic data

3.4.1.1 Experimental design

The aim of this section is to study the performance of the methods when they are applied to instances with different number of local optima and different distributions of the sizes of the attraction basins. Therefore, we are interested in a set of data that includes instances with attraction basins very similar in size, as well as instances with very different sizes of attraction basins. However, it is not easy to obtain many real or random instances with the desired characteristics. On the one hand, looking for the real number of local optima and the sizes of their attraction basins of a given instance of a COP would require high computation time. On the other hand, it is not easy to find in the literature instances with a high number of local optima that are realistic enough. These are the reasons why we decide to simulate instances of COP, instead of working with real ones.

As far as the methods for estimating the number of local optima are concerned, an instance of a COP is determined by the number of local optima and the size of their attraction basins. Therefore, we can summarize an instance as the pair $(v, \mathbf{p})$, where v denotes the number of local optima and $\mathbf{p} = (p_1, p_2, ..., p_v)$, with $0 < p_i < 1\ ,\ \forall i \in \{1, 2, ..., v\}$, $\sum\limits_{i=1}^{v} p_i = 1$, is the vector that gives the relative sizes of the attraction basins of the local optima. So, we create instances just assuming a certain number of local optima and assigning to each local optimum a probability of being chosen (or a certain relative size of its attraction basin).

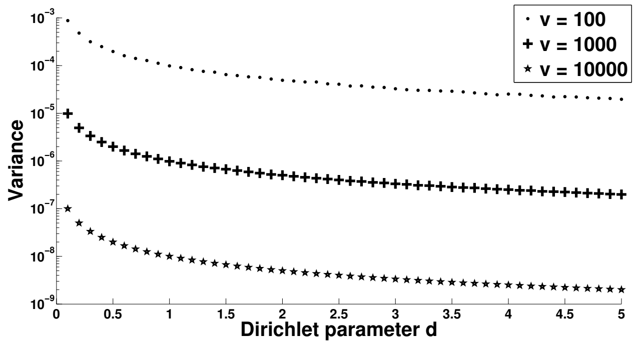


Fig. 3.1. Average variance of the values of the samples obtained from $D(v, d)$, for the different values of d, and for $v = 100, 1000$ and 10000. The Y axis is at logarithmic scale.

One way to do that is by sampling a Dirichlet distribution. When a Dirichlet distribution with v parameters $D(d_1, d_2, ..., d_v)$ is sampled, a vector with v values $(r_1, r_2, ..., r_v)$ is obtained that fulfills the following two relations:

$$0 < r_i < 1\ ,\ \forall i \in \{1, 2, ..., v\} \quad \text{and} \quad \sum_{i=1}^{v} r_i = 1.$$

So, we take advantage of this property of the Dirichlet distribution to simulate the relative sizes of the attraction basins of v local optima. We simplify the sampling by assuming $d_1 = d_2 = ... = d_v = d$, so that we sample Dirichlet distributions with only two parameters: $D(v, d)$.

For the parameter v we decide to work with the following values: 100, 1000 and 10000. Regarding d, working with many values of this parameter would be unfeasible. We decide to make an initial experiment to choose values of d that could simulate very different distributions of the sizes of the attraction basins. In order to choose these values, we sample different $D(v, d)$, for $v = 100, 1000$ and 10000, and $d = 0.1, 0.2, 0.3, ..., 4.8, 4.9, 5$. For each v and d we take 100 samples and calculate the variance of the v data obtained in each sample. Then, according to the average of the 100 variances of each case, we will choose the values of d that we will use to simulate the instances. Figure 3.1 shows the average values of the 100 variances obtained for each combination of d and v.

Observing the plot, we choose the following values for d: 0.1, 0.2, 0.5 (high-medium variance) and 2, 4 (low variance). Once the values of d are decided, we obtain 10000 samples of a Dirichlet distribution for each combination of d and v, that pretend to be 10000 different instances of COPs for each case. So, we have a set of 150000 instances divided in 15 equally sized groups according to v and d. Each method is run 100 times for each of the simulated instances using two different sample sizes: $M = 1000$ and $M = 10000$. The results provided are the average of the 100 values. Note that as *FRT* does not depend on the sample size, it is only applied 100 times for each instance.

3.4.1.2 Results

In this section we analyze the performance of the methods and compare them taking into account the different parameters (v, d and M). Firstly, we check if the methods provide useful estimates. Secondly, we use nonparametric tests to rank the methods and study the significant differences among the observed results. Finally, a more qualitative study is developed, emphasizing an important characteristic which is the stability of the methods.

The closeness of the estimates provided by the methods to the value we want to estimate is the most important factor. Obviously, there are methods that estimate better than others, but it does not mean that the estimates provided by the best methods are close enough to the real value. In order to check if we are able to obtain good estimates with these methods, we choose for each combination of v, d, and M, the method that provides the best average estimation over all the 1000000 results (10000 instances x 100 repetitions). In Figure 3.2 we represent the average estimate obtained with this best method and the real number of local optima (with a dashed line) for each v and d. As expected, the quality of the estimate depends on the parameters of the instances (v and d) and the sample size. For small values of d (0.1, 0.2 and 0.5, i.e. high variance of the sizes of the attraction basins) the estimates are really far from the real number. Furthermore, the higher the number of local optima, the worse the estimate (see Figure 3.2 (a)-(c)-(e) and (b)-(d)-(f)). For

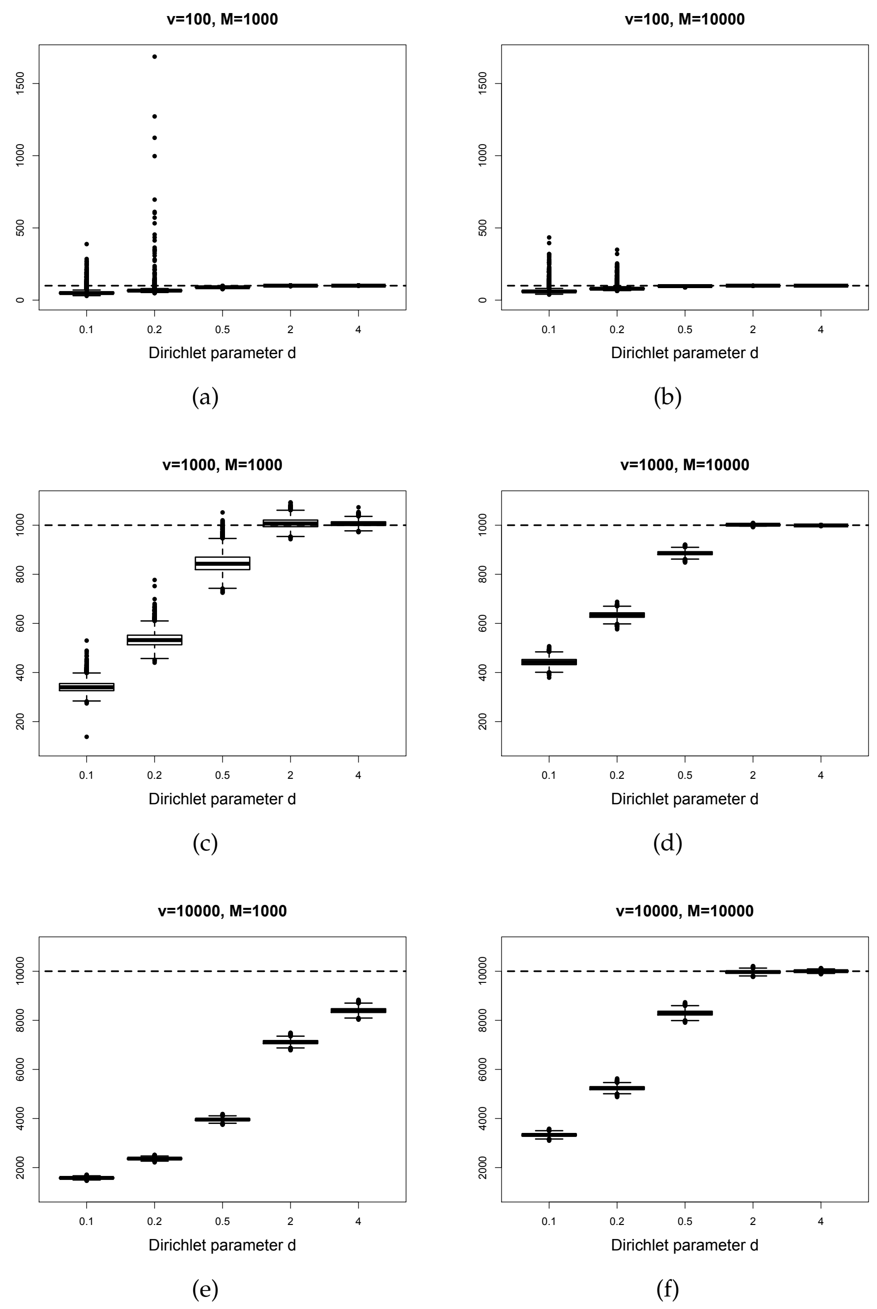


Fig. 3.2. Boxplot that represents the estimations provided by the best method (on average over the 10000 instances x 100 repetitions) for the different values of d. The datasets are created assuming 100 ((a), (b)), 1000 ((c), (d)), and 10000 ((e), (f)) local optima, which are indicated in each figure with a dashed line. In (a), (c) and (e) the methods are applied with sample size 1000, while in (b), (d) and (f) the sample size is 10000.

scenarios where the sizes of the attraction basins are quite similar ($d = 2, 4$), the methods provide precise estimates. As regards the sample size, it can be observed that the larger the sample size, the better the estimates provided are (see Figure 3.2 (a)-(b), (c)-(d) and (e)-(f)). However, this improvement is not enough to reach accurate results in cases of low values of d.

Continuing with the study of the methods, we carry out a statistical analysis to compare the estimates obtained for the different methods. We consider three different scenarios for comparison according to the parameters of the study (M, v, d). In the first scenario considered, the estimates are grouped in two sets according to $M = 1000$ and 10000. The second scenario considers three different sets that contain the estimates of the instances with $v = 100$, 1000 and 10000 local optima. In the last scenario the estimates are grouped in five sets, according to the parameter $d = 0.1, 0.2, 0.5, 2, 4$. A nonparametric Friedman's test with level of significance $\alpha = 0.05$ is used to test if there are statistical significant differences between the estimates provided by the 9 methods in the different scenarios. It provides a ranking of the methods while also giving an average rank value for each method. As we always find statistical differences in all the cases, we proceed with a post-hoc test which carries out all pairwise comparisons. Particularly, we use the Holm's procedure fixing the level of significance to $\alpha = 0.05$. Pairwise significant differences are found between all of the methods in the three scenarios.

Table 3.2 shows the ranking obtained for the methods with the Friedman's test in the first scenario, when using a sample of size $M = 1000$ (first pair of columns) and $M = 10000$ (last pair of columns). The lower the rank, the worse the performance of the method is. So, the methods are ordered from best to worst. Therefore, the best methods when separating the estimates according to the sample size, are *ChaoLee2* and *ChaoBunge*.

In Table 3.3 the ranking for the methods is shown, but this time for the second scenario, that is, when grouping the estimates for the instances created with $v = 100$ (the first pair of columns), $v = 1000$ (the pair of columns in the middle), and $v = 10000$ (the last pair of columns). In these three cases the Holm's procedure states that significant differences exist among each pair of methods. From this table we can observe that the lower the number of local optima, the better the estimates provided by *Chao1984* are. On the contrary, *ChaoLee2* improves its performance as the number of local optima grows.

Finally, Table 3.4 shows the ranking obtained for the methods when the estimates are separated according to the instances created with the same Dirichlet parameter d. This ranking gives an idea of the method that is better to use according to the variance of the sizes of the attraction basins of the local optima. For high variance (small values of d) the recommended method is *ChaoBunge*, but *ChaoLee2* also provides very good estimates. For instances with quite similar sizes of attraction basins the best method is *ChaoLee2*.

From the statistical analysis, we can conclude that the worst methods in all scenarios are *FRT*, *MFRT* and *Sch-Cen*. On the other hand, we can not conclude that there is a best overall method for all scenarios. However, if

Table 3.2. Average rankings of the methods according to the sample size M

M=1000		M=10000	
Method	Ranking	Method	Ranking
ChaoLee2	7.79	*ChaoBunge*	7.78
ChaoBunge	7.03	*ChaoLee2*	6.88
Chao1984	6.87	*Chao1984*	6.68
ChaoLee1	6.50	*Jckk*	6.31
Jckk	5.67	*ChaoLee1*	5.71
Boots	4.59	*Boots*	4.71
Sch-Cen	3.27	*Sch-Cen*	3.63
MFRT	1.94	*MFRT*	1.88
FRT	1.35	*FRT*	1.41

Table 3.3. Average rankings of the methods according to the number of local optima v

v=100		v=1000		v=10000	
Method	Ranking	Method	Ranking	Method	Ranking
Chao1984	7.25	*ChaoBunge*	8.25	*ChaoLee2*	8.33
ChaoBunge	7.05	*ChaoLee2*	7.30	*ChaoLee1*	7.06
ChaoLee2	6.37	*Chao1984*	6.85	*ChaoBunge*	6.91
Jckk	6.35	*Jckk*	6.03	*Chao1984*	6.24
ChaoLee1	5.40	*ChaoLee1*	5.87	*Jckk*	5.58
Boots	5.21	*Boots*	4.51	*Boots*	4.23
Sch-Cen	3.93	*Sch-Cen*	3.20	*Sch-Cen*	3.22
MFRT	2.22	*MFRT*	2.00	*FRT*	1.92
FRT	1.22	*FRT*	1.00	*MFRT*	1.52

we consider the first three best methods, *ChaoBunge* and *ChaoLee2* are always among them. So, in case of lack of information about the number of local optima of the instance, or the sizes of their attraction basins, using both of them we will probably be obtaining more accurate estimates than by using any other method.

Although the statistical analysis gives a global picture of the performance of the methods, it is also relevant to consider some aspects that are not reflected in the hypothesis tests. One of these aspects is that of stability. Imagine that we have some instances with similar properties (the same number of local optima and similar distribution of sizes of the attraction basins). We are interested in knowing if the method will provide comparable estimates for the different instances, or if they will be extremely different. In the first case we say that the method is stable, while in the second it is unstable. *ChaoBunge* is a very unstable method in certain situations, while the rest of the methods are very stable. Under some circumstances, *ChaoBunge* provides a very good

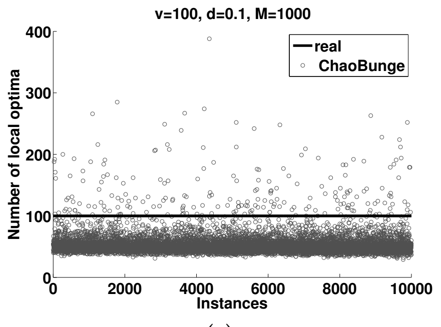
Table 3.4. Average rankings of the methods according to the Dirichlet parameter d

d=0.1		d=0.2		d=0.5	
Method	Ranking	Method	Ranking	Method	Ranking
ChaoBunge	8.37	*ChaoBunge*	8.02	*ChaoBunge*	7.39
ChaoLee2	7.52	*ChaoLee2*	7.31	*ChaoLee2*	7.13
Chao1984	7.07	*Chao1984*	7.01	*Chao1984*	7.08
Jckk	6.30	*Jckk*	6.58	*Jckk*	7.01
ChaoLee1	5.51	*ChaoLee1*	5.82	*ChaoLee1*	6.05
Boots	4.06	*Boots*	4.07	*Boots*	4.13
Sch-Cen	3.06	*Sch-Cen*	3.07	*Sch-Cen*	3.07
MFRT	2.05	*MFRT*	1.90	*MFRT*	1.73
FRT	1.05	*FRT*	1.23	*FRT*	1.40

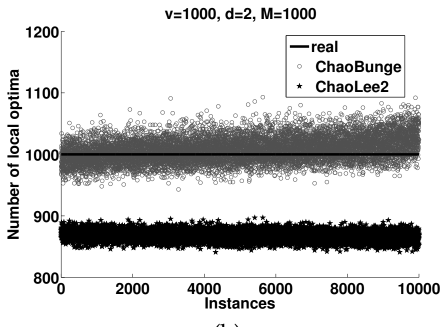
d=2		d=4	
Method	Ranking	Method	Ranking
ChaoLee2	7.17	*ChaoLee2*	7.53
Chao1984	6.75	*ChaoLee1*	6.84
ChaoBunge	6.59	*ChaoBunge*	6.63
ChaoLee1	6.31	*Chao1984*	5.99
Boots	6.03	*Boots*	4.96
Jckk	5.16	*Jckk*	4.90
Sch-Cen	3.72	*Sch-Cen*	4.33
MFRT	1.80	*MFRT*	2.07
FRT	1.47	*FRT*	1.74

estimate for the number of local optima for most of the instances, but there are instances where the given estimate is very far from v. For example, for $v = 100$, $M = 1000$ and $d = 0.1$ (Figure 3.3 (a)). In other situations, for example, for $v = 1000$, $M = 1000$ and $d = 2$ (Figure 3.3 (b)) it is the method that provides the best estimates of the number of local optima for all instances. In order to compare the performance of this method with *ChaoLee2* in these particular scenarios, the estimates provided by *ChaoLee2* are also reflected in Figure 3.3 (b). We provide all the figures that represent the estimates given by each method for each v and each d, in the Appendix 9.1.1.

We conclude from the tables that *ChaoBunge* is one of the best methods, but we find specific situations where the estimate provided is very far from the real value we want to estimate. In order to know if the estimate provided by *ChaoBunge* is valid, we could also apply other methods, such us *ChaoLee2*, and compare both results. If these estimates are close enough, the one provided by *ChaoBunge* could be accepted. Otherwise, if these estimates are very far one from each other, we are almost sure that *ChaoBunge* is giving a useless estimate. So, we consider that a suitable way for estimating the number of local optima of an instance is not by using a single method, but a comparison of different methods.



(a)



(b)

Fig. 3.3. Estimates of the number of local optima provided by *ChaoBunge* (a) and *ChaoBunge* and *ChaoLee2* (b) for 10000 synthetic instances created by sampling $D(100, 0.1)$ and $D(1000, 2)$, respectively. The sample size used in both cases is 1000. In (a) we see the instability of the *ChaoBunge* method, but in (b) it is stable.

3.4.2 Random instances of TSP

3.4.2.1 Experimental design

In order to contrast our initial conclusions, in this section we work with random instances of the Traveling Salesman Problem. Particularly, we work with random instances of the Symmetric Traveling Salesman Problem with 14 and 15 cities. The instances were created by placing 14 and 15 points respectively, uniformly at random on a square of area 100 in an Euclidean space [40]. Afterwards, we calculated the matrix that gives the Euclidean distance between every pair of cities. We randomly created 500 instances with 14 cities, and 110 instances with 15 cities.

For the purpose of measuring the accuracy of the estimation methods we first calculated the exact number of local optima of the instances when using the swap neighborhood (N_S). So, applying to each solution of the search space a deterministic local search algorithm (see Algorithm 1 in Section **??**) with a swap neighborhood, the exact number of local optima of the instance and their corresponding sizes of the attraction basins are obtained. Notice that in the symmetric TSP there are $2n$ permutations encoding the same solution. Therefore, we only take into account one of all these different representations, and thus we search in a space of size $(n-1)!/2$.

The different methods for estimating the number of local optima were applied to all the instances considering two sample sizes: $M = 1000$ and 10000. For each instance the methods are repeated 100 times and we evaluate and compare the average estimates of each method.

3.4.2.2 Results

A first step in the analysis of the methods when applying them to random instances of the TSP is the study of the accuracy of the estimates provided by the methods. Secondly, a parameter d is associated to each instance and, as in the previous section, the performance of the methods is studied again according to d, v and M.

In order to check if the methods provide useful estimates, the average errors of the estimates with respect to the real number of local optima are calculated. Table 3.5 shows the average relative errors and the standard deviations (in brackets) grouped by the number of cities and the sample size.

A general conclusion deduced from Table 3.5 is that for $n = 14$ the methods provide better estimates than for $n = 15$. Table 3.5 also confirms the improvement of the estimates as the sample size grows. It is remarkable that for $n = 15$ cities (higher number of local optima than for $n = 14$) and sample size $M = 1000$, the estimates are very far from the real value, and the standard deviations for these estimates are considerably high. Particularly, *ChaoBunge* has a very high standard deviation when the sample size is 1000. This fact confirms the unstable behavior observed in the previous experiments. The instability of this method is a consequence of the variability on the estimation of the parameter $\hat{\theta}$ (see equation (3.4) of Section 3.2). If $\beta_1 >> \beta_i$ $(2 \leq i \leq \delta)$, then $\hat{\theta} \approx 0$ and the estimation $\hat{v}_{\mathit{ChaoBunge}}$ in this case is very large and very far from the real value. This occurs when we have a sample where a lot of local optima are seen only once, but there is a small number of local optima seen twice, three times, etc. These particularities are commonly found when the sample size is small with respect to the number of local optima, or even when the variance of the sizes of the attraction basins of the local optima is high.

To visualize the performance of the methods, in Figure 3.4 we represent the estimates provided. We first arrange the instances according to the number of local optima and take 10 groups of 11 instances. For each group we calculate the average estimate of each method. We represent the five methods that provide the best results: *Jckk*, *Boots*, *Chao1984*, *ChaoLee1* and *ChaoLee2*. *ChaoBunge* is removed from the plots because of its instability. Figure 3.4 shows the average estimates obtained for the instances of the TSP with 15 cities, for sample size 1000 (up) and 10000 (bottom). We observe from the graphs that, when the number of local optima is lower than $400 - 500$, the estimates are close to the real values. However, as the number of local optima grows, the estimates provided by all the methods tend to distance themselves from the real number of local optima. For sample size 1000, it can be clearly seen that in all groups the best method is *ChaoLee2*, while *ChaoLee1*, *Chao1984* and *Jckk* provide similar estimates. For sample size 10000, we observe that *Jckk* is the best method. Additional figures representing the estimations provided by each method for each instance, can be found in the Appendix 9.1.2.

Table 3.5. Average relative errors and standard deviations (in brackets) of the estimates provided by the different methods, according to the number of cities n and the sample size M. The range for the real number of local optima of the instances appears in brackets under the number of cities n.

		FRT	*MFRT*	*Sch-Cen*	*Jckk*	*Boots*	*Chao1984*	*ChaoBunge*	*ChaoLee1*	*ChaoLee2*
n = 14 **($34 \leq v \leq 648$)**	**M=1000**	96.15 (3.48)	89.20 (17.76)	43.61 (76.89)	27.95 (71.81)	36.75 (74.53)	26.08 (57.02)	27.15 (5949.52)	28.42 (49.49)	**22.70** **(45.48)**
	M=10000		87.15 (23.55)	14.29 (29.03)	**5.65** **(14.61)**	10.28 (21.50)	6.71 (14.77)	7.34 (15.49)	8.93 (15.99)	7.86 (13.83)
n = 15 **($97 \leq v \leq 1087$)**	**M=1000**	97.46 (2.11)	93.04 (11.48)	59.38 (66.19)	44.42 (83.04)	52.87 (73.92)	41.42 (62.66)	59.46 (19247.69)	41.72 (61.08)	**33.78** **(61.47)**
	M=10000		91.68 (15.19)	26.47 (47.48)	14.64 (29.51)	20.91 (39.67)	16.33 (24.74)	**14.64** **(14.20)**	18.16 (24.32)	16.20 (18.40)

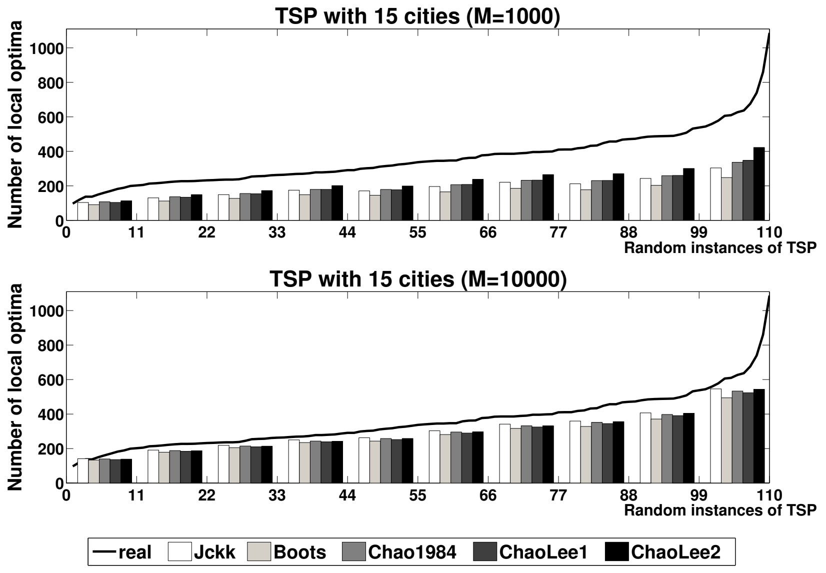


Fig. 3.4. Estimates of the number of local optima provided by the different methods for 110 random instances of the TSP with 15 cities. The 110 instances are arranged according to the number of local optima, and are put in 10 groups of 11 instances each. The average of the 11 estimates is shown by the histogram for each method. The methods consider a sample of size 1000 (top graph) and 10000 (bottom graph). The solid line indicates the number of local optima of the instances.

We compare the methods according to different parameters, as proceeded in the previous section. Firstly, a parameter d is associated to each instance, supposing that the sizes of the attraction basins were created sampling a Dirichlet distribution with that particular d. Due to the fact that we know the number of local optima v of the 610 random instances of the TSP, for each value of v we sample Dirichlet distributions $D(v, d)$, for each $d = 0.1, 0.2, 0.5, 2, 4$. We take 100 samples for each v and d and the variance of the v data is calculated in each sample. On the other hand, the variance of the relative sizes of the attraction basins of the local optima of each of the random instances is calculated. For each instance, we compare the variance of its relative sizes of the attraction basins with the variances obtained when sampling $D(v, d)$, being v the corresponding number of local optima of that instance. We associate to each instance the value of d for which the variance of the instance is closer to the average variance of $D(v, d)$.

A classification of the instances according to d and v is carried out and we realize that most of the instances have low values of d, that is, they have high variances of the sizes of the attraction basins of the local optima. Table 3.6 shows the number of instances that we associate with the different values of d depending on the different number of local optima. Next, we study the performance of the methods taking into account d, v and M, and we compare it with the results of the previous section. As we have only found one instance with parameter $d = 4$, we analyze it separately. We saw in the previous section that the three methods that provided the best estimates for $d = 0.1, 0.2, 0.5, 2$ as well as for $M = 1000, 10000$, were *Chao1984*, *ChaoLee2* and *ChaoBunge*. Table 3.7 shows the percentage of instances for which the three best estimates are provided by these three methods, according to d and M. We observe that for $M = 1000$ the percentages are lower than for $M = 10000$ and this is because when $M = 1000$ *ChaoBunge* is more unstable than for $M = 10000$. For small values of d ($d = 0.1, 0.2, 0.5$), and when the sample size used by the methods is 1000, the best method is *ChaoBunge* in more than 77% of the estimates. In 379 of the 610 instances the second best method is *ChaoLee2*, and the third best method is *Chao1984* in 367 of the 610 instances. These results corroborate the conclusions obtained from the analysis of the methods for the synthetic instances (Table 3.4). When sample size is 10000, for small values of d, *Chao1984* provides very good estimates, and in most of the instances it is the best method. The reason is that almost all of the instances that have been associated a low value of d have also a low number of local optima and, as was seen in Table 3.3, the best method in these scenarios is *Chao1984*.

We only find one instance with a high value of d, and furthermore, it is the instance that has the highest number of local optima ($v = 1087$). When the methods are applied to this instance with sample size 1000, the best estimates are provided by *ChaoLee2* and *ChaoLee1*. As we saw in Table 3.4, these are the best methods for d=4. On the other hand, when sample size is 10000, the best methods are *ChaoBunge* and *ChaoLee2*. This matches the results shown in Table 3.2 (last pair of columns) and Table 3.3 for $v = 1000$.

Table 3.6. Number of instances that are assigned different values of d according to v.

	$30 < v < 500$	$500 < v < 1100$
d=0.1	346	0
d=0.2	155	0
d=0.5	81	3
d=2	13	11
d=4	0	1

Table 3.7. Percentages of the number of instances for which the best three estimates obtained are provided by *Chao1984*, *ChaoBunge* and *ChaoLee2*.

	$M = 1000$	$M = 10000$
d=0.1	80.64%	97.40%
d=0.2	78.71%	99.35%
d=0.5	55.95%	100.00%
d=2	41.67%	95.83%

3.4.3 Instances of TSP and PFSP

3.4.3.1 Experimental design

This section is devoted to experiments with real instances of COPs, as well as instances taken from the Taillard's benchmark. We work with 10 instances of the Traveling Salesman Problem (with real distances between cities) and other 10 instances of the Permutation Flowshop Scheduling Problem.

For the TSP we take the real distances between 14 cities of the continents Africa, America, Asia and Europe, and 14 cities of The United States, Spain and Australia and Pacific cities [1]. For the PFSP we consider instances with 13 jobs and 5 machines, obtained from the well-known benchmark proposed by Taillard [2] that has been commonly used by numerous authors, such as

[1] http://www.mapcrow.info
http://locuraviajes.com/blog/wp-content/uploads/2011/08/cuadro-distancias-ciudades-espaa.gif

[2] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir /ordonnancement.html

[9, 15, 84]. The instances of both TSP and PFSP used in this section of the experiments are available in the website [3].

We apply Algorithm 1 to each instance starting from each solution of the search space. Notice that the size of the search space of the TSP instances is $13!/2$, while the instances of the PFSP have a search space of size $13!$. In this section we consider two neighborhoods: swap and insert.

The different methods for estimating the number of local optima are applied to all the instances using both neighborhoods. The reason why we have considered these two neighborhoods in this section is that they provoke different situations for the estimates obtained with the different methods. As the insert neighborhood explores at each step more solutions than the swap neighborhood, the number of local optima obtained when considering the first neighborhood is probabilistically lower than when assuming the second one.

3.4.3.2 Results

In this final section of this chapter, our aim is to extend the previous analysis focusing on the accuracy of the methods and their relation to the sample size. For this purpose we first look for the minimum sample size that allows each method to reach estimates closed to the real number of local optima. We consider that a method that needs a smaller sample size to provide good estimates will be more useful. In addition, we also analyze the effect of the sample size on the methods using small as well as large sample sizes, in order to find the methods that provide better estimates in more realistic situations. That is, when the sample size is very small compared with the real number of local optima of the instance.

In order to study the sample sizes needed to obtain good estimates, we choose for each method the minimum sample size for which at least 95 of 100 estimates provided are closer than 95% from the real number of local optima of each instance under each neighborhood. The algorithm used to obtain the minimum sample sizes starts with $M = 100$. It doubles the value of M, until it succeeds or reaches the maximum sample size considered (6553600 = $100\text{x}2^{16}$). In case of success for a given M, a bisection procedure is applied until the difference between the last accepted sample size and the previously discarded one is 100. So, it converges to the minimum sample size wanted. We repeat this process 10 times and show the average values.

Tables 3.8 and 3.9 show the average sample sizes that the methods need when they are applied to the TSP and PFSP instances, respectively. In both tables, each row represents an instance and a neighborhood. The first half of the tables is related to the insert operator and the second half to the swap operator. Inside each group, instances are put in an ascending order according to the number of local optima (first column). In most of the instances the

[3] http://www.sc.ehu.es/ccwbayes/members/leticia/EstimationNumOpt /EstNumOptInst.html

MFRT method is not able to fulfill the condition stated (a line is drawn for these cases). Notice that *FRT* is not taken into account because this method does not depend on the sample size and, as we saw in the previous sections, this method provides such bad estimates that we decided to take it out from the study in this section.

Table 3.8. Average of the minimum sample sizes obtained for which at least 95 of 100 estimates provided by each of the methods are closer than 95% to the real number of local optima for each TSP instance under insert and swap neighborhoods. The minimum sample size obtained for each instance is in bold.

	Number of local optima	*MFRT*	*Sch-Cen*	*Jckk*	*Boots*	*Chao1984*	*ChaoBunge*	*ChaoLee1*	*ChaoLee2*
TSP Insert	1	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	2	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	2	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	5	650	**100**	110	**100**	**100**	**100**	**100**	**100**
	9	——	1320	2030	**1220**	1330	1910	1870	1840
	9	353940	**200**	250	**200**	**200**	230	**200**	**200**
	12	——	1690	2750	**1650**	1750	2520	2600	2590
	22	——	2740	4330	**2720**	3600	4030	3580	4000
	29	——	**2260**	3360	2270	2670	5240	2860	5480
	32	——	2370	3490	**2320**	2960	3110	2750	2950
TSP Swap	67	——	23270	28810	**22010**	37130	34250	24280	28590
	73	——	522490	**483810**	515930	647950	1095310	793700	1009610
	90	——	55220	**44460**	57090	81340	174680	55320	152200
	92	——	24980	30450	**20020**	39110	23770	21740	23250
	103	——	40450	**26450**	32110	48070	35920	36390	36790
	117	——	179670	**132460**	173490	251070	228310	165270	177680
	188	——	88860	**47150**	66520	129460	75440	74640	70690
	201	——	93970	**45850**	68870	107460	69160	70890	65720
	393	——	224390	**91510**	150410	167600	168980	167560	166950
	455	——	275540	**89380**	161850	189310	189850	199250	193950

Looking at the overall results, one could conclude that the best methods are Jackknife and Bootstrap, because in almost all instances they need a smaller sample size to provide very good estimates. This fact seems to be in conflict with almost all the results obtained in the previous sections, where *Chao1984*, *ChaoBunge*, *ChaoLee1* and *ChaoLee2* seemed to be the most promising. However, this result agrees with that observed in the previous section, where *Jckk* provided better estimates than the rest of the methods for sample size 10000. Therefore, and in order to obtain additional information about the performance of the methods, we decided to study the estimates provided by them when the sample sizes are small. This idea arose when we realized that in real life we have to face problems of such high dimensions that they have a

Table 3.9. Average of the minimum sample sizes obtained for which at least 95 of 100 estimates provided by each of the methods are closer than 95% to the real number of local optima for each PFSP instance under insert and swap neighborhoods. The minimum sample size obtained for each instance is in bold.

	Number of local optima	*MFRT*	*Sch-Cen*	*Jckk*	*Boots*	*Chao1984*	*ChaoBunge*	*ChaoLee1*	*ChaoLee2*
PFSP Insert	14	——	3320	5540	**3150**	3640	4720	4830	4760
	70	——	17920	19380	16520	25220	17680	**16390**	18130
	134	——	25710	25730	**18880**	45320	21540	19870	20750
	160	——	47300	**28170**	34350	63310	39850	37040	35380
	190	——	19930	**11630**	13770	32110	16050	15910	15960
	285	——	23820	**9840**	15260	19310	16620	18110	17240
	404	——	29830	**10670**	18430	19880	18790	20100	19270
	461	——	51320	**17550**	31200	35040	34770	36670	33560
	506	——	77700	**24780**	45910	52990	54730	58060	56180
	923	——	137950	**40850**	79530	88750	92460	94730	93780
PFSP Swap	192	——	39410	**19540**	29300	55320	31750	32290	33600
	1643	——	445780	**134260**	254870	264950	264640	285060	264130
	1846	——	628440	**199600**	363320	338380	323990	366380	338730
	1997	——	592030	**177390**	341020	337870	343950	370630	354610
	2130	——	912200	**273490**	527160	508690	521420	576430	539570
	2382	——	763420	**227140**	435560	411080	426840	466190	431230
	2386	——	613130	**179250**	353810	346650	357130	384600	363640
	5119	——	2149000	**643230**	1229450	1098740	1093250	1235370	1128300
	6485	——	1671460	**456690**	927350	863640	875150	994270	900410
	8194	——	2052570	**568480**	1148250	1032760	1058350	1204950	1094970

huge number of local optima. So, the sample we are able to deal with is usually tiny compared to the number of local optima. Thus, we are interested in finding methods that do not need such a large sample size to provide a good estimate.

We apply the methods taking small sample sizes (compared to the number of local optima) and analyze them according to the estimate they provide. We have only considered the instances with more than 100 local optima, without making distinctions between the two neighborhoods. We take sample sizes in the range 50-600 with steps of 50. *MFRT* and *Sch-cen* have been removed from the study because for the instances with the highest number of local optima, the estimates provided by these methods are lower bounds very far from the real values. So we analyze *Jckk, Boots, Chao1984, ChaoBunge, ChaoLee1* and *ChaoLee2*.

The methods are applied 100 times for each sample size. We carry out the nonparametric Friedman's test with level of significance $\alpha = 0.05$ to the estimates, grouping them according to the sample size. We observe that there are statistical significant differences between the estimates provided by the

Table 3.10. Best methods obtained from the Friedman's test for the TSP and PFSP according to the sample size. The average relative error of the estimates provided by these methods is also shown.

	Sample size	Best method	Average relative error
TSP	50 ... 350	*ChaoLee2*	22.56
	400 ... 450	*ChaoLee2*, *ChaoBunge*	18.76, 20.05
	500 ... 600	*ChaoBunge*	11.46
PFSP	50 ... 600	*ChaoLee2*	49.95

six methods for all the sample sizes. We continue with the Holm's procedure which carries out all pairwise comparisons, setting the level of significance to $\alpha = 0.05$. Table 3.10 shows the best methods obtained from the Friedman's test for the TSP and PFSP according to the sample sizes. For the TSP, and using sample sizes lower than 400, we find that the best method is *ChaoLee2* and significant differences between *ChaoLee2* and the rest of the methods are found. For sample sizes 400 and 450, the best methods are *ChaoLee2* and *ChaoBunge*. There are no significant differences between them, but there are between them and the rest of the methods. For sample sizes larger than 450, the best method in the ranking is *ChaoBunge*, with significant differences between this method and the rest. For the PFSP instances, and for all sample sizes, *ChaoLee2* is the best performing method, and when we study the pairwise significant differences with the Holm's procedure we can see that there are significant differences between *ChaoLee2* and the rest of the methods.

Let's now study in detail the estimates obtained for the two instances with the highest number of local optima. Tables 3.11 and 3.12 show the average of 100 estimates provided by each method for small sample sizes (with respect to the number of local optima) for the instances 9 and 10 of PFSP, respectively, when using the swap neighborhood. These tables show that, when sample size is small, *Boots* and *Jckk* provide worse estimates than the other methods. Obviously, the estimates improve as the sample size grows for all methods, except for *ChaoBunge*. As was seen in previous sections, the *ChaoBunge* method is very unstable. The estimate provided by this method varies significantly depending on the sample size. Notice that although *ChaoLee2*, *ChaoLee1* and *Chao1984* provide the best estimates, these are also far from the real number of local optima.

If we analyze all the results obtained in this section, on the one hand, we find that *Jckk* and *Boots* need a smaller sample size than the rest of the methods to provide very good estimates. On the other hand, *ChaoLee2* and *ChaoBunge* are considered the best methods for small sample sizes. So, we suspect that there is a threshold for the sample size where the estimates provided by *ChaoLee2* and *ChaoBunge*, or even *ChaoLee1* and *Chao1984*, are worse when compared with the estimates provided by *Jckk* and *Boots*.

Table 3.11. Average of 100 estimates provided by each method for small sample sizes for the instance 9 of PFSP when using swap neighborhood.

Instance 9 PFSP. Real number of local optima: 6485

Method	M=50	M=100	M=150	M=200	M=250	M=300	M=350	M=400	M=450	M=500	M=550	M=600
Jckk	92	171	241	304	363	418	469	517	563	605	646	687
Boots	64	122	173	220	265	307	344	381	416	449	481	511
Chao1984	508	683	741	756	798	835	879	919	968	1016	1061	1099
ChaoBunge	254	390	508	259	425	607	600	533	1345	53853	519	393
ChaoLee1	576	682	799	847	876	910	950	988	1032	1074	1129	1176
ChaoLee2	789	1009	1221	1293	1290	1292	1341	1378	1441	1493	1584	1664

Table 3.12. Average of 100 estimates provided by each method for small sample sizes for the instance 10 of PFSP when using swap neighborhood.

Instance 10 PFSP. Real number of local optima: 8194

Method	M=50	M=100	M=150	M=200	M=250	M=300	M=350	M=400	M=450	M=500	M=550	M=600
Jckk	95	182	262	335	405	470	532	593	648	702	754	803
Boots	66	129	185	241	292	340	386	431	473	512	551	587
Chao1984	1018	1050	1048	1077	1150	1194	1240	1278	1334	1367	1405	1445
ChaoBunge	341	476	753	1634	5392	1144	5634	1211	1165	1279	4140	540
ChaoLee1	701	1035	1055	1061	1140	1200	1274	1323	1402	1443	1492	1543
ChaoLee2	745	1195	1195	1247	1396	1492	1633	1718	1858	1915	1992	2080

These suspicions motivate us to represent the estimates obtained by the different methods for the two instances with the highest number of local optima as the sample size grows. Here, we just plot the estimates corresponding to *Jckk*, *ChaoLee1* and *ChaoLee2* to see the threshold mentioned more clearly. Nevertheless, more detailed graphs are available in the Appendix 9.1.3. In Figure 3.5 we take into account very small sample sizes as well as high sample sizes (from $M = 50$ to $M = 200000$). We observe that, when the sample size is small, the best methods are *ChaoLee2* and *ChaoLee1*. There is a threshold (for sample size between 20000 and 60000) where *Jckk* improves its estimates compared with those provided by *ChaoLee1*, and for sample size between 80000 and 200000, the estimates given by *Jckk* also improve those provided by *ChaoLee2*. The reason is that with a small growth in the sample size, the estimate provided by *Jckk* improves more than the estimates given by *ChaoLee1* or ChaoLee2. So, our recommendation is to use the *Jckk* and *Boots* methods when we are able to work with large sample sizes. But, if we suspect that our sample is very small compared to the real number of local optima, the best methods to apply are *ChaoLee2* or *ChaoLee1*.

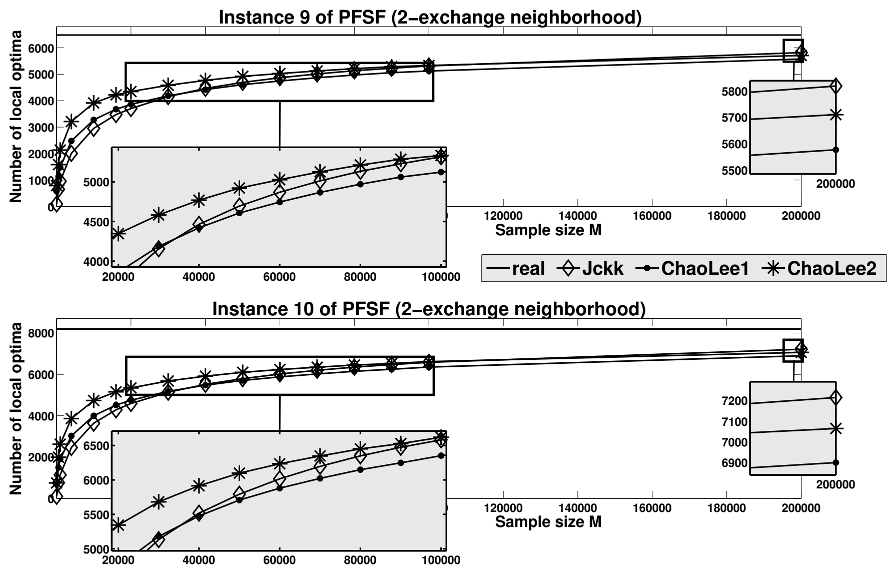


Fig. 3.5. Average of 100 estimations obtained by the different methods for the instance number 9 (up) and instance number 10 (bottom) of PFSP using the swap neighborhood as the sample size grows.

3.5 Conclusions

In this chapter we have reviewed different methods for estimating the number of local optima of instances of COPs. Our main contribution is the comparison of methods in the optimization field with some methods previously used for estimating the number of species in a population in the field of statistics.

The methods have been applied to three datasets: synthetic instances, instances of the TSP with 14 and 15 cities taken at random, and instances of TSP with real distances between cities and instances of PFSP taken from the well-known Taillard's benchmark. The main conclusions observed for all the methods in the three scenarios are the following:

1. When the attraction basins are similar in size, the methods provide estimates close to the real number of local optima. Of course, the higher the sample, the more precise the estimates.
2. The further the sizes of the attraction basins from the uniformity, the worse the estimates. In fact, in the real instances (where the variance of the sizes of the attraction basins is very high) the predictions are really far from the real number of local optima.

Based on the results observed through the experiments, we provide the following rules of thumb:

- If we are able to take a sample of large size with respect to the number of local optima, we recommend using *Jckk*.
- If we suspect that our sample size is small (with respect to the number of local optima), we recommend using *ChaoBunge* and *ChaoLee2*. Due to the instability observed for *ChaoBunge*, both methods should be executed independently. If the results provided are close, *ChaoBunge* is usually the choice. Otherwise, select *ChaoLee2*.
- If analyzing the sample we realize that each (or most) of the initial solutions reach different local optima, that is $r = M$ and $\beta_1 = M$, none of the previous methods can be applied. In this case, we can base our estimator on the proportion of local optima over the sample size [14, 43].

4

Estimating the attraction basin sizes of the local optima in permutation-based combinatorial optimization problems

4.1 Introduction

As seen in Chapter 2, the attraction basin size of the global optimum and the number of local optima can be useful indicators of the complexity of a given instance. Unfortunately, for moderate values of n these measures can not be exactly calculated. For this reason, in Chapter 3, we presented a review of different techniques to estimate the number of local optima. According to the experiments conducted, the methods that come from the statistical field obtained the best results. Their accuracy, however, is highly affected by the variance of the attraction basin sizes. That is, the more uniform the attraction basin sizes are, the better the estimation. Looking at the best performing methods (*ChaoLee* and *ChaoBunge*), we can observe that their estimations rely on the concept of sample coverage, that is, the sum of the probabilities of finding the local optima observed in the sample. This concept is clearly related to the size of the attraction basins. Hence, if we were able to obtain an accurate estimation of the sample coverage, we could improve the estimation of these methods. Therefore, our interest in this chapter lies in finding methods to estimate the attraction basin size of any local optimum.

The most intuitive way of obtaining the exact attraction basin of a local optimum, would be by exhaustively applying the local search algorithm starting from each solution of the search space, and taking those solutions that finish at such local optimum. Of course, this is useless because if we were able to do this process, we would be able to solve the optimization problem instance. So, another method for calculating the attraction basin of a local optimum is by considering, as the starting point, this local optimum. Then a recursive procedure is applied and at each time the neighbors of the current solution are checked whether they belong to the attraction basin. The procedure finishes when there are no more possible solutions to add to the attraction basin.

This last method is more efficient than the first one, as applying the local search algorithm to any solution of the search space is not required, we only need to evaluate just some of the solutions. So, we could find local optima for which this process rapidly returns their attraction basin sets. However, in general, the number of solutions to evaluate in this process grows exponentially with respect to the problem size. Therefore, for large permutation sizes, there could be local optima for which it becomes computationally intractable to exactly calculate their attraction basins.

The fact that there is no known method that calculates, in polynomial time, the exact attraction basins of the local optima, or at least, their sizes, leads us to focus on methods that estimate the sizes of the attraction basins. In Chapter 2, we used an estimation method for the sizes of the attraction basins. The method consisted of applying a local search to a sample of solutions, estimating each proportion of the size of the attraction basin of the local optima as the proportion of times that it has been reached in the sample. However, this method has a major weakness. It is supposed that there are no more local optima in the search space except just those encountered in the sample. Of course, in general, this is not true.

Given a local optimum, we propose in this chapter two methods for estimating its attraction basin size. The first method is based on taking uniformly at random solutions from the whole search space. In the second method, we take into account the structure that the neighborhood imposes in the search space. Particularly, the solutions are chosen paying special attention to their distance to such local optimum. The distance can be defined in terms of the neighborhood. For example, the distance between two permutations that are neighbors is one. According to the results, the second method provides estimations closer to the real sizes of the attraction basins, but it also implies greater computational efforts.

The rest of this chapter is organized as follows. In Section 4.2 we explain in detail both methods and, in Section 4.3 we make a comparison of their accuracy when they are applied to instances of PFSP and LOP, using the adjacent swap as well as the insert neighborhoods. Finally, in Section 4.4 we review the main conclusions obtained.

4.2 Two methods for estimating the attraction basin size

We propose two estimators to calculate the size of the attraction basin of a given local optimum π^*. This estimation will be denoted by $|\hat{\mathcal{B}}(\pi^*)|$.

- **Uniformly at random Method (UM)**
  Given a local optimum π^*, we sample solutions uniformly at random from the search space counting those that belong to its attraction basin. That is, we take a sample of size M of random initial solutions: $S = \{\pi_1, \pi_2, \ldots, \pi_M\} \subseteq \Omega$. The number of those solutions that belong to the attraction basin of π^* divided by the total number of solutions evaluated (M) is the estimated proportion of the size of the attraction basin of π^*

over the size of the search space $|\Omega|$. Therefore, the size of the attraction basin of π^* is this proportion multiplied by the size of the search space ($n!$).
In Algorithm 2 we specify the steps to follow in order to estimate the attraction basin size of π^*.

Algorithm 2 Uniformly at random Method (UM) to estimate the size of the attraction basin of a local optimum π^*

```
1: Input: $M$
2: Initialize $InAB = 0$
3: for $i = 1 \to M$ do
4:     take a random permutation $\pi_i \in \Omega$
5:     $\sigma = \mathcal{H}(\pi_i)$
6:     if $\sigma == \pi^*$ then
7:         $InAB++$
8:     end if
9: end for
10: $|\hat{\mathcal{B}}(\pi^*)| = \frac{InAB}{M} \cdot n!$
11: Output: $|\hat{\mathcal{B}}(\pi^*)|$
```

- **Distance-based Method (DM)**
  In this second proposal, we do not take a random sample directly from the whole search space Ω. Instead, given a local optimum π^*, we choose the solutions from different subsets of Ω related to π^*. That is, we consider the different subsets $D_i = \{\pi_1^i, \pi_2^i, \ldots, \pi_{|D_i|}^i\} \subseteq \Omega$ that are composed of those solutions at distance i from the local optimum π^*. We say that two permutations π_1 and π_2 are at distance i if, starting from π_1, and moving from neighboring to neighboring solutions, the length of the smallest path until reaching π_2 is i. Particularly, two neighboring permutations are at distance one.
  Notice that any permutation in $\Omega \setminus \{\pi^*\}$ should belong to one, and just one, of these subsets D_i:

$$D_i \cap D_j = \emptyset, \forall i \neq j$$

$$\bigcup_i D_i \cup \{\pi^*\} = \Omega.$$

  So, given the local optimum π^*, we take samples $S_1, S_2, \ldots$, of uniformly at random solutions at distances $1, 2, \ldots$, respectively, from π^*:

$$S_1 = \{\pi_1^1, \pi_2^1, \ldots, \pi_{M_1}^1\} \subseteq D_1,$$
$$S_2 = \{\pi_1^2, \pi_2^2, \ldots, \pi_{M_2}^2\} \subseteq D_2,$$
$$\ldots$$

We use the methods described in [52, 53, 54] to obtain these uniformly at random solutions π_j^i for the different distances. In order to estimate the size of the attraction basin of π^*, we proceed in a similar way to the previous method but, we work with the different subsets D_i independently. That is, we record the number of solutions that belong to the attraction basin of π^* in each sample set S_i, divided by the sample size considered for each distance M_i, and multiplied by the total number of permutations that exist in each subset D_i ($|D_i|$). Therefore, the sum of these quantities obtained for each distance plus one (π^* itself is in its attraction basin and has not been considered in any subset), is the resultant attraction basin size of the local optimum π^*. This process is detailed in Algorithm 3, where $MaxDist$ denotes the maximum distance between two permutations and $|D_{dist}|$ refers to the number of permutations at distance $dist$. Both the maximum distance and the number of permutations at a given distance depend on the problem size and the neighborhood used.

Algorithm 3 Distance-based Method (DM) to estimate the size of the attraction basin of a local optimum π^*.

```
1: Input: $\mathbf{M} = \{M_1, \dots, M_{MaxDist}\}$
2: $|\hat{\mathcal{B}}(\pi^*)| = 1$
3: **for** $dist = 1 \to MaxDist$ **do**
4:    Initialize $InAB = 0$
5:    **for** $j = 1 \to M_{dist}$ **do**
6:       take a random permutation $\sigma \in D_{dist}$
7:       $\pi = \mathcal{H}(\sigma)$
8:       **if** $\pi == \pi^*$ **then**
9:          $InAB++$
10:      **end if**
11:   **end for**
12:   $|\hat{\mathcal{B}}(\pi^*)| = |\hat{\mathcal{B}}(\pi^*)| + \frac{InAB}{M_{dist}} \cdot |D_{dist}|$
13: **end for**
14: Output: $|\hat{\mathcal{B}}(\pi^*)|$
```

The input parameter of the algorithm is $\mathbf{M} = \{M_1, \dots, M_{MaxDist}\}$, that is, we need to set in advance the sample size used at each distance $dist$. On the one hand, different subsets D_i have different sizes, which could lead us to change the sample size taking values proportional to $|D_i|$. On the other hand, the closer a permutation to the optimum, the more probable it belongs to its attraction basin. Thus, we could consider the possibility of taking more random solutions in the subsets D_i than D_j, with $i < j$, or even, to stop taking solutions from a certain distance on.

4.3 Experiments

We analyze and compare the two proposed methods for estimating the sizes of the attraction basins of the local optima for instances of different problems and considering different neighborhoods.

We work with instances of the PFSP and the LOP, and we focus on two common neighborhoods: adjacent swap and swap. For the PFSP we consider 5 instances with 10 jobs and 5 machines, obtained from the well-known benchmark proposed by Taillard [1]. The 5 instances of the LOP have been obtained from the xLOLIB benchmark [76], and the matrix size considered is 10x10.

So, in both problems the size of the search space is $10!$. Notice that the problem size is quite small. The reason is that, in order to measure the accuracy of the methods, we calculated the exact sizes of the attraction basins of each of the local optima of the instances, and, therefore, working with large permutation sizes becomes computationally unaffordable.

Regarding the parameters of the algorithms, we need to specify the sample sizes used. For the first method we choose samples of sizes: $M = \{1125, 2250, 4500\}$. For the second method, we need to fix different sample sizes M_i according to the distance. In order to study different possibilities, for the second method we consider three different cases:

1. **Equal Sample sizes for each distance (ES):** $M_i = M_j, \forall i \neq j$.

2. **Sample sizes Proportional to the number of permutations at each distance (SP):** $M_i \propto |D_i|$.

3. **Sample sizes Decreasing as the distance increases (SD):** $M_i \propto \frac{1}{i}$, and $M_i = 0, i > MaxDist/2$.

We should use the same (or almost similar) total sample size, when comparing the second method with the first one. So, we need to choose, in this case, M_i such that

$$M \approx \sum_{i=1}^{MaxDist} M_i \approx \begin{cases} 1125 \\ 2250 \\ 4500 \end{cases} \tag{4.1}$$

We show in Table 4.1 the sample sizes used at each distance according to the neighborhood, in order to fulfill equation (4.1). In Table 4.2, the number of permutations of size 10 at each distance from a given permutation for the adjacent swap and swap neighborhoods is facilitated.

Both algorithms are applied 10 times to each local optimum for each sample size, and the average estimations of the sizes of the attraction basins are recorded and compared to the real one.

[1] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir /ordonnancement.html

Table 4.1. Sample sizes used.

	Adjacent swap			Swap		
UM	DM-ES	DM-SP	DM-SD ($i \leq 22$)	DM-ES	DM-SP	DM-SD ($i \leq 4$)
1125	25	$\left\lceil \frac{\lvert D_i \rvert}{3300} \right\rceil$	$\left\lceil \frac{302}{i} \right\rceil$	125	$\left\lceil \frac{\lvert D_i \rvert}{3240} \right\rceil$	$\left\lceil \frac{540}{i} \right\rceil$
2250	50	$\left\lceil \frac{\lvert D_i \rvert}{1630} \right\rceil$	$\left\lceil \frac{608}{i} \right\rceil$	250	$\left\lceil \frac{\lvert D_i \rvert}{1616} \right\rceil$	$\left\lceil \frac{1080}{i} \right\rceil$
4500	100	$\left\lceil \frac{\lvert D_i \rvert}{811} \right\rceil$	$\left\lceil \frac{1217}{i} \right\rceil$	500	$\left\lceil \frac{\lvert D_i \rvert}{807} \right\rceil$	$\left\lceil \frac{2160}{i} \right\rceil$

Table 4.2. Number of permutations of size 10 at different distances from a given permutation according to the adjacent swap and the swap neighborhoods.

Adjacent swap						Swap	
dist	#perms	dist	# perms	dist	# perms	dist	# perms
1	9	16	135853	31	86054	1	45
2	44	17	162337	32	64889	2	870
3	155	18	187959	33	47043	3	9450
4	440	19	211089	34	32683	4	63273
5	1068	20	230131	35	21670	5	269325
6	2298	21	243694	36	13640	6	723680
7	4489	22	250749	37	8095	7	1172700
8	8095	23	250749	38	4489	8	1026576
9	13640	24	243694	39	2298	9	362880
10	21670	25	230131	40	1068		
11	32683	26	211089	41	440		
12	47043	27	187959	42	155		
13	64889	28	162337	43	44		
14	86054	29	135853	44	9		
15	110010	30	110010	45	1		

We first analyze the estimations obtained for the attraction basin size of the global optimum. In Table 4.3, we report the relative errors $\frac{||\hat{\mathcal{B}}(\pi^*)|-|\mathcal{B}(\pi^*)||}{|\mathcal{B}(\pi^*)|}$ for the UM and the DM distinguishing the three different ways of taking the sample (ES, SP and SD), and according to the problem and neighborhood considered. The values indicated in this table are, for each instance, the average errors obtained from the 10 repetitions of each method. As we can appreciate, when estimating the attraction basin size of the global optimum, there is not a best overall method. However, we can obtain some conclusions from this table. The estimation methods provide quite different results for different instances. This fact leads us to think of the diversity of instances that we can find when considering the same problem and neighborhood. In general, for both problems, UM, DM-ES and DM-SP obtain estimations that are worse for the adjacent swap neighborhood than for the swap neighborhood. On the contrary, the results given by the DM-SD when considering the swap neighborhood are really poor. We must remember that the DM-SD takes a lower number of solutions as the distance to the global optimum increases, and it stops taking solutions at $MaxDist/2$. The bad performance of this method indicates that, when using the swap neighborhood, the attraction basin of the global optimum is composed by a high number of solutions that are far from it, but this does not occur when using the adjacent swap neighborhood. Of course, even if we increase the sample size with this method, as we stop considering solutions at certain distance, for both neighborhoods we do not obtain better results. For the UM, in general, we observe that as we increase the sample size, the estimations improve. Nevertheless, this does not happen with the DM. In order to show the average exact values of attraction basin sizes obtained with the different methods, as well as the real sizes, we include Table 4.4. In this table we can appreciate the global accuracy of the estimations. In fact, we observe the poor estimations given by the DM-SD for the swap neighborhood, which are always much lower than the real values.

Table 4.3. Relative errors of the attraction basin sizes of the global optimum.

			$M = 1125$				$M = 2250$				$M = 4500$			
			UM	DM-ES	DM-SP	DM-SD	UM	DM-ES	DM-SP	DM-SD	UM	DM-ES	DM-SP	DM-SD
PFSP	Adjacent swap	1	0.25	**0.10**	0.24	0.24	0.23	0.10	**0.04**	0.07	**0.03**	0.24	0.22	**0.03**
		2	0.21	0.18	**0.04**	0.23	**0.00**	0.03	0.12	0.10	0.31	**0.00**	0.01	0.11
		3	0.44	0.20	0.27	**0.03**	0.12	0.10	**0.04**	**0.04**	0.23	**0.00**	0.22	0.02
		4	0.80	**0.01**	0.08	0.10	0.10	0.10	0.21	**0.04**	0.10	**0.04**	0.22	0.06
		5	**0.02**	0.17	0.33	0.45	**0.02**	**0.02**	0.23	0.08	0.12	**0.02**	0.06	0.03
	Swap	1	**0.00**	0.03	0.06	0.61	0.02	**0.01**	**0.01**	0.62	0.02	0.02	**0.00**	0.61
		2	0.08	0.07	**0.03**	0.55	**0.00**	0.05	**0.00**	0.56	0.03	0.04	**0.01**	0.59
		3	0.07	**0.01**	0.02	0.68	0.05	0.02	**0.01**	0.69	**0.02**	**0.02**	0.03	0.69
		4	**0.02**	0.05	0.07	0.39	**0.04**	0.06	0.07	0.43	0.02	0.02	**0.01**	0.40
		5	**0.01**	0.03	**0.01**	0.71	**0.00**	0.02	**0.00**	0.72	**0.00**	**0.00**	0.01	0.72
LOP	Adjacent swap	1	**0.01**	0.05	0.03	0.17	0.05	**0.01**	**0.01**	0.11	0.02	0.05	**0.00**	0.08
		2	0.71	0.19	0.25	**0.02**	0.33	0.13	0.15	**0.11**	0.17	**0.05**	0.10	0.08
		3	0.92	**0.11**	0.37	0.18	0.39	**0.03**	0.26	0.12	0.09	**0.07**	0.15	0.09
		4	0.09	**0.03**	0.20	0.17	**0.01**	0.09	0.04	0.05	0.03	0.04	0.02	**0.01**
		5	**0.04**	0.06	0.06	0.14	0.04	0.02	**0.01**	0.07	0.04	**0.00**	**0.00**	0.07
	Swap	1	**0.00**	0.05	0.01	0.79	**0.00**	0.04	0.01	0.80	**0.00**	0.01	**0.00**	0.80
		2	0.07	**0.02**	0.03	0.65	**0.02**	0.04	**0.02**	0.63	0.04	0.04	**0.01**	0.65
		3	0.10	**0.00**	0.02	0.46	**0.00**	0.12	**0.00**	0.38	0.04	0.07	**0.00**	0.44
		4	0.01	0.01	**0.00**	0.81	0.01	**0.00**	0.01	0.81	**0.01**	**0.01**	0.02	0.81
		5	0.01	0.01	**0.00**	0.81	0.02	0.01	**0.00**	0.81	**0.00**	0.01	**0.00**	0.81

Table 4.4. Average estimations for the attraction basin sizes of the global optimum.

			AB size	$M = 1125$				$M = 2250$				$M = 4500$			
				UM	DM-ES	DM-SP	DM-SD	UM	DM-ES	DM-SP	DM-SD	UM	DM-ES	DM-SP	DM-SD
PFSP	Adjacent swap	1	4396	5485.10	**4856.10**	3345.10	3347.40	3388.30	4851.00	**4550.20**	4709.80	**4517.40**	5454.60	5353.70	4259.80
		2	1606	1936.60	1311.20	**1546.40**	1230.60	**1614.00**	1655.20	1408.90	1452.00	2098.20	**1606.30**	1629.00	1780.80
		3	2304	1291.40	1851.60	1680.90	**2365.80**	2581.70	2539.80	**2221.10**	2211.10	1775.30	**2307.80**	1798.70	2253.20
		4	538	968.80	**535.10**	495.10	486.60	484.90	484.40	652.70	**515.80**	485.10	**557.80**	658.80	567.70
		5	12915	**13226.40**	15169.10	17119.60	18687.60	13226.20	**13181.70**	9898.20	11918.20	14516.60	**12721.10**	13695.50	13324.40
	Swap	1	262394	**263532.90**	255786.40	277957.70	101838.40	267887.40	**263801.10**	265867.00	99096.40	257646.10	257333.90	**262530.10**	102443.40
		2	149913	161281.30	160388.40	**154900.30**	66776.30	150636.90	157029.00	**149635.70**	66279.60	145395.30	155370.40	**148659.90**	62205.20
		3	383869	411910.50	**379717.30**	375513.90	123252.40	365623.10	377877.60	**379652.80**	117901.90	391831.20	**391113.70**	395266.00	119910.80
		4	70557	**71932.20**	67285.80	75289.50	43061.60	**68061.70**	74618.80	75688.30	40552.60	72012.70	71719.80	**69886.90**	42234.80
		5	634073	**628025.70**	613812.60	627908.90	181696.70	**634315.70**	623906.30	634350.30	176275.80	**632864.20**	631700.70	638120.90	178065.60
LOP	Adjacent swap	1	67124	**68061.60**	70449.70	65160.30	78359.40	63868.40	68042.10	**66731.40**	60024.90	68706.70	70390.20	**66865.90**	72744.10
		2	3370	968.80	2743.40	4227.10	**3314.90**	2259.20	2934.50	3862.10	**3728.20**	3952.90	**3192.60**	3714.10	3631.20
		3	1847	3549.50	**2049.00**	1164.50	1517.20	1130.10	**1796.50**	2332.80	1622.60	2017.30	**1980.30**	2119.60	2010.80
		4	34497	31289.80	**33365.40**	41377.80	40227.40	**34676.70**	37589.60	33250.10	36320.60	33305.70	35809.70	35022.30	**34000.30**
		5	208047	**216761.80**	195707.80	219492.80	178351.30	199343.60	212924.20	**211088.00**	192945.80	216116.70	209029.10	**207467.40**	193339.20
	Swap	1	668046	**670281.00**	698986.30	672843.60	139062.10	**670603.60**	693298.40	674397.70	134614.50	**669393.90**	671496.00	664826.30	135413.40
		2	194291	208375.40	**190376.40**	189409.00	68737.00	**189505.40**	201257.30	189450.00	71451.40	187166.90	186765.90	**191706.90**	68667.90
		3	28001	25161.20	**27893.30**	28593.80	15058.50	27902.80	31346.90	**27919.00**	17271.30	26935.10	30090.50	**28076.20**	15562.20
		4	1015176	1029613.10	1007928.30	**1017755.40**	192397.80	1020581.30	**1018401.30**	1021491.70	189691.70	**1020258.70**	1020772.60	999638.20	190564.70
		5	1154196	1140251.10	1137721.40	**1148887.10**	218336.40	1178313.20	1162159.80	**1153829.30**	217650.40	1158959.50	1170939.80	**1156797.70**	217923.30

We continue with the comparison of the methods by reporting in Table 4.5 the average relative error and the variances (in brackets) given by each method for all the local optima (including the global optimum) of each instance. We observe that, clearly, on average terms, the method that provides the best results for the instances of the PFSP considering the adjacent swap neighborhood is DM-SD, where we find the lowest errors and very small variances. For the instances of the LOP considering the adjacent swap neighborhood, the DM-SD performs well, but we also find good results for the DM-ES. For both problems, when using the swap neighborhood, as was observed for the attraction basin size of the global optimum, the results given by DM-SD are really bad (high errors and high variances). The best method on average for almost all instances for the swap neighborhood is DM-ES. So, for this neighborhood it seems convenient to take the same number of solutions at different distances. As a general rule, the higher the sample size, the lower the average relative errors and variances. Except for the DM-SD when using the swap neighborhood, that results are almost similar for the different sample sizes.

We carry out a statistical analysis to compare the estimation obtained for the different methods. A nonparametric Friedman's test with level of significance $\alpha = 0.05$ is used to test if there are statistical significant differences between the estimates provided by the 4 methods in the different scenarios (according to problem and neighborhood). This test provides a ranking of the methods while also giving an average rank value for each method. As we always find statistical differences in all the cases, we proceed with a post-hoc test which carries out all pairwise comparisons. Particularly, we use the Holm's procedure fixing the level of significance to $\alpha = 0.05$. In the case of the PFSP with the adjacent swap neighborhood, we find that the best method is the DM-SD with significant differences. When using the swap neighborhood, the best method is the DM-ES but with no significant differences with the DM-SP. For the LOP and the adjacent swap neighborhood, the best methods are the DM-SD and the DM-ES with no significant differences, while for the swap neighborhood the UM, the DM-ES and the DM-SP are the best methods without significant differences among them. Of course, for both problems with the swap neighborhood the worst performing one is the DM-SD with significant differences.

Table 4.5. Average errors (and variances) of the attraction basin sizes of the local optima.

			$M = 1125$				$M = 2250$				$M = 4500$			
			UM	DM-ES	DM-SP	DM-SD	UM	DM-ES	DM-SP	DM-SD	UM	DM-ES	DM-SP	DM-SD
PFSP	Adjacent swap	1	1.16 (5.13)	0.19 (0.03)	0.56 (0.33)	**0.16 (0.03)**	0.90 (1.77)	0.14 (0.02)	0.49 (0.25)	**0.12 (0.02)**	0.75 (1.08)	0.10 (0.01)	0.40 (0.14)	**0.09 (0.01)**
		2	1.30 (4.61)	0.15 (0.02)	0.55 (0.30)	**0.12 (0.01)**	1.06 (2.16)	0.11(0.01)	0.50 (0.22)	**0.09 (0.01)**	0.84 (1.06)	0.11(0.01)	0.43 (0.14)	**0.07 (0.00)**
		3	1.33 (5.26)	0.14 (0.01)	0.51 (0.25)	**0.10 (0.01)**	1.11 (2.95)	0.10 (0.01)	0.47 (0.21)	**0.08 (0.00)**	0.83 (1.27)	0.08 (0.00)	0.42 (0.14)	**0.07 (0.00)**
		4	1.49 (8.77)	0.14 (0.02)	0.54 (0.31)	**0.11 (0.01)**	1.17 (3.05)	0.11 (0.01)	0.51 (0.24)	**0.08 (0.01)**	1.02 (2.27)	0.12 (0.01)	0.45 (0.15)	**0.07 (0.00)**
		5	0.60 (0.46)	0.24 (0.05)	0.46 (0.22)	**0.23 (0.04)**	0.45 (0.24)	0.18 (0.03)	0.35 (0.13)	**0.17 (0.03)**	0.34 (0.15)	**0.13 (0.01)**	0.26 (0.07)	**0.13 (0.01)**
	Swap	1	0.08 (0.01)	**0.05 (0.00)**	0.07 (0.00)	0.43 (0.05)	0.06 (0.01)	**0.04 (0.00)**	0.05 (0.01)	0.43 (0.05)	**0.03 (0.00)**	**0.03 (0.00)**	0.04 (0.00)	0.42 (0.05)
		2	0.30 (0.13)	**0.13 (0.02)**	0.25 (0.10)	0.24 (0.03)	0.24 (0.11)	**0.09 (0.01)**	0.20 (0.09)	0.21 (0.02)	0.19 (0.06)	**0.07 (0.00)**	0.14 (0.03)	0.21 (0.02)
		3	0.22 (0.10)	**0.10 (0.01)**	0.17 (0.03)	0.24 (0.03)	0.15 (0.05)	**0.08 (0.00)**	0.13 (0.02)	0.25 (0.03)	0.10 (0.01)	**0.05 (0.00)**	0.10 (0.02)	0.23 (0.03)
		4	0.40 (0.52)	**0.13 (0.01)**	0.29 (0.11)	0.21 (0.02)	0.22 (0.12)	**0.10 (0.01)**	0.20 (0.06)	0.18 (0.02)	0.19 (0.07)	**0.07 (0.00)**	0.15 (0.04)	0.17 (0.02)
		5	**0.04 (0.00)**	0.05 (0.00)	0.05 (0.00)	0.55 (0.05)	0.03 (0.00)	0.03 (0.00)	**0.02 (0.00)**	0.54 (0.05)	**0.02 (0.00)**	0.03 (0.00)	**0.02 (0.00)**	0.55 (0.05)
LOP	Adjacent swap	1	0.93 (2.51)	0.21 (0.03)	0.56 (0.44)	**0.18 (0.03)**	0.79 (1.81)	0.15 (0.02)	0.48 (0.25)	**0.13 (0.01)**	0.55 (0.77)	0.11 (0.01)	0.37 (0.14)	**0.10 (0.01)**
		2	1.37 (10.99)	0.20 (0.03)	0.65 (0.48)	**0.16 (0.02)**	1.02 (2.29)	0.15 (0.02)	0.57 (0.29)	**0.12 (0.01)**	0.79 (1.36)	0.11 (0.01)	0.48 (0.22)	**0.10 (0.01)**
		3	1.64 (28.06)	0.24 (0.05)	0.73 (0.80)	**0.20 (0.04)**	1.50 (16.84)	0.18 (0.03)	0.68 (0.57)	**0.16 (0.02)**	1.26 (6.72)	0.15 (0.01)	0.61 (0.36)	**0.14 (0.01)**
		4	0.44 (0.19)	**0.26 (0.09)**	0.36 (0.15)	0.27 (0.07)	0.34 (0.21)	**0.20 (0.04)**	0.27 (0.12)	**0.20 (0.04)**	0.26 (0.11)	**0.13 (0.01)**	0.21 (0.08)	**0.13 (0.01)**
		5	0.33 (0.13)	**0.23 (0.05)**	0.26 (0.10)	0.27 (0.07)	0.19 (0.05)	**0.17 (0.04)**	0.19 (0.04)	0.19 (0.03)	0.14 (0.02)	**0.11 (0.01)**	0.14 (0.03)	0.13 (0.02)
	Swap	1	0.18 (0.03)	**0.10 (0.01)**	0.15 (0.03)	0.33 (0.07)	0.13 (0.03)	**0.09 (0.01)**	0.12 (0.02)	0.32 (0.07)	0.11 (0.01)	**0.05 (0.00)**	0.09 (0.01)	0.31 (0.07)
		2	0.32 (0.91)	**0.12 (0.02)**	0.23 (0.09)	0.33 (0.05)	0.16 (0.05)	**0.08 (0.01)**	0.18 (0.07)	0.30 (0.04)	0.13 (0.06)	**0.06 (0.00)**	0.12 (0.02)	0.29 (0.04)
		3	0.43 (0.27)	**0.19 (0.03)**	0.33 (0.12)	0.23 (0.02)	0.32 (0.13)	**0.13 (0.01)**	0.25 (0.07)	0.20 (0.02)	0.19 (0.04)	**0.09 (0.01)**	0.18 (0.04)	0.17 (0.02)
		4	**0.02 (0.00)**	0.03 (0.00)	0.05 (0.01)	0.60 (0.03)	**0.04 (0.00)**	**0.04 (0.00)**	**0.04 (0.00)**	0.60 (0.03)	0.02 (0.00)	0.02 (0.00)	**0.01 (0.00)**	0.59 (0.04)
		5	**0.06 (0.00)**	0.09 (0.01)	0.10 (0.02)	0.49 (0.04)	**0.05 (0.00)**	0.06 (0.00)	0.07 (0.01)	0.47 (0.05)	**0.04 (0.00)**	**0.04 (0.00)**	0.05 (0.01)	0.46 (0.05)

4.4 Conclusions

We have presented two methods for estimating the size of the attraction basin of a given local optimum. These estimators are based on the proportion of solutions that belong to its attraction basin when a sample is taken from the whole search space. The estimation of the attraction basin sizes of the local optima could help to, for example, estimate the proportion of the search space that has been explored and, therefore, to have knowledge of the shape of the landscape.

In the first method proposed (UM), the proportion of the size of the attraction basin of the local optimum is estimated as the proportion of solutions of a random sample that belong to it. The second method (DM), which is more computationally demanding, consists of taking random solutions at different distances from the search space, and estimating the total size considering the sum of the estimations of the sizes that are related to each distance. In this second method, the sample size taken at each distance is of high relevance. We have noticed differences in considering different ways of taking the samples for the different neighborhoods.

First, we consider the case where the same sample size is taken for each distance (DM-ES). Then, we take samples of sizes proportional to the number of permutations that are at the different distances (DM-SP). Finally, the samples are chosen with sizes that decrease as the distance to the local optimum increases, and we stop taking solutions further than $MaxDist/2$ (DM-SD).

The main result observed is that for the swap neighborhood, the DM-SD provides bad estimations. We have concluded that this is due to the fact that, for this neighborhood, the attraction basins of the local optima must have a high number of solutions far from it. The methods perform similar for instances of both problems. However, we observe differences in the estimations provided for the different instances considering the same problem. Another important observation derived from this analysis is that the sample size does not have a high influence on the three versions of the DM, while it has to be taken into account if we use the UM. Of course, the higher the sample size considered in the UM, the more accurate the estimations.

After observing the results of the statistical analysis, we recommend the following:

- If we are working under the adjacent swap neighborhood, we should apply the DM-SD .
- If we are under the swap neighborhood, the use of the DM-ES or the DM-SP is recommended.

We could study the estimations provided by these methods considering other different neighborhoods. The performance of these methods, of course, depend on some properties of the instances, above all, the distributions of the sizes of the attraction basins. Therefore, we could think of other ways of taking the samples according to the different distances. However, these methods should be designed focusing on the specific distance considered, as the way of choosing the sample sizes is the most important aspect that influences their behavior.

Part II

A tunable generator of instances of permutation-based combinatorial optimization problems

5

Generator of instances

5.1 Introduction

In the optimization arena, the evaluation of algorithms is usually measured by means of benchmark problems. However, algorithms show different behaviors for different problems, or even for different instances of the same problem. So commonly, when trying to study their performance, assumptions are needed to be made on the algorithm itself, the problem to which it is applied or the specific instance of the problem. Thus, having information about the characteristics of the problem instances at hand would be really useful for improving the design of algorithms or to identify the best performing algorithm from a toolbox. Due to the difficulty of having real instances whose properties we know a priori at our disposal, generators are designed in order to provide a large set of instances with different characteristics. Therefore, a tunable generator of optimization problem instances that depends on a reduced number of parameters able to control the properties of the instance is an important and useful tool for this purpose.

In [75] a software framework that generates multimodal test functions for optimization in the continuous field was presented. In [37, 66, 67, 95] the authors proposed a continuous search space generator based on a mixture of Gaussians. A proposal in the discrete domain, and particularly for binary spaces, can be found in [25], where a NK landscape generator is described. We also find a proposal of a generator of instances in the dynamics optimization field: the moving peaks benchmark [10].

Unfortunately, these generators lack the flexibility to generate instances with controlled properties. Most of them are based on populating a cost matrix (depending on the problem they focus on) by taking random values, or taking random points in a square and calculating the distances between each pair of points. The complexity of the instances varies according to the interval (values and variances) used during the random sampling. On the other hand, as pointed out in [28, 70], there are proposals in the literature that create instances where the optimum is known. Nevertheless, this is the only information provided, and there is no clue about how easy or difficult it is to solve the instance. Precisely, in [70], the authors state that the toughest technical challenges are finding (or generating) suitable test instances, and assessing how close heuristic algorithms come to the optimum.

Related to this, we present a generator for permutation-based COPs. Our work is inspired by [37], where the authors proposed a generator of optimization problem instances in the continuous domain. We adapt the concepts that they used to those that are similar but in the permutation space. For example, instead of working with Gaussian density functions we work with Generalized Mallows distributions. This distribution is an exponential probability model that depends on a consensus permutation and spread parameters (analogous to the mean and the variance in the Gaussian function). We study the influence of the parameters that the generator uses when creating the instances. We also provide some clues on how to tune them with the aim of controlling the properties of the instances. We pay special attention to a particular property, the number of local optima, and set restrictions in the parameters so as to have a predefined number of them in the resultant instance. In order to obtain a solution for these restrictions in the parameters, the generator is seen as a linear programming problem, and a linear function is defined that promotes obtaining qualitative properties in the generated instance.

In our framework, the cost of generating the instances, as well as the cost of evaluating each solution of the search space, is dominated by the number of local optima. In order to test our generator, we take into account properties that generators of problem instances in optimization should have when the resultant instances are used to evaluate Evolutionary Algorithms [36, 37, 50, 93]. Specifically, we carry out experiments to evaluate two important properties: the flexibility of the generator, and its ability to create instances of very different complexity levels for common metaheuristics. To test the first property, we adjust different parameters of our generator. We considered it interesting to check the ability of the generator to provide, for small problem sizes, instances almost identical to those found in well-known benchmarks. We compare the instances by means of the sizes of the attraction basins of the local optima. To evaluate the second property of the generator, we generate instances playing with the size of the attraction basins for the global and local optima, in addition to varying the location of the different local optima. Then, three different algorithms, a Local Search (LS), an Estimation of Distribution Algorithms (EDA), and a Genetic Algorithm (GA) are applied to observe their behavior when solving the different types of instances. According to the results, the generator arises as a very useful tool to perform coarse as well as fine grain analysis of optimization algorithms, as the practitioner can observe the algorithms under controlled scenarios (instances). Derived from the experiments, an interesting and novelty property in the performance of the EDA and the GA is observed.

The rest of the chapter is organized as follows. In Section 5.2 we explain the Mallows and the Generalized Mallows models, which are the basis of our generator. In Section 5.3 the most common distances used for the Generalized Mallows model are detailed. Our generator is introduced in Section 5.4 and additional details are provided in Section 5.5. In Section 5.6 we present three examples of how to tune the parameters involved in the model to create instances with different properties. The experimentation is shown in Section 5.7, firstly, testing the flexibility of the model by comparing created instances with instances of common benchmarks, and secondly, comparing and analyzing properties in the performance of different metaheuristics when applying them to instances created with our generator with different input parameters. Finally, the conclusions are presented in Section 5.8.

5.2 Mallows and Generalized Mallows models

5.2.1 Mallows model

The Mallows model [60] is an exponential probability model for permutations based on a distance. Inside the space of permutations Ω, a special permutation which is worth mentioning is the identity permutation, $e = (123 \cdots n)$ which maps each item i to position i. By composing two permutations σ and π of n items, we obtain a new permutation $\sigma\pi$ such that $\sigma\pi(j) = \sigma(\pi(j))$. Specifically, the inverse π^{-1} of π is the permutation such that $\pi\pi^{-1} = e$. Note that the composition between a permutation and the identity is the same permutation: $\pi e = e\pi = \pi$.

The Mallows distribution is specified by two parameters: the consensus permutation $\pi_0 \in \Omega$ and the spread parameter $\theta \in \mathbb{R}$. Hence, the probability assigned to each $\pi \in \Omega$ is:

$$p(\pi|\pi_0, \theta) = \frac{1}{Z(\theta)} e^{-\theta d(\pi, \pi_0)}$$

where $Z(\theta) = \sum_{\pi' \in \Omega} e^{-\theta d(\pi', \pi_0)}$ is a normalization term and $d(\pi, \pi_0)$ is a distance between π and the consensus permutation π_0.

Different values for the parameter θ produce different distributions. For instance, when $\theta = 0$ it is the uniform distribution. However, when $\theta > 0$, then π_0 is the permutation with the highest probability. The rest of permutations $\pi \in \Omega - \{\pi_0\}$ have probability inversely exponentially proportional to θ and their distance to π_0. In this sense, the Mallows distribution with $\theta > 0$ is usually considered analogous to the Gaussian distribution on the space of permutations. In Figure 5.1 we represent the probability assigned to the permutations of size $n = 5$ for different θ values considering the Kendall-tau metric (this distance is introduced in Section 5.3.1). In the X axis we represent the permutations symmetrically distributed according to their distance to the consensus permutation π_0, in order to see the analogy with the Gaussian distribution. We observe that the larger the value of θ, the more peaked the distribution becomes around the consensus permutation.

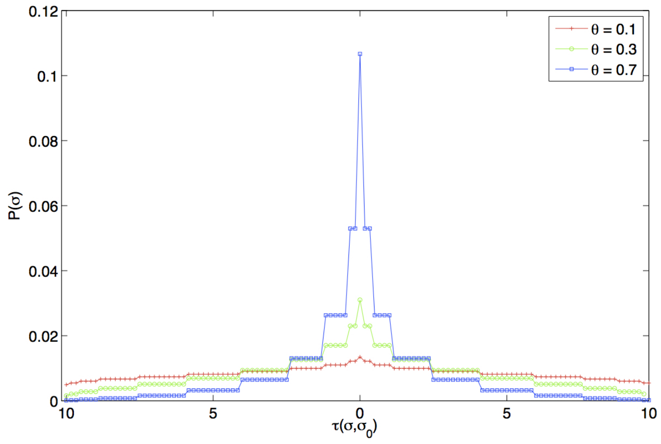


Fig. 5.1. Probability assigned to permutations by three Mallows functions centered at π_0 and with $\theta = 0.1, 0.3$ and 0.7. In the X axis the permutations are distributed symmetrically according to their distance to π_0.

5.2.2 Generalized Mallows model

The Mallows model is a simple yet efficient probability model for permutations. However, the fact that it uses just a single spread parameter limits its flexibility. In this sense, it assigns the same probability to all the permutations that are at the same distance from the consensus permutation. That is, for any distance d:

$$\forall \pi \neq \pi' \in \Omega \text{ s.t. } d(\pi, \pi_0) = d(\pi', \pi_0), \text{ then } p(\pi) = p(\pi').$$

In [33], an extension of this model was proposed, called the Generalized Mallows (GM) model. For this model, we need to work with a distance which is able to be decomposed $d(\pi_1, \pi_2) = \sum_{s=1}^{n-1} d_s(\pi_1, \pi_2)$, in such a way that each d_s is related with the s-th item (position) of the permutation. Therefore, a different spread parameter $\theta^s \in \mathbb{R}$ can be associated to each of the elements d_s that is involved in the decomposition of the distance, so that instead of using one spread parameter, a vector of them is defined. In this distribution, the probability assigned to any permutation $\pi \in \Omega$ is:

$$p(\pi|\pi_0, \theta^1, \ldots, \theta^{n-1}) = \frac{1}{Z(\theta^1, \ldots, \theta^{n-1})} e^{-\sum\limits_{s=1}^{n-1} \theta^s d_s(\pi, \pi_0)},$$

where

$$Z(\theta^1, \ldots, \theta^{n-1}) = \sum_{\pi' \in \Omega} e^{-\sum_{s=1}^{n-1} \theta^s d_s(\pi', \pi_0)}.$$

Notice that, the Mallows model can be seen as a particular case of the GM model, where $\theta^s = \theta, \forall s$. So, from now on, we will refer to the GM model taking the Mallows model as a specific case of it, and we will consider that $\theta^s \in \mathbb{R}^+, \forall s$.

5.3 Distances

As seen in the previous section, the GM model makes use of a distance or metric. The metrics utilized in this chapter are the Kendall-tau and the Cayley distances, as they are the most commonly jointly used with this probabilistic model [34, 54, 56, 58, 61].

We define a distance or metric between two permutations π_1 and π_2 [26] as a function

$$\begin{aligned} d: \ \Omega \text{x} \Omega &\longrightarrow \mathbb{R} \\ (\pi_1, \pi_2) &\longmapsto d(\pi_1, \pi_2) \end{aligned}$$

that fulfills the following properties $\forall \pi_1, \pi_2, \pi_3 \in \Omega$:

1. Non-negativity: $d(\pi_1, \pi_2) \geq 0$ (with equality iff $\pi_1 = \pi_2$).
2. Symmetry: $d(\pi_1, \pi_2) = d(\pi_2, \pi_1)$.
3. Triangle inequality: $d(\pi_1, \pi_2) \leq d(\pi_1, \pi_3) + d(\pi_3, \pi_2)$.

Specifically, the two metrics considered in this chapter also fulfill the left-invariant property:

$$d(\pi_1, \pi_2) = d(\pi_3\pi_1, \pi_3\pi_2).$$

5.3.1 Kendall-tau distance

The most commonly used distance in the GM model is the Kendall-tau [13, 21, 61]. This metric measures the minimum number of adjacent swaps between two permutations. Formally, given two permutations π_1 and π_2, we write this metric as:

$$d_K(\pi_1, \pi_2) = \sum_{r \prec_{\pi_1} s} \mathbb{1}_{[s \prec_{\pi_2} r]} \tag{5.1}$$

where $r \prec_\pi s$ means that the item r precedes s in the permutation π, and $\mathbb{1}_{[\cdot]}$ denotes the indicator function. As it fulfills the left-invariant property, we can write indistinctly $d_K(\pi_1, \pi_2)$ or $d_K(\pi_2^{-1}\pi_1, e)$. The Kendall-tau distance can be decomposed into the following form:

$$d_K(\pi_1, \pi_2) = d_K(\pi_2^{-1}\pi_1, e) = \sum_{s=1}^{n-1} V_s(\pi_2^{-1}\pi_1, e) \tag{5.2}$$

where $V_s(\pi, e)$ $(s = 1, 2, ..., n-1)$ is the number of items $r > s$ that precede the item s in π, that is:

$$V_s(\pi, e) = \sum_{r>s} \mathbb{1}_{[r \prec_\pi s]}.$$

Notice, that we define each $V_s(\pi, e)$ with respect to the identity permutation, so that when decomposing the Kendall-tau distance between two permutations π_1 and π_2, we just need to consider the distance between the composition $\pi_2^{-1}\pi_1$ and e. From now, we simplify the notation by writing $V_s(\pi)$ instead of $V_s(\pi, e)$.

For example, lets suppose that we have two permutations of size $n = 4$:

$$\pi_1 = (4132) \quad \text{and} \quad \pi_2 = (1342).$$

Thus,

$$\pi_2^{-1} = (1423) \quad \text{and} \quad \pi_2^{-1}\pi_1 = (3124).$$

Therefore,

$$d_K(\pi_1, \pi_2) = d_K(\pi_2^{-1}\pi_1, e) = \sum_{s=1}^{n-1} V_s(\pi_2^{-1}\pi_1) = 1 + 1 + 0 = 2.$$

As the terms V_s (for $s = 1, 2, ..., n-1$) are bounded:

$$0 \leq V_s \leq (n - s),$$

the maximum distance between two permutations that can be reached by this metric is

$$(n-1) + (n-2) + ... + 1 = \frac{n(n-1)}{2}.$$

When using the Kendall-tau metric in the GM model, we consider a different spread parameter θ^s associated to each V_s. So that the model depends on a consensus permutation $\pi_0 \in \Omega$ and the spread parameters $\boldsymbol{\theta} = (\theta^1, \cdots, \theta^{n-1}) \in (\mathbb{R}^+)^{n-1}$. The normalization term $Z(\boldsymbol{\theta}) = Z(\theta^1, \ldots, \theta^{n-1})$ has the following closed form that does not depend on π_0:

$$Z(\boldsymbol{\theta}) = \prod_{s=1}^{n-1} \left[\frac{1 - e^{-(n-s+1)\theta^s}}{1 - e^{-\theta^s}} \right].$$

Therefore the probability assigned to each permutation $\pi \in \Omega$ can be written as:

$$p(\pi|\pi_0, \boldsymbol{\theta}) = \prod_{s=1}^{n-1} \left[\frac{1 - e^{-\theta^s}}{1 - e^{-(n-s+1)\theta^s}} \right] e^{-\sum_{s=1}^{n-1} \theta^s V_s(\pi_0^{-1}\pi)}.$$

5.3.2 Cayley distance

The Cayley distance $d_C(\pi_1, \pi_2)$ between two permutations π_1 and π_2 measures the minimum number of swaps (not necessarily adjacent) that are needed to transform π_1 into π_2. A concept closely related to the Cayley distance is the number of cycles in the permutations. A cycle is a subset $\{i_1, i_2, ..., i_r\}$ of the set of the items of the permutation such that

$$\pi(i_1) = i_2 \ , \ \pi(i_2) = i_3 \ , \ \dots \ , \ \pi(i_r) = i_1.$$

As previously mentioned, when we calculate $d_C(\pi, e)$ we should count the number of swaps to transform π into e. Note that, in this case, every swap involves two items of the same cycle. So, the minimum number of swaps needed to sort the r items of the same cycle is $r - 1$. This means that the Cayley distance between π and e is n minus the number of cycles.

Taking this definition into account, we can decompose the Cayley distance between two permutations π_1 and π_2:

$$d_C(\pi_1, \pi_2) = d_C(\pi_2^{-1}\pi_1, e) = \sum_{s=1}^{n-1} X_s(\pi_2^{-1}\pi_1, e) \tag{5.3}$$

where

$$X_s(\pi, e) = \begin{cases} 0 \text{ if } s \text{ is the largest item in its cycle in } \pi \\ \\ 1 \text{ otherwise} \end{cases}$$

In this case, we also define $X_s(\pi, e)$ with respect to the identity permutation e. So, similarly to the case of the Kendall-tau distance, we also simplify the notation using $X_s(\pi)$.

For instance, considering again

$$\pi_1 = (4132) \quad \text{ and } \quad \pi_2 = (1342),$$

we have:

$$d_C(\pi_1, \pi_2) = d_C(\pi_2^{-1}\pi_1, e) = \sum_{s=1}^{n-1} X_s(\pi_2^{-1}\pi_1) = 1 + 1 + 0 = 2.$$

The minimum number of cycles in a permutation is 1. This means that the maximum distance between two permutations that can be given by this metric is $n - 1$.

The GM distribution uses a different spread parameter θ^s for each X_s seen in (5.3). Considering the vector of spread parameters $\boldsymbol{\theta} = (\theta^1, \cdot\cdot\cdot, \theta^{n-1}) \in (\mathbb{R}^+)^{n-1}$, a closed form for the normalization term $Z(\boldsymbol{\theta})$ is also found [54]:

$$Z(\boldsymbol{\theta}) = \prod_{s=1}^{n-1} (1 + (n-s)e^{-\theta^s}).$$

Finally, the probability assigned to each $\pi \in \Omega$ under the Cayley metric is defined by:

$$p(\pi|\pi_0, \boldsymbol{\theta}) = \frac{1}{\prod\limits_{s=1}^{n-1}(1+(n-s)e^{-\theta^s})} e^{-\sum\limits_{s=1}^{n-1} \theta^s X_s(\pi_0^{-1}\pi)}.$$

5.4 The generator of instances: Overview

In [37] the authors proposed a generator of instances of optimization problems in the continuous domain based on a mixture of Gaussian distributions where each local optimum of the landscape corresponds with the mean of a Gaussian distribution. The generated instances are such that the function value of a solution is defined by the maximum value that any of the Gaussian components associates to that solution. Inspired by that work, we present a generator of instances of permutation-based COPs. The generator is founded on a mixture of GM distributions.

The general idea of our approach to generate an instance is to choose m consensus permutations $\{\pi_1, \pi_2, \dots, \pi_m\}$, and m spread vectors of parameters $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_m\}$ to create m GM distributions with them:

$$\begin{gathered} p_1(\pi|\pi_1, \boldsymbol{\theta}_1) \\ p_2(\pi|\pi_2, \boldsymbol{\theta}_2) \\ \vdots \\ p_m(\pi|\pi_m, \boldsymbol{\theta}_m) \end{gathered}$$

where

$$p_i(\pi|\pi_i, \boldsymbol{\theta}_i) = p_i(\pi|\pi_i, \theta_i^1, \theta_i^2, ..., \theta_i^{n-1}) = \frac{e^{-\sum\limits_{s=1}^{n-1} \theta_i^s d_s(\pi, \pi_i)}}{Z(\boldsymbol{\theta}_i)}.$$

We also consider weights $\{w_1, w_2, \dots, w_m\}$, $(w_i > 0, \forall i)$ associated to each of the GM distributions. From now on, we will assume that we generate instances of maximization problems. In this sense, the objective function value of a solution (permutation) of the instance is defined as the maximum value reached by all the GM distributions multiplied by the corresponding associated weight:

$$f(\pi) = \max_{1 \le i \le m} \{w_i p_i(\pi|\pi_i, \theta_i^1, \theta_i^2, \dots, \theta_i^{n-1})\}. \tag{5.4}$$

Thus, this model has $(n+1)m$ parameters: $(n-1)m$ spread parameters, m consensus permutations, and m weights. Notice that we need to evaluate m GM models in order to assign an objective function value to each solution of the search space.

5.5 The generator of instances: Particularities

Our aim is to provide a generator able to create instances with different properties. In order to do that, we should tune the parameters of our generator appropriately. This is done by following two complementary tasks; one quantitative and one qualitative. In the quantitative approach we add constraints to the parameters in order to have a predefined number of local optima. These constraints are linear in the weights of the GM components that define the instance. The qualitative approach consists of the definition of a linear function on the weights that tries to promote certain characteristics of the instance such as the relative size of the attraction basin of the global optimum versus the local optima. Different linear functions can create different types of instances.

As a result of this process, each instance of the generator is created by solving a linear programming problem in the weights.

5.5.1 Controlling the local optima in the instance

In this section, we study the constraints we should add in order to ensure that all the consensus permutations of the GM models are the local optima in the generated instance. These local optima are defined when considering a neighborhood determined by the solutions at distance one. Logically, the particular distance used coincides with that used in the GM models (Kendall-tau or Cayley).

Notice that by our definition of an instance, no other permutation of the search space, apart from the consensus permutations, can be a local optimum (local maximum). This means that with our generator we know that the number of local optima is always smaller or equal to the number m of GM components. However, it is possible that a consensus permutation is buried by a GM component centered at any other permutation, and thus, this permutation will not be a local optimum. Hence, the key point is to ensure that all the consensus permutations are, indeed, local optima for our instance. In this way, we will be able to create instances with exactly m local optima.

In what follows, we will assume that it is not possible to find two local optima with the same objective function value. Therefore, the distance between two local optima needs to be higher than one, so the consensus permutations need to satisfy:

$$d(\pi_i, \pi_j) \geq 2, \forall i \neq j. \tag{5.5}$$

We remark that, although we work under this assumption, the generator would be able to face with a situation in which $f(\pi_i) = f(\pi_j)$, $i \neq j$.

By the definition of local maximum, the following constraint has to be fulfilled for π_i to be a local optimum:

$$f(\pi_i) > f(\pi), \;\; \forall \pi \in \Omega \;\; s.t. \;\; d(\pi, \pi_i) = 1. \tag{5.6}$$

The following Lemma 1 shows a necessary condition for a consensus permutation to be a local optimum. In Theorem 1 we give the necessary and sufficient condition in the parameters to guarantee that all consensus permutations are local optima in the instance.

Lemma 1. *Let's consider an instance I created with our generator and let π_i be the consensus permutation of the i-th GM model used in it. If π_i is a local optimum in the generated instance, that is:*

$$f(\pi_i) > f(\pi), \ \forall \pi \in \Omega \ \textit{ such that } \ d(\pi, \pi_i) = 1,$$

then the objective function value of π_i is reached by the i-th GM model, i.e.:

$$\begin{aligned} f(\pi_i) &= \max_{1 \le j \le m} \{w_j p_j(\pi_i | \pi_j, \boldsymbol{\theta}_j)\} = w_i p_i(\pi_i | \pi_i, \boldsymbol{\theta}_i) \\ &= w_i \frac{e^{-\sum\limits_{s=1}^{n-1} \theta_i^s d_s(\pi_i, \pi_i)}}{Z(\boldsymbol{\theta}_i)} = \frac{w_i}{Z(\boldsymbol{\theta}_i)}. \end{aligned} \tag{5.7}$$

Proof. Let's suppose that (5.7) is false, that is: $\exists k \neq i$ such that

$$\begin{aligned} f(\pi_i) &= \max_{1 \le j \le m} \{w_j p_j(\pi_i | \pi_j, \boldsymbol{\theta}_j)\} \\ &= w_k p_k(\pi_i | \pi_k, \boldsymbol{\theta}_k) = w_k \frac{e^{-\sum\limits_{s=1}^{n-1} \theta_k^s d_s(\pi_i, \pi_k)}}{Z(\boldsymbol{\theta}_k)}. \end{aligned}$$

Then, the objective function value of the first permutation found in the shortest path between π_i and π_k, that is, the permutation π such that $d(\pi, \pi_i) = 1$ and $d(\pi, \pi_k) = d(\pi_i, \pi_k) - 1$, is:

$$\begin{aligned} f(\pi) &= \max_{1 \le j \le m} \{w_j p_j(\pi | \pi_j, \boldsymbol{\theta}_j)\} \\ &\ge w_k p_k(\pi | \pi_k, \boldsymbol{\theta}_k) = w_k \frac{e^{-\sum\limits_{s=1}^{n-1} \theta_k^s d_s(\pi, \pi_k)}}{Z(\boldsymbol{\theta}_k)}. \end{aligned}$$

Notice that

$$d(\pi, \pi_k) = \sum_{s=1}^{n-1} d_s(\pi, \pi_k) < d(\pi_i, \pi_k) = \sum_{s=1}^{n-1} d_s(\pi_i, \pi_k),$$

in fact, $d(\pi, \pi_k) = d(\pi_i, \pi_k) - 1$, so that the elements involved in the decomposition of the distance between π and π_k need to be equal to the elements of the decomposition of the distance between π_i and π_k, with the exception of one that has to be one unit lower. This is:

$$\begin{aligned} d_s(\pi, \pi_k) &= d_s(\pi_i, \pi_k), \forall s \neq t \\ d_t(\pi, \pi_k) &= d_t(\pi_i, \pi_k) - 1. \end{aligned}$$

So then,

$$\sum_{s=1}^{n-1} \theta_k^s d_s(\pi, \pi_k) = \sum_{s \neq t} \theta_k^s d_s(\pi_i, \pi_k) + \theta_k^t [d_t(\pi_i, \pi_k) - 1] < \sum_{s=1}^{n-1} \theta_k^s d_s(\pi_i, \pi_k)$$

$$\Rightarrow e^{-\sum_{s=1}^{n-1} \theta_k^s d_s(\pi, \pi_k)} > e^{-\sum_{s=1}^{n-1} \theta_k^s d_s(\pi_i, \pi_k)},$$

and therefore

$$w_k \frac{e^{-\sum_{s=1}^{n-1} \theta_k^s d_s(\pi, \pi_k)}}{Z(\boldsymbol{\theta}_k)} > w_k \frac{e^{-\sum_{s=1}^{n-1} \theta_k^s d_s(\pi_i, \pi_k)}}{Z(\boldsymbol{\theta}_k)}.$$

Thus, we have a permutation π such that $d(\pi, \pi_i) = 1$ and $f(\pi) > f(\pi_i)$. This proves that if (5.7) is not fulfilled, π_i is not a local optimum.

□

Theorem 1. *A consensus permutation π_i of any of the GM models involved in our generator is a local optimum in the generated instance if and only if:*

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi, \pi_j)}, \forall j \neq i, \tag{5.8}$$

$\forall \pi$ such that $d(\pi, \pi_i) = 1$.

Proof. The definition of local optimum is given by (5.6), so that: $f(\pi_i) > f(\pi), \forall \pi \in \Omega$ s.t. $d(\pi, \pi_i) = 1$. By lemma 1, $f(\pi_i) = \frac{w_i}{Z(\theta_i)}$, and therefore we can rewrite (5.6) in the following form:

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \max_{1 \leq j \leq m} \left\{ \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi, \pi_j)} \right\},$$

$\forall \pi$ *s.t.* $d(\pi, \pi_i) = 1$.
Obviously, this is fulfilled for $j = i$, so that this is equivalent to:

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi, \pi_j)}, \forall j \neq i,$$

$\forall \pi$ *s.t.* $d(\pi, \pi_i) = 1$. Therefore, (5.8) is fulfilled.

Let's suppose now that the constraints in (5.8) are satisfied, we will prove that, then, π_i is a local optimum. The definition of objective function value of π_i in our generator is the following:

$$f(\pi_i) = \max_{1 \leq j \leq m} \left\{ \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi_i, \pi_j)} \right\}.$$

We can rewrite this expression, distinguishing between $j = i$ and $j \neq i$, as:

$$f(\pi_i) = \max_{j \neq i} \left\{ \frac{w_i}{Z(\boldsymbol{\theta}_i)}, \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi_i, \pi_j)} \right\}$$

However, we know by (5.8) that

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi, \pi_j)}, \forall \pi \; s.t. \; d(\pi, \pi_i) = 1$$

If this is fulfilled for all π such that $d(\pi, \pi_i) = 1$, specifically, this is fulfilled for the first permutation π' found in the shortest path between π_i and π_j, such that $d(\pi', \pi_i) = 1$ and $d(\pi', \pi_j) = d(\pi_i, \pi_j) - 1$. Reasoning as in the proof of the Lemma 1, the elements involved in the decomposition of the distance between π' and π_j are:

$$\begin{aligned} d_s(\pi', \pi_j) &= d_s(\pi_i, \pi_j), \forall s \neq t \\ d_t(\pi', \pi_j) &= d_t(\pi_i, \pi_j) - 1 \end{aligned}$$

and therefore

$$\sum_{s=1}^{n-1} \theta_j^s d_s(\pi', \pi_j) < \sum_{s=1}^{n-1} \theta_j^s d_s(\pi_i, \pi_j) \Rightarrow e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi', \pi_j)} > e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi_i, \pi_j)}.$$

So, (5.8) implies $\forall \pi \; s.t. \; d(\pi, \pi_i) = 1$:

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi_i, \pi_j)} \Rightarrow f(\pi_i) = \frac{w_i}{Z(\boldsymbol{\theta}_i)} > f(\pi).$$

□

In conclusion, if the parameters fulfill the restrictions in (5.8), we know that all the consensus permutations of the GM models of our generator are local optima in the instance. By fixing the values of θ_j^s in the inequalities in (5.8), the restrictions are linear in the weights, and therefore they can be solved with a linear programing problem. However, we can find inconveniences when working with the restrictions in (5.8).

On the one hand, we could have numerical problems. For example, for high permutation sizes n, the values $e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi, \pi_j)}$ will be close to 0 (as the distances could be considerably high), and, due to precision limits, when handling such values in the resolution of the linear programming problem, the results could be stored as 0.00. Therefore, the desired restrictions are not fulfilled and finally, the consensus permutations are not local optima in the instance. On the other hand, the number of constraints needed to be fulfilled is considerably high: $m(m-1)(n-1)$ and $m(m-1)n(n-1)/2$, in the case of the Kendall-tau and the Cayley distances, respectively.

With the aim of avoiding the numerical instability of (5.8) and in order to reduce the number of restrictions, we show in the following Theorem 2 sufficient conditions to fulfill (5.8). In fact, we find a significant decrease in the number of restrictions, as they are a total of m, independently of the distance used. This reduction in the number of restrictions does not imply, however, limitations in the flexibility of the generator to create different types of instances.

Theorem 2. *Let $\{w_1, \ldots, w_m\}$, $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m\}$ and $\{\pi_1, \ldots, \pi_m\}$ be the weight values, the spread parameters and the consensus permutations, respectively, of the m GM components used in our generator to create an instance I. If the following constraints are fulfilled:*

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_{i+1}}{Z(\boldsymbol{\theta}_{i+1})}, \quad i = 1, \ldots, m-1 \tag{5.9}$$

$$\left[2 - e^{-\left(\min\limits_{j,s}\{\theta_j^s\}\right)}\right] \frac{w_m}{Z(\boldsymbol{\theta}_m)} > \frac{w_1}{Z(\boldsymbol{\theta}_1)} \tag{5.10}$$

then, the restrictions in (5.8) are satisfied.

Proof. From (5.9) we have:

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)}, \forall i < j$$

and obviously, as $\frac{w_j}{Z(\boldsymbol{\theta}_j)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum_{s=1}^{n-1} \theta_j^s d_s(\pi, \pi_j)}$, $\forall \pi$, then inequalities in (5.8) when $i < j$ are fulfilled. Thus, we just have to prove that (5.9) and (5.10) imply (5.8) when $i > j$.

We can rewrite (5.10) in the following form:

$$\left[1 - e^{-\left(\min\limits_{j,s}\{\theta_j^s\}\right)}\right] \frac{w_m}{Z(\boldsymbol{\theta}_m)} > \frac{w_1}{Z(\boldsymbol{\theta}_1)} - \frac{w_m}{Z(\boldsymbol{\theta}_m)} \tag{5.11}$$

As it is known by (5.9):

$$\frac{w_m}{Z(\boldsymbol{\theta}_m)} \leq \frac{w_j}{Z(\boldsymbol{\theta}_j)}, \forall j.$$

So, (5.11) implies:

$$\left[1 - e^{-\left(\min\limits_{j,s}\{\theta_j^k\}\right)}\right] \frac{w_j}{Z(\boldsymbol{\theta}_j)} > \frac{w_1}{Z(\boldsymbol{\theta}_1)} - \frac{w_m}{Z(\boldsymbol{\theta}_m)}, \forall j$$

Moreover,

$$\frac{w_1}{Z(\boldsymbol{\theta}_1)} - \frac{w_m}{Z(\boldsymbol{\theta}_m)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} - \frac{w_i}{Z(\boldsymbol{\theta}_i)}, \forall i > j$$

so that,

$$\left[1 - e^{-\left(\min\limits_{j,s}\{\theta_j^s\}\right)}\right] \frac{w_j}{Z(\boldsymbol{\theta}_j)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} - \frac{w_i}{Z(\boldsymbol{\theta}_i)}, \forall i > j$$

and thus

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} - \left[1 - e^{-\left(\min\limits_{j,s}\{\theta_j^s\}\right)}\right] \frac{w_j}{Z(\boldsymbol{\theta}_j)}$$

$$\Rightarrow \frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\left(\min\limits_{j,s}\{\theta_j^s\}\right)} \tag{5.12}$$

Notice that as $d(\pi, \pi_j) \geq 1, \forall \pi \neq \pi_j$:

$$\min_{j,s}\{\theta_j^s\} < \min_{j,s}\{\theta_j^s\} d(\pi, \pi_j) = \min_{j,s}\{\theta_j^s\} \sum_{s=1}^{n-1} d_s(\pi, \pi_j)$$
$$< \sum_{s=1}^{n-1} [\theta_j^s d_s(\pi, \pi_j)].$$

Finally, inequality (5.12) implies

$$\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_j}{Z(\boldsymbol{\theta}_j)} e^{-\sum\limits_{s=1}^{n-1}[\theta_j^s d_s(\pi, \pi_j)]}.$$

□

In summary, if restrictions (5.9) and (5.10) are fulfilled, we ensure that the consensus permutations of the GM distributions are the local optima of the instance when considering the neighborhood defined by the distance used in the probabilistic models. Note that, by the Lemma 1, these constraints also imply that the objective function value of a consensus permutation is reached by the GM component centered at itself: $f(\pi_i) = \frac{w_i}{Z(\boldsymbol{\theta}_i)}$. Thus, the restrictions in (5.9) mean that: $f(\pi_i) > f(\pi_{i+1})$. So, π_1 is the local optimum with the highest objective function value, that is, the global optimum. Although the restrictions in (5.9) impose an order in the local optima regarding their objective function value, this can be assumed without loss of generality. That is, one can previously sort the consensus permutations in the desired order.

5.5.2 The linear programming problem

Once the number of local optima has been established, in order to customize the properties of an instance, we use a linear function in the weights associated to the GM models. This function carries out qualitative properties of the instance. In the following section we use three different functions to exemplify three different kinds of landscapes regarding the sizes of the attraction basins of the local optima. Of course, the careful choice of the location of the different local optima (consensus permutations), as well as the choice of the spread parameters, help in the generation of the instances with the requested properties.

In general terms, the generator of instances of COPs based on permutations can be described as a 5-tuple $(n, m, \Sigma, \Theta, G)$, where:

- $n \in \mathbb{N}$ is the size of the permutation.
- $m \in \mathbb{N}$ is the number of GM functions (number of local optima of the instance).
- $\Sigma = \{\pi_1, \pi_2, \ldots, \pi_m\}$ is the set of consensus permutations of size n.
- $\Theta = \begin{pmatrix} \theta_1^1 & \cdots & \theta_1^{n-1} \\ \cdots\cdots & \cdots & \cdots \\ \theta_m^1 & \cdots & \theta_m^{n-1} \end{pmatrix} \in (\mathbb{R}^+)^{m\times(n-1)}$

  is the matrix with the spread parameters of the m GM distributions.
- G is the linear objective function of the linear programming problem.

See Algorithm 4 to observe the steps that the generator follows. Notice that in the linear programming problem (step 5 in the algorithm), we add the constraint $w_m > 0$, just to force the weights to be positive: if $w_m > 0$, then $w_i > 0, \forall i$, as (5.9) implies $\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_m}{Z(\boldsymbol{\theta}_m)}$. We also force $w_1 < k \in \mathbb{R}^+$, so as to upper bound the values of the weights.

If required by the user, the framework can be adapted to generate instances with two or more local optima with the same fitness value (for example, to generate multimodal instances). For this to happen, equality $\frac{w_i}{Z(\boldsymbol{\theta}_i)} = \frac{w_{i+1}}{Z(\boldsymbol{\theta}_{i+1})}$ should be used instead of $\frac{w_i}{Z(\boldsymbol{\theta}_i)} > \frac{w_{i+1}}{Z(\boldsymbol{\theta}_{i+1})}$. Nevertheless, note that, as defined previously, we do not consider the possibility of different local optima to be neighbors (which could cause flat regions). The framework could be adapted to that scenario by removing restriction (5.5), but this change would also require to redefine the concept of local optimum, using for example the definition given in [85]. However, this change could have additional and unknown consequences, and thus, a deeper study should be carried out depending on the definition of local optimum considered.

We would like to remark that the $\boldsymbol{\theta}$ values should be chosen taking into account the type of distance used and the permutation size n, in order to avoid numerical problems when calculating the normalization terms $Z(\boldsymbol{\theta})$. We suggest using these values in the intervals:

$$[ln(\frac{n-1}{2}), 3ln(n-1)] \quad \text{for the Kendall-tau distance}$$

and

$$[2ln(\frac{n-1}{3}), 6ln(n-1)] \quad \text{for the Cayley distance.}$$

The source code (in R-project) of the generator is available in the website[1].

Algorithm 4 Algorithm to generate instances of permutation-based Combinatorial Optimization Problems

1. Set n
2. Set m
3. Choose the consensus permutations:

$$\pi_1, \pi_2, ..., \pi_m, \text{ such that: } d(\pi_i, \pi_j) \geq 2, \forall i \neq j.$$

4. Choose the spread parameters:

$$\Theta = \begin{pmatrix} \theta_1^1 & \cdots & \theta_1^{n-1} \\ \cdots & \cdots & \cdots \\ \theta_m^1 & \cdots & \theta_m^{n-1} \end{pmatrix}$$

5. Solve the linear programming problem in the weights $\{w_1, \ldots, w_m\}$:

$$min/max\{G(w_1, ..., w_m)\}$$

subject to

$$\begin{aligned} \frac{w_i}{Z(\boldsymbol{\theta}_i)} &> \frac{w_{i+1}}{Z(\boldsymbol{\theta}_{i+1})}, \quad i = 1, \ldots, m-1 \\ \frac{w_1}{Z(\boldsymbol{\theta}_1)} &< \left[2 - e^{-\left(\min\limits_{i,s}\{\theta_i^s\}\right)}\right] \frac{w_m}{Z(\boldsymbol{\theta}_m)} \\ w_m &> 0 \\ w_1 &< k \quad (k \in \mathbb{R}^+) \end{aligned}$$

6. $\forall \pi \in \Omega$ define the objective function value as:

$$f(\pi) = \max_{1 \leq i \leq m} \left\{ \frac{w_i}{Z(\boldsymbol{\theta}_i)} e^{-\sum\limits_{s=1}^{n-1} \theta_i^s d_s(\pi, \pi_i)} \right\}$$

[1] http://www.sc.ehu.es/ccwbayes/members/leticia/GeneratorOfInstances/code.html

5.5.3 Complexity of the generator

We can distinguish two kinds of complexity associated with the generator: (i) the computational complexity of generating a particular instance, and (ii) the cost of evaluating the objective function value of a solution in the generated instance.

The first one is dominated by the solution of the linear programming problem. In the literature, we can find powerful algorithms whose cost, in the worst case, depends on the number of variables, which in our problem is m: $\{w_1, \dots, w_m\}$.

Given an output instance of the generator, when assigning the objective function value to any solution of the search space (ii), we should evaluate in the worst case m components $w_i p_i(\pi|\pi_i, \boldsymbol{\theta}_i)$ in order to look for the highest value. Each of these evaluations implies the calculation of the $n-1$ terms of the decomposition of the distance (equalities (5.2) and (5.3)). In the case of the Kendall-tau distance, the cost of calculating these terms is $O(n^2)$, whereas for the Cayley distance it is $O(n)$. Therefore, in the worst case, the cost in the evaluation of each solution is $O(mn^2)$ and $O(mn)$ for the Kendall-tau and the Cayley distances, respectively. Commonly, in real instances of COPs, we find that the number of local optima is considerably higher than the size of the permutations [47]. So that we will be willing to generate instances with $m >> n$. Thus, the cost of evaluating the solutions is principally conditioned by the number of GM components used in the generator.

5.6 Creating Instances Using our Generator

In this section we present three examples of how to generate instances with different properties. These three types of instances are associated with three linear functions and three careful choices of the π_i and θ_i^s, $\forall i, s$. Basically, we design instances for three different scenarios.

In the first scenario, we try to generate easy to optimize instances. In this sense, our goal is to make the attraction basin of the global optimum π_1 as large as possible. Our second scenario contemplates difficult instances and the function we optimize tries to make the attraction basin of the global optimum as small as possible. Our last setting generates instances in the middle of the two previous examples, where we attempt to have all the attraction basins of the local optima (including the global optimum) of similar sizes. Obviously, these are just some illustrative examples but, by properly choosing the linear function and tuning the rest of parameters, the generator would be able to provide other kinds of instances.

5.6.1 Easy instances: Global optimum with a large attraction basin

In this example, we tune the parameters with the aim of obtaining an instance with a large attraction basin for the global optimum. As this scenario describes a general situation, we do not refer to any specific distance. We assume that $\{\pi_1, \pi_2, \ldots, \pi_m\}$ are the consensus permutations in our generator, so they are the local optima in the instance, where π_1 is the global optimum.

First, we choose the linear function to optimize in the linear programming problem. In this scenario, we propose maximizing the difference between the objective function value of π_1 (global optimum) and the objective function value of π_2 (local optimum with the highest value after π_1):

$$G_{\mathrm{Max}GO} = max \left\{ \frac{w_1}{Z(\boldsymbol{\theta}_1)} - \frac{w_2}{Z(\boldsymbol{\theta}_2)} \right\}.$$

By means of this linear function, we promote that those permutations π that are close to π_1 will receive an objective function value given by the first GM component, i.e.:

$$f(\pi) = \max_{1 \le i \le m} \{w_i p_i(\pi|\pi_i, \boldsymbol{\theta}_i)\} = w_1 p_1(\pi|\pi_1, \boldsymbol{\theta}_1).$$

Thus, with the help of the rest of the parameters, we can easily force as many solutions as possible to be in the attraction basin of π_1.

The local optima can be chosen paying special attention to the distances among them. For example, it is recommendable to choose π_1 far from the rest of the local optima, in order to contribute to obtaining a large attraction basin for it. In doing this, we provoke a large number of solutions appear closer to π_1 than to the rest of local optima.

The parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_m$ are also essential when trying to control the properties of the generated instance. In order to provide an easy and intuitive example, let's assume that we are in the situation $\theta_i^j = \theta_i^k = \theta_i, \forall j \neq k$. So, we are considering the Mallows model. Under this assumption, we are assigning the same objective function value to all the solutions that are at the same distance from the consensus permutation. This situation gives us intuition about how to choose the spread parameters in order to model the different attraction basins of the local optima. If we want the global optimum to have a large attraction basin, we should choose $\theta_1 << \theta_i, i \geq 2$, because, as we saw in Section 5.2.1, the larger the value of θ the more peaked the Mallows function. Hence, with a small θ those solutions close to π_1 have an objective function value similar to π_1, and therefore the possibilities of having them in the attraction basin of the global optimum are very high.

5.6.2 Hard instances: Global optimum with a small attraction basin

We propose, in this scenario, to minimize the difference between the objective function value of the global optimum π_1 and its neighbors. When looking for the objective function value of the neighbors we need to calculate the values of the elements involved in the decomposition of the distance between those neighbors and all the local optima of the instance: $\{\pi_1, \pi_2, \ldots, \pi_m\}$. However, this calculation would imply a high computational cost. Therefore, we can simplify this idea taking into account the following function to minimize:

$$G_{\mathrm{Min}GO} = min\left\{ \frac{w_1}{Z(\boldsymbol{\theta}_1)} - \frac{1}{m-1}\sum_{i=2}^{m} \frac{w_i}{Z(\boldsymbol{\theta}_i)} e^{-\max_j\{\theta_i^j\}(d(\pi_1,\pi_i)+1)} \right\}.$$

Notice that the second element in the subtraction is the average of lower bounds for the values assigned to those neighbor solutions given by the GM components centered at the local optima which are different from the global optimum. Thus, it is a lower bound for the objective function values of the neighbors of π_1, and therefore, minimizing this difference, we would be minimizing the desired difference. The aim of choosing this linear function is to help the neighbor solutions of π_1 to have their objective function value assigned by the GM component centered at other local optimum different to π_1, and thus, to belong to a different attraction basin.

For this scenario, π_1 can be chosen as a consensus permutation that is close to, at least, one local optimum. For example, this local optimum could be π_2, i.e., the local optimum with the highest objective function value after π_1. If π_2 is near π_1, there are less possibilities of having a large number of solutions in the attraction basin of π_1, because we can provoke (also taking into account the rest of parameters) that the solutions belong to the attraction basin of π_2.

As we explained in the example above, choosing $\theta_i^j = \theta_i^k, \forall j \neq k$ we provide a more intuitive example. So, under this context, by denoting θ_i as the spread parameter of the i-th Mallows model ($\theta_i = \theta_i^1 = \ldots = \theta_i^{n-1}$), the larger the value of θ_1 in comparison to the rest of θ_i, $i > 1$, the smaller the size of the attraction basin of the global optimum. Moreover, as an example of a more difficult scenario, we try to obtain the size of the attraction basin of π_2 as large as possible by choosing $\theta_2 << \theta_i$ $(i \neq 2)$.

5.6.3 Local optima with similar sizes of attraction basins

In this last example, we are interested in creating instances where all the local optima (included the global optimum) have attraction basins similar in size.

Assuming that π_1 is the global optimum, and π_m is the local optimum with the lowest objective function value, the considered function to minimize in this case, is the difference between the objective function values of these two permutations (and implicitly, minimize the difference between the objective function values of all the local optima):

$$G_{\mathrm{Sim}AB} = \min\left\{ \frac{w_1}{Z(\boldsymbol{\theta}_1)} - \frac{w_m}{Z(\boldsymbol{\theta}_m)} \right\}.$$

In this sense, the landscape created with this linear function is flatter than in the other two examples.

Taking into account that we want to obtain local optima with attraction basins of similar sizes, these solutions should be uniformly distributed in the search space. So that we could choose them in order for them to be in a situation as similar to the following one as possible: $d(\pi_i, \pi_j) \approx d, \forall i, j,$ where d is the largest possible value.

The values of all θ_i^j should be similar for all the elements of the decomposition of the distance, and also similar for all the GM components. That is, with $\theta_i^s = \theta, \forall i, s$, the Mallows models centered at all the consensus permutations have the same shape. This implies that we do not force any of the local optima to have more solutions in their attraction basins than the rest of the local optima.

5.7 Experiments

A generator of instances of COPs is a useful tool when analyzing, comparing, and evaluating the behavior of different optimization algorithms. In this section, we show the influence of the input parameters in the resultant instance, and present several use cases of our generator with the aim of demonstrating its importance and applicability.

First, we want to show that the generator is highly flexible. For this purpose, we find it interesting to check if we are able to generate instances similar to those available in well-known benchmarks. Thus, the first part of the experimentation is devoted to this goal, working with small problem sizes. Secondly, we demonstrate that, by tuning the input parameters, the user can create different instances, placing emphasis on coarse grain characteristics (such as the attraction basin sizes of the local optima) as well as on fine grain characteristics (distribution of the local optima). This is a highlighting characteristic of our generator, as it allows to conduct a deeper analysis of the behavior of optimization algorithms.

5.7.1 Flexibility of the generator

In order to create an artificial instance similar to an instance that can be found in well-known benchmarks (from now on we will refer to them as benchmark instances), we carry out the following steps:

i) Calculate, for the benchmark instance, the local optima and the sizes of their attraction basins.
ii) Use these values to fix three input parameters of our generator: n (permutation size), m (number of local optima) and Σ (set of consensus permutations).
iii) Choose one of the linear functions to optimize from those defined in Section 5.6 ($G_{\mathrm{Max}GO}$, $G_{\mathrm{Min}GO}$ and $G_{\mathrm{Sim}AB}$), taking into account the characteristics of the benchmark instance. Of course, the generator allows the user to define other linear functions.

iv) Carry out a search over the space of Θ parameters to minimize the difference between the artificial instance and the benchmark instance. Note that each time a set of Θ parameters is tested, the linear programming problem needs to be solved.

A key point in the previous process is to define a way to measure the similarity between the artificial and the benchmark instance (step iv). For example, one option could be to compare the objective function values of the solutions, trying to create an artificial instance with the same absolute objective function values as the given benchmark instance. However, most of the local and population-based algorithms only use the relative value of the solutions instead of the exact function values. Therefore, we could rank the solutions according to their objective function value in the artificial and the benchmark instances, comparing both rankings. Nevertheless, considering that we generate the instances with locality criteria in mind, we have used the difference between the attraction basin sizes of the artificial and the benchmark instances as the similarity measure. Although we are aware that this measure is not as appropriate as the ranking, if we find an artificial instance whose attraction basin sizes are identical to those of benchmark instances, then the behavior of a Local Search algorithm would be the same in both instances.

We generate instances with properties similar to those of two well known COPs: the Permutation Flowshop Scheduling Problem and the Linear Ordering Problem. We work with 5 instances of the PFSP obtained from the well-known benchmark proposed by Taillard[2], and 5 instances of the LOP, obtained from the xLOLIB benchmark [76]. The size of the original instances has been reduced to $n = 8$, that is, in the instances of the PFSP, we consider 8 jobs and 5 machines, and in the LOP instances, the size of the matrices is 8x8. The instances used are available in the website[3]. The reason for choosing a small permutation size is to keep, for each candidate Θ (step iv), the cost of calculating the attraction basin sizes of the local optima affordable.

We consider two different neighborhoods: swap (N_S) and adjacent swap (N_A). These neighborhoods can be defined using the Cayley and the Kendall-tau distances, respectively. Particularly, as we defined in Section 1.3, the swap neighborhood considers that two solutions are neighbors if they differ from one swap (Cayley distance is one), whereas they are neighbor solutions under the adjacent swap neighborhood if they differ from one adjacent swap (Kendall-tau distance is one). Using a deterministic greedy local search algorithm, we start from each solution of the search space with the aim of finding the number of local optima and their attraction basin sizes. We denote by $\mathcal{B}(\pi_i)$ the attraction basin of π_i in the benchmark instance, while $\hat{\mathcal{B}}(\pi_i)$ refers

[2] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir /ordonnancement.html
[3] http://www.sc.ehu.es/ccwbayes/members/leticia/GeneratorOfInstances /Instances.html

to the attraction basin of π_i in the artificial instance created with our generator. The error ϵ_i is, therefore, $\epsilon_i = ||\mathcal{B}(\pi_i)| - |\hat{\mathcal{B}}(\pi_i)||$. We remark that, when the local optima of a benchmark instance are calculated using N_S (N_A), the artificial instance is generated using the Cayley distance (Kendall-tau metric) in the GM models.

In order to choose the linear function to optimize in the linear programming problem (step iii), we take into account the proportion of the attraction basin size of the global optimum with respect to the size of the search space: $\frac{|\mathcal{B}(\pi_1)|}{|\Omega|}$. According to preliminary experiments, we choose in our generator one of the objective functions: $G_{\mathrm{Max}GO}$, $G_{\mathrm{Min}GO}$ and $G_{\mathrm{Sim}AB}$ (explained in Section 5.6), if such a proportion is higher than $\frac{0.6}{m}$, lower than $\frac{0.4}{m}$, or between these two values, respectively.

As explained in the last step (iv), we look for the Θ values that generate an instance with similar attraction basin sizes of the local optima. So as to do this, we develop a search (Algorithm 5) in the space of the Θ parameters. For each Θ parameters candidate we solve the linear programming problem, and with the obtained instance we calculate the attraction basin sizes of the local optima. Finally, we calculate the difference between the attraction basin sizes in the generated artificial instance with those of the real instance.

Algorithm 5 Algorithm to adjust θ_i^j in the generator

```
Choose initial values for $\theta_i^j$, $\forall i, j$
$\rho = 0$
**repeat**
    Solve the linear programming problem in $w_1, \ldots, w_m$.
    Apply Algorithm 1 to calculate the attraction basin sizes of the local optima: $|\hat{\mathcal{B}}(\pi_i)|$
    Find the errors $\epsilon_i = ||\mathcal{B}(\pi_i)| - |\hat{\mathcal{B}}(\pi_i)||$
    **for** $i = 1 \rightarrow m$ **do**
        **if** $\epsilon_i > \frac{|\Omega|}{100m}$ **then**
            Update $\theta_i^j$, $\forall j$
        **end if**
    **end for**
    $\rho = \rho + 1$
**until** $\epsilon_i \leq \frac{|\Omega|}{100m}$, $\forall i$ or $\rho = 100$
```

In Tables 5.1 and 5.2 we provide, for PFSP and LOP respectively, the differences between the sizes of the attraction basins of the local optima of each benchmark instance and the instance generated with our model. The first five rows in each table refer to the results when using the N_A neighborhood in the local search and the Kendall-tau distance in our generator. The last five rows show the results for the same five instances when we utilize the N_S neighborhood for solving them and the Cayley distance in our generator. In the first column we provide the number of the instance (1-5), and in the second column we indicate the number of local optima obtained in each case. The third column shows the proportion of solutions of the search space that are in the attraction basin of the global optimum but should not be there, or that should be there but they are not, that is: $\frac{||\mathcal{B}(\pi_1)|-|\hat{\mathcal{B}}(\pi_1)||}{|\Omega|}$. The fourth and the fifth columns indicate the average and the variance of the proportion of solutions of the search space that are not in the corresponding attraction basin. Notice that we divide by $2|\Omega|$, because if a solution is not in its corresponding attraction basin, the error is counted twice: in the attraction basin that it is in and in the one that it should be in.

Table 5.1. Results obtained for the errors in the attraction basin sizes of the local optima for the 5 instances of the PFSP, for the Cayley and the Kendall-tau distances.

	Inst	m	$\frac{\epsilon_1}{\lvert\Omega\rvert}$	$\bar{\epsilon} = \frac{1}{m}\sum_{i=1}^{m}\frac{\epsilon_i}{2\lvert\Omega\rvert}$	$\frac{1}{m-1}\sum_{i=1}^{m}\left(\frac{\epsilon_i}{2\lvert\Omega\rvert}-\bar{\epsilon}\right)^2$
Kendall	1	296	0.007465	0.001312	0.000004
	2	319	0.005903	0.000771	0.000001
	3	424	0.003571	0.000732	0.000000
	4	469	0.008656	0.000575	0.000001
	5	655	0.004439	0.000418	0.000000
Cayley	1	10	0.000273	0.004058	0.000031
	2	12	0.005357	0.007252	0.000073
	3	24	0.000570	0.004384	0.000026
	4	14	0.005456	0.005308	0.000042
	5	22	0.001811	0.003102	0.000006

We observe from Tables 5.1 and 5.2 that the artificial instances generated are almost identical in terms of sizes of attraction basins of the local optima. The average error found in the sizes of the attraction basins of all the local optima is lower than 0.14% in the PFSP instances when using the adjacent neighborhood (Kendall-tau distance), and lower than 0.73% when applying the swap neighborhood (Cayley distance). In the LOP instances, the errors are lower than 0.12% and 0.81%, for the adjacent and the swap neighborhoods (Kendall-tau and Cayley distances), respectively. Moreover, we observe that these errors are uniformly distributed among the attraction basins of the local optima, as the variances are very small: less than $0.73 \cdot 10^{-4}$ in all cases and almost zero in some of them. We pay special attention to the difficulty of finding the global optimum, and thus, to the error found in the sizes of the attraction basins of the global optimum. For the instances of the PFSP solved with the N_A (Kendall-tau distance) the error is smaller than 0.0087,

Table 5.2. Results obtained for the errors in the attraction basin sizes of the local optima for the 5 instances of the LOP, for the Cayley and the Kendall-tau distances.

	Inst	m	$\frac{\epsilon_1}{\lvert\Omega\rvert}$	$\bar{\epsilon} = \frac{1}{m}\sum_{i=1}^{m}\frac{\epsilon_i}{2\lvert\Omega\rvert}$	$\frac{1}{m-1}\sum_{i=1}^{m}\left(\frac{\epsilon_i}{2\lvert\Omega\rvert} - \bar{\epsilon}\right)^2$
Kendall	1	435	0.003423	0.001168	0.000002
	2	720	0.003993	0.000700	0.000001
	3	895	0.003795	0.000536	0.000001
	4	920	0.004886	0.000542	0.000001
	5	3737	0.001265	0.000111	0.000000
Cayley	1	28	0.002307	0.005175	0.000032
	2	27	0.001910	0.008091	0.000037
	3	19	0.002679	0.003836	0.000015
	4	24	0.005704	0.005958	0.000013
	5	319	0.004812	0.000877	0.000001

and for the N_S (Cayley distance) the highest error found is 0.0055. In the LOP instances the maximum errors found are 0.0049 and 0.0057 for the Kendall-tau and the Cayley distances, respectively.

For illustrative purposes, we take one example of each of the problems considered (PFSP and LOP) and we plot the sizes of the attraction basins of the local optima for two pairs of benchmark and artificial instances, when using the Cayley distance (swap neighborhood). The examples for the case of the Kendall-tau are not shown due to the high number of local optima. Particularly, Figure 5.2 shows the fifth instance of the PFSP and Figure 5.3 shows the third instance of the LOP. The remaining figures can be found in the Appendix 9.2. For both figures, in the X axis the local optima are indicated sorted by their objective function value. That is, the local optimum number 1 is the global optimum, and the local optimum number m is the local optimum with the lowest objective function value. As can be observed, the sizes of the attraction basins of the local optima for the benchmark and artificial instances are almost identical. Notice that, when an attraction basin of a local optimum π_i is larger or smaller than the attraction basin of the local optimum π_{i+1} in the benchmark instance, it also happens in almost all of the cases of the artificial instances. Thus, we can conclude that the generator is flexible enough to create instances as complex as the instances found in common benchmark problems.

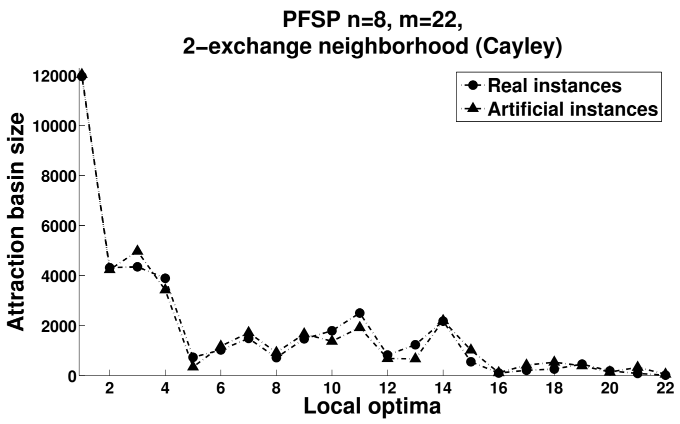


Fig. 5.2. Sizes of the attraction basins of the 22 local optima found in the fifth instance of the PFSP using the swap neighborhood. We represent those sizes found in the benchmark instance with a circle, and the sizes obtained in the generated instance with a triangle.

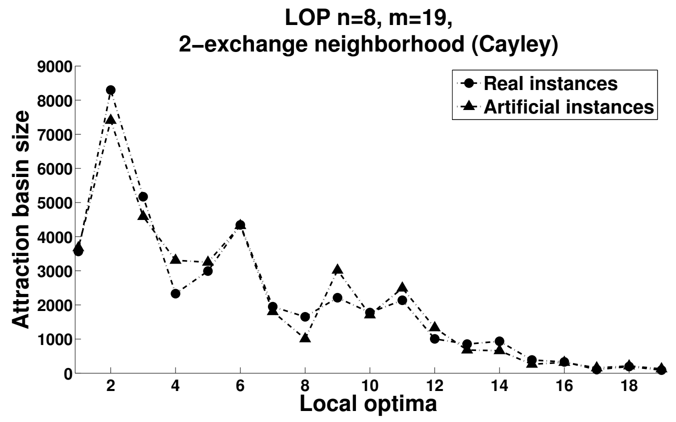


Fig. 5.3. Sizes of the attraction basins of the 19 local optima found in the third instance of the LOP using the swap neighborhood. We represent those sizes found in the benchmark instance with a circle, and the sizes obtained in the generated instance with a triangle.

5.7.2 Tuning the parameters of the generator

We will show that our generator allows the user to control coarse grain features of the instances, such as the sizes of the attraction basins of the local optima, as well as to characterize the instances in a more detailed way. In this section we work with the Mallows (not Generalized) models.

5.7.2.1 Choosing the linear function G to optimize

As described previously, the generator allows the practitioner to define the linear function G to optimize. In this case, without loss of generality, we use the three functions introduced in Section 5.6: G_{MaxGO}, G_{MinGO} and G_{SimAB}. Our goal is to create instances with different attraction basin sizes for the local optima. Based on this feature, one can expect that the easiest instances will be those with a large attraction basin size for the global optimum. Regarding the remaining parameters, we use four permutation sizes $n = \{30, 40, 50, 100\}$, two different numbers of local optima $m = \{10^4, 10^5\}$, and two distances (Kendall-tau and Cayley). The consensus permutations (local optima) Σ are taken uniformly at random. The range of values for θ has been chosen following the recommendations given in Section 5.5.2 to avoid numerical errors. We specify them according to the desired type of instance. For each combination we create 10 instances.

- *MaxGO-Instances*:
  In this set, we want to promote a radical difference between the sizes of the attraction basins of the global optimum and the rest of the local optima, with the aim of obtaining the attraction basin of the global optimum as large as possible. We fix the value of θ_1:
  - $\theta_1 = ln(\frac{n-1}{3})$ for the Kendall-tau distance,
  - $\theta_1 = 2ln(\frac{n-1}{3})$ for the Cayley distance.

  The values of $\theta_i, i \neq 1$ are chosen uniformly at random in the intervals:
  - $I_K = [ln(n-1), 2ln(n-1)]$ for the Kendall-tau distance,
  - $I_C = [3ln(n-1), 4ln(n-1)]$ for the Cayley distance.

  Note that, in this sense, the value of θ_1 is always smaller than the rest of θ_i. The local optima are sorted such that

  $$d(\pi_2, \pi_1) \geq d(\pi_3, \pi_1) \geq ... \geq d(\pi_m, \pi_1) \geq 2.$$

  Therefore, the closest local optimum to the global optimum is the one that has the minimum objective function value: π_m.

- *MinGO-Instances*:
  This set of instances is generated with the objective of having a small attraction basin of the global optimum and a large attraction basin of the closest local optimum. The values of $\theta_i, i \neq \{1, 2\}$, are chosen uniformly at random in the intervals I_K and I_C for the Kendall-tau and the Cayley

distances, respectively, as in the *MaxGO-Instances*. The values for θ_1 and θ_2 are chosen as the highest and the lowest. So, we choose:
– $\theta_1 = 3ln(n-1)$ and $\theta_2 = ln(\frac{n-1}{3})$ for Kendall-tau,
– $\theta_1 = 6ln(n-1)$ and $\theta_2 = 2ln(\frac{n-1}{3})$ for Cayley.
The local optima are sorted such that

$$2 \leq d(\pi_2, \pi_1) \leq d(\pi_3, \pi_1) \leq ... \leq d(\pi_m, \pi_1).$$

- *SimAB-Instances*:
  In this set of instances, the values of $\theta_i, \forall i$, are chosen uniformly at random from the intervals I_K and I_C for the Kendall-tau and the Cayley distances, respectively, as in the previous two examples. Notice that this time, we do not make any distinction between θ_1, θ_2 and the rest of θ_i. The consensus permutations are also taken at random without taking into account the distances between them.

In order to confirm our suspicion, that is, that the difficulty of the problem will be conditioned by the size of the attraction basin of the global optimum, we apply a local search (LS) algorithm, an Estimation of Distribution Algorithm (EDA) and a Genetic Algorithm (GA) to the three sets of instances proposed above. The algorithms are run 20 times for each instance, and we take the best solution reached by each algorithm for each repetition. The stopping criterion in the algorithms is the evaluation of $1000n^2$ solutions. Below, we detail the three algorithms used:

- **LS:** We use a random multi-start hill climbing approach where the best solution found in the neighborhood is chosen at each step. The neighborhoods used are: N_A and N_S for the instances where the Kendal-tau and the Cayley distances, respectively, were used in the Mallows models.
- **EDA:** We use the EDA presented in [16] for solving permutation-based problems. The authors used the Mallows distribution with the Kendall-tau distance as the probability model. As proposed by the authors, the population size is $10n$, and the n best individuals are selected for learning the probability distribution.
- **GA:** We use the GA proposed in [4]. As suggested by the authors, the population size is $20n$, a position based crossover operator (POS) and an insertion mutation operator (ISM) are used, and a tournament selection of size 2.

Tables 5.3, 5.4 and 5.5 show how the LS, the EDA and the GA, respectively, behave according to the type of instance they are applied to. Particularly, the tables show the percentage of the times that the best solution reached by the algorithm is the global optimum π_1, the best local optimum π_2, or any other local optimum. For the case of the EDA and the GA not reaching a local optimum, we have added additional information indicating which is the closest local optimum. With these results, we prove that the *MaxGO-Instances* are, indeed, easy for the LS as the global optimum is reached 100% of the times. On the other hand, we also check that the *MinGO-Instances* are instances where the global optimum is never seen. We find that in the *SimAB-Instances*, it is also difficult to find the global optimum. However, the difference between the *SimAB-Instances* and the *MinGO-Instances* is that in the *MinGO-Instances* there is a local optimum (π_2 in all cases) that is seen in 100% of the runs, whereas in the set of *SimAB-Instances* different local optima are reached.

Table 5.3. Average percentage of the times that the best solution reached by the LS is the global optimum π_1, the local optimum π_2, or other different local optimum.

	Kendall-tau			Cayley		
Best solution	*MaxGO-Inst*	*MinGO-Inst*	*SimAB-Inst*	*MaxGO-Inst*	*MinGO-Inst*	*SimAB-Inst*
π_1	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
π_2	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
π_i, $(i \neq 1, 2)$	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%

Table 5.4. Average percentage of the times that the best solution reached by the EDA is (or is close to) the global optimum π_1, the local optimum π_2, or other different local optimum.

	Kendall-tau			Cayley		
	MaxGO-Inst	*MinGO-Inst*	*SimAB-Inst*	*MaxGO-Inst*	*MinGO-Inst*	*SimAB-Inst*
π_1	77.54%	0.00%	0.00%	0.00%	0.00%	0.00%
π_2	0.00%	77.87%	0.00%	0.00%	0.00%	0.00%
π_i, $(i \neq 1, 2)$	0.00%	0.00%	76.37%	0.00%	0.00%	0.00%
closest π_1	22.46%	0.00%	0.00%	99.50%	0.00%	0.00%
closest π_2	0.00%	22.13%	0.00%	0.00%	99.88%	0.00%
closest π_i $(i \neq 1, 2)$	0.00%	0.00%	23.63%	0.50%	0.12%	100.00%

For the Kendall-tau distance, when applying the EDA (Table 5.4) and the GA (Table 5.5) to the *MaxGO-Instances*, the global optimum is reached, or, if not, the best solution found is always closer to the global optimum than to any other local optimum. In the *MinGO-Instances*, the global optimum is never observed, however the best solution is π_2, or, at least, the closest local optimum to the best solution is π_2. In the *SimAB-Instances*, π_1 and π_2 are never found, and other different local optima, or solutions closer to other lo-

Table 5.5. Average percentage of the times that the best solution reached by the GA is (or is close to) the global optimum π_1, the local optimum π_2, or other different local optimum.

	Kendall-tau			Cayley		
	MaxGO-Inst	*MinGO-Inst*	*SimAB-Inst*	*MaxGO-Inst*	*MinGO-Inst*	*SimAB-Inst*
π_1	40%	0.00%	0.00%	8.00%	0.00%	0.00%
π_2	0.00%	40.00%	0.00%	0.00%	13.00%	0.00%
π_i, $(i \neq 1, 2)$	0.00%	0.00%	31.00%	0.00%	0.00%	20.00%
closest π_1	60%	0.00%	0.00%	92.00%	0.00%	0.00%
closest π_2	0.00%	60.00%	0.00%	0.00%	87.00%	0.00%
closest π_i $(i \neq 1, 2)$	0.00%	0.00%	69.00%	0.00%	0.00%	80.00%

cal optima than to π_1 and π_2, are always seen. Despite for the Kendall-tau (N_A) most of the best solutions found are local optima, in the case of the Cayley distance,the percentage of instances where the solution reached is not a local optimum is really high. Particularly, for the instances generated using the Cayley distance, we find that the best solution reached by the EDA is never a local optimum. This is due to the fact that the probabilistic distribution assumed in the EDA is the Mallows model with the Kendall-tau distance, and therefore it is difficult for this model to reach a local optimum for the swap neighborhood. However, a really high percentage of the times the best solutions found are closer to π_1, π_2 and other different π_i for the *MaxGO-Instances*, the *MinGO-Instances* and the *SimAB-Instances*, respectively. This also happens to the GA when it is applied to the instances generated using the Cayley distance. However, this algorithm is able to find π_1 and π_2 in the *MaxGO-Instances* and the *MinGO-Instances*, respectively, at least a small percentage of the times.

We also record, for each instance and both population-based algorithms, the distance of the best solution found to the global optimum of each run of the algorithms distinguishing among the permutation sizes and the number of local optima. In Table 5.6 we indicate the average distance of the best solution found to the global optimum for the *MaxGO-Instances*, *MinGO-Instances* and *SimAB-Instances*. Each value indicated in the table is the average of the results obtained for the 10 instances generated with the same properties and the 20 repetitions of the algorithms. As expected, the values for the *MaxGO-Instances* are considerably smaller than the values for the *MinGO-Instances* and *SimAB-Instances*. So, as was seen in the previous tables, the best solutions found in the *MaxGO-Instances* are closer to the global optimum than the best solutions obtained in the other two sets of instances. According to these results observed the *MinGO-Instances* and the *SimAB-Instances* are diffi-

Table 5.6. Average distance of the solution found with the EDA and the GA to the global optimum according the type of instance, the number of local optima, the permutation size and the distance considered.

			$n = 30$		$n = 40$		$n = 50$		$n = 100$	
			Kendall	Cayley	Kendall	Cayley	Kendall	Cayley	Kendall	Cayley
MaxGO-Inst	$m = 10^4$	EDA	8.20	15.70	0.00	22.58	0.00	30.70	0.92	75.88
		GA	0.00	0.70	0.00	0.72	0.00	0.68	0.00	0.80
	$m = 10^5$	EDA	146.88	15.32	74.34	22.82	0.00	30.94	0.96	76.00
		GA	0.00	0.62	0.00	0.60	0.00	0.70	0.00	0.70
MinGO-Inst	$m = 10^4$	EDA	123.84	25.00	235.20	35.38	390.34	45.14	1826.32	94.46
		GA	114.10	19.28	235.20	28.86	386.10	38.30	1826.20	86.90
	$m = 10^5$	EDA	178.02	24.00	243.80	34.70	361.22	44.72	1737.40	94.42
		GA	97.70	18.26	210.90	27.38	355.20	36.50	1659.34	85.30
SimAB-Inst	$m = 10^4$	EDA	214.48	26.14	383.74	35.84	595.36	45.46	2399.92	94.88
		GA	216.78	26.22	410.36	35.80	612.34	45.26	2379.90	95.40
	$m = 10^5$	EDA	227.20	25.56	388.62	35.42	594.88	45.70	2479.50	94.68
		GA	212.10	26.26	374.50	35.60	623.80	45.60	2457.86	95.02

cult for the EDA and the GA, as the distance of the best solution to the global optimum in all cases is considerably high. We notice that the distances for the *SimAB-Instances* are higher than those of *MinGO-Instances*. This is due to the fact that, as we saw in Tables 5.4 and 5.5, the best solutions found with both algorithms for the *MinGO-Instances* are closer to the local optimum π_2 than to the rest of local optima. In fact, in most of the cases, the best solution found for these instances is π_2. So, as we create this set of instances denoting by π_2 the closest local optimum to π_1, in Table 5.6 we obtain smaller distances for the *MinGO-Instances* than for the *SimAB-Instances* (where the local optima are chosen randomly).

5.7.2.2 Influence of Σ and Θ

As seen in the previous experiments, changing the function to optimize, together with a careful ranking of randomly located local optima and choice of parameters Θ, allows us to create instances with different qualitative characteristics and hence complexities. However, it is also interesting to know the sensitivity of the generated instances to the input parameters, i.e. the set of permutations Σ, and the set of parameters Θ. It is clear that the location of the local optima can have a high impact on the complexity of an instance. Taking that into account, our generator provides complete freedom on the choice of the location of the local optima. We can devise several ways to do that, such as: choosing random permutations, using some proximity criterion between the global optimum and the local optima, or even creating a matrix with distance constraints and looking for the set of permutations that tries to fulfill such constraints.

The contribution of the set of parameters Θ is not so intuitive. They control the shape of the Mallows models involved in the generator and, thus, they have a big influence on the attraction basin of the local optima. Therefore, we consider it interesting to measure their contribution.

The experiments carried out to measure the sensitivity of the instances to the input parameters are as follows. The permutation size and the number of local optima have been fixed to $n = 30$ and $m = 10^4$, using the Kendall-tau and the Cayley distances in the Mallows models. The linear function considered is G_{SimAB}, as we think it is the least biased to analyze the influence of the input parameters. We have evaluated 11 values for θ_1 for each distance. Nine of these values are the points that divide the intervals I_K and I_C previously defined, in 10 identical subintervals and the other two are the extremes of the intervals. The rest of the spread parameters $\theta_i, i \neq 1$, are chosen uniformly at random in those intervals I_K and I_C.

For the location of the local optima, they have been chosen according to three different configurations:

- 1^{st} configuration: Global optimum surrounded by all the local optima, as close as possible (see Algorithm 6).
- 2^{nd} configuration: All the local optima are close except the global optimum that is as far from them as possible (see Algorithm 7).
- 3^{rd} configuration: All the local optima, including the global optimum, are uniformly spread along the search space (see Algorithm 8).

In summary, we have a total of 66 combinations: 2 (types of distance) x 3 (distribution of Σ) x 11 (values of Θ). We create 10 instances for each possible combination. As the criterion to evaluate the influence of the input parameters we have used the attraction basin size of the global optimum. For each of the generated instances we have estimated this size with the following procedure (note that an exact basin calculation is computationally unfeasible): we run the LS algorithm of the previous section, recording the number of times that π_1 is seen. The proportion of times that the LS reaches π_1 is an estimator of the proportion of its attraction basin size.

Figures 5.4, 5.5 and 5.6 show the results for the 1^{st}, 2^{nd} and 3^{rd} configurations of the local optima, respectively. Each point represents the average value of the attraction basin sizes of π_1 of 10 instances. The figures mainly prove the big influence that the choice of θ_1 has in the attraction basin of π_1. Although the localization of the local optima is relevant, (as can be seen with small values of θ_1) this influence is neglected by the peaky shape imposed in the Mallows model that generates the global optimum for medium to high values of θ_1.

Algorithm 6 Algorithm to choose the global optimum surrounded by all the local optima, as close as possible.

```
1. Choose at random $\pi_1 \in \Omega$
2. $k = 2$
3. $dist = 2$
4. while $k \leq m$ do
5.    $t = 0$
6.    while $t < 50$ do
7.       Choose at random $\pi_k \in \Omega$ such that $d(\pi_k, \pi_1) = dist$
8.       if $\exists i < k$ such that $\pi_k = \pi_i$ then
9.          $t = t + 1$
10.      else
11.         $t = 0$
12.         $k = k + 1$
13.      end if
14.   end while
15.   $dist = dist + 2$
16. end while
```

Algorithm 7 Algorithm to choose all the local optima close except the global optimum that is as far of them as possible.

```
1. Choose at random $\pi_1 \in \Omega$
2. $k = 2$
3. $dist = 2$
4. while $k \leq m - 1$ do
5.    $t = 0$
6.    while $t < 50$ do
7.       Choose at random $\pi_k \in \Omega$ such that $d(\pi_k, \pi_1) = dist$
8.       if $\exists i < k$ such that $\pi_k = \pi_i$ then
9.          $t = t + 1$
10.      else
11.         $t = 0$
12.         $k = k + 1$
13.      end if
14.   end while
15.   $dist = dist + 2$
16. end while
17. Choose $\pi_m \in \Omega$ such that $d(\pi_m, \pi_1) = maximum\_distance$
```

Algorithm 8 Algorithm to choose all the local optima, including the global optimum, uniformly spread along the search space.

```
1. Choose at random $\pi_1 \in \Omega$
2. $k = 2$
3. $dist = 2$
4. while $k \leq m$ do
5.    Choose $\pi_k \neq \pi_i, \forall i < k$, such that $d(\pi_k, \pi_1) = dist$
6.    if $dist + 2 \leq maximum\_distance$ then
7.       $dist = dist + 2$
8.    else
9.       $dist = 2$
10.   end if
11.   $k = k + 1$
12. end while
```

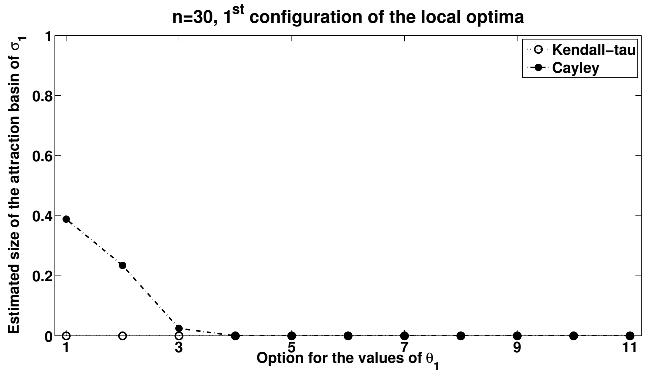


Fig. 5.4. Estimated attraction basin sizes of π_1 for the 1^{st} configuration.

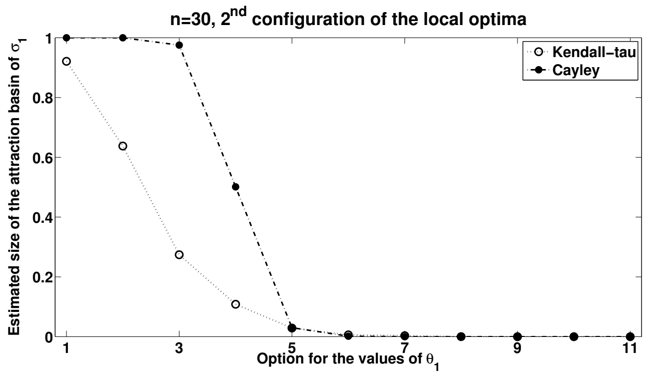


Fig. 5.5. Estimated attraction basin sizes of π_1 for the 2^{nd} configuration.

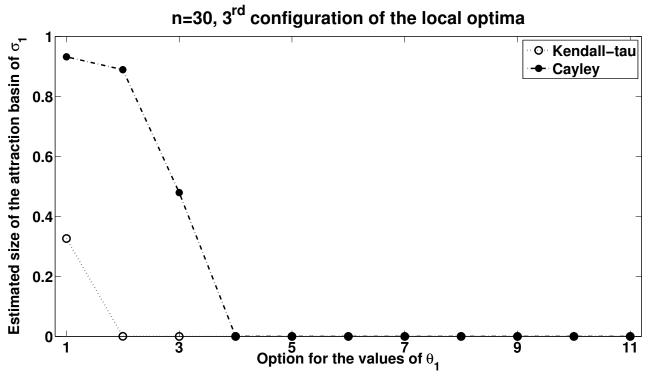


Fig. 5.6. Estimated attraction basin sizes of π_1 for the 3^{rd} configuration.

5.7.2.3 A case study: Discovering differences between the LS, the GA and the EDA

As a case of use of our generator, we show in this section an example on how to utilize the generated instances to discover new facts about metaheuristic algorithms. Particularly, we have tested the previously defined LS, EDA and GA in all the instances generated in the previous section. Each algorithm has been run 20 times in each instance.

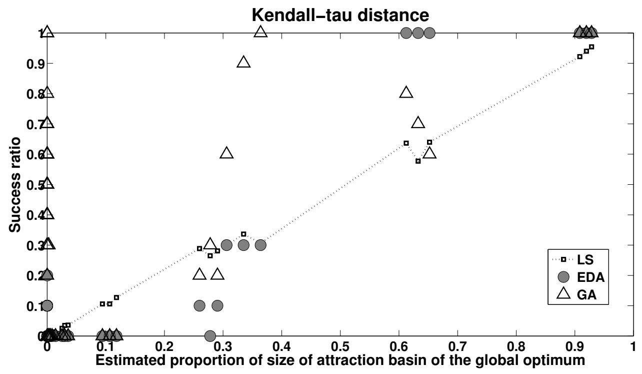


Fig. 5.7. Performance of the LS, the EDA and the GA with respect to the estimated attraction basin size of the global optimum. Kendall-tau distance.

In Figures 5.7 and 5.8 we show the success ratio of the LS, the EDA, and the GA (average proportion of the times that the best solution reached is the global optimum) with respect to the estimated size of the attraction basin of the global optimum, for the Kendall-tau and Cayley metrics, respectively. As expected, a good correlation can be observed for the LS. The larger the

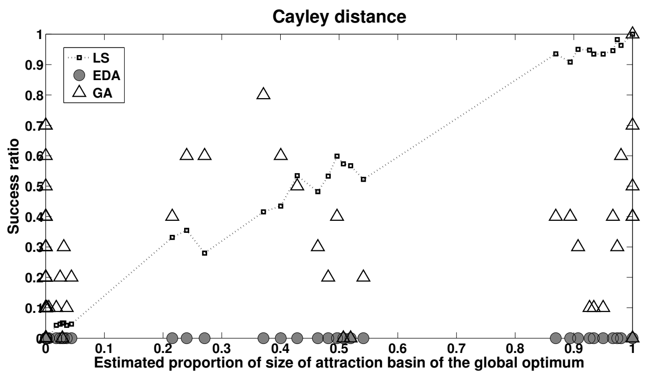


Fig. 5.8. Performance of the LS, the EDA, and the GA with respect to the estimated attraction basin size of the global optimum. Cayley distance.

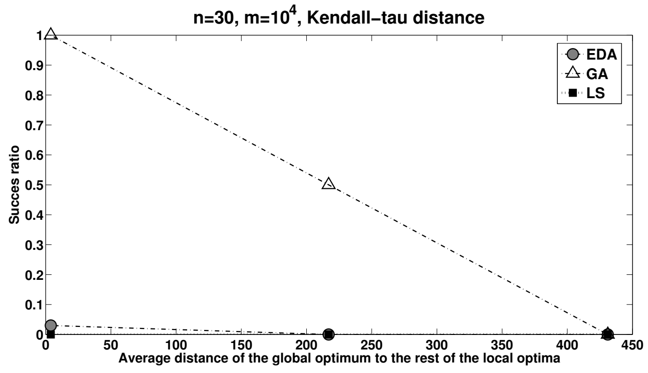


Fig. 5.9. Performance of the LS, the EDA, and the GA according to the average distance of the local optima to the global optimum. Kendall-tau distance.

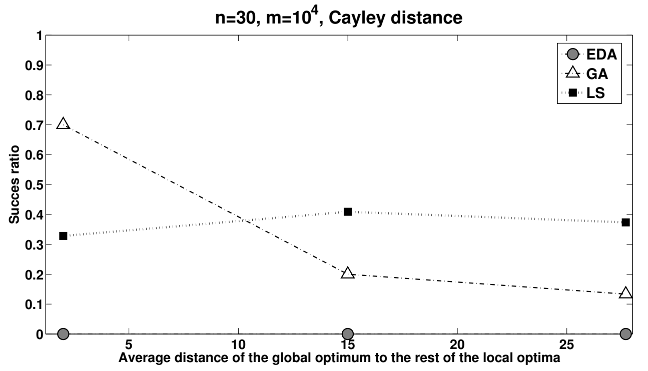


Fig. 5.10. Performance of the LS, the EDA, and the GA according to the average distance of the local optima to the global optimum. Cayley distance.

attraction basin of the global optimum, the higher the probability of finding it. In contrast, it is remarkable that the behavior of the EDA and the GA is not so clearly correlated. In fact, there are cases of small attraction basins and high success ratios and vice versa.

This type of scenario puts in value the usefulness of our generator: it allows us to go further, analyzing the impact of other characteristics such as the distribution of the local optima in the behavior of the algorithms. For this purpose, in Figures 5.9 and 5.10 the success ratio of the LS, the EDA, and the GA is measured related to the average distance of the global optimum to the rest of the local optima. For these two figures, 10 instances of each configuration of the local optima with similar attraction basin sizes of the global optimum have been chosen (between $1.1 \cdot 10^{-31}$ and $1.4 \cdot 10^{-25}$ for Kendall-tau and between 0.3 and 0.4 for Cayley). Due to this similarity, the behavior of the LS is almost identical, independently of the average distances between the local optima and the global optimum. However, this feature (average distance) notably affects the performance of the other two algorithms (particularly clear is the influence in the GA).

In view of the experiments, we conjecture that, for the GA the average distance between the local optima and the global optimum has a higher influence on its behavior than the size of the attraction basin of the global optimum. The potential of our generator is shown, as it would allow to perform a set of experiments analyzing the effect that reducing or increasing the average distance between the local optima and the global optimum has on the GA.

Regarding the EDA, for the Kendall-tau distance, we can also conclude that, if the global optimum is close to the rest of the local optima, the probability of finding it is a bit higher (near 0.04) than if the global optimum is not so close (almost 0.00). Notice that in these instances the proportion of the size of the attraction basin of the global optimum is very small, and the EDA finds it difficult (in all the cases) to reach it. However, for the Cayley distance, despite the proportion of the size of the attraction basin of the global optimum being considerably high (between 0.3 and 0.4), the EDA is not able to reach the global optimum, not even in the case that the global optimum is close to the rest of the local optima. We can conclude that this algorithm is not suitable for this type of instances. This is another example of a conclusion that we can obtain about the performance of this EDA when it is applied to instances with the properties chosen.

5.8 Conclusions

In the optimization field we can find several proposals of generators of instances. Above all, these generators are for the continuous domain, or for binary spaces. However, a few generators have been proposed for permutation-based COPs, and they are not able to control many properties of the generated instance. In this chapter, we present a flexible generator, based on a mixture of GM models. The parameters of these GM models are input parameters for the generator, and by tuning them, the user can control quantitative as well as qualitative properties of the resultant instance, such as the attraction basin sizes, the number of local optima, or their location. We have provided the restrictions that these parameters need to fulfill to obtain quantitative properties in the generated instance. Precisely, we have given sufficient conditions to ensure that the instance has a predefined number of local optima. Moreover, we have proposed to solve the constraints in the parameters by means of a linear programming problem. For this purpose, we have added a linear function to optimize that helps to obtain qualitative properties in the instance.

We have tested two important properties of our generator: its flexibility and its ability to create instances of very different complexities for local and population-based common algorithms. To assess the first property, we have considered instances of the PFSP and the LOP. We have measured the similarity between the artificial and the benchmark instances by means of the attraction basin sizes of the local optima. We find that, for small permutation sizes (those computationally comparable), our generator is flexible enough to create instances with almost the same sizes of attraction basins of the local optima as the benchmark instances. In order to study the second property, we have created a large set of instances of very different properties (permutation size, number of local optima, size of the attraction basins of the local optima, etc.) playing with the different input parameters available. According to the results, we claim that our generator is a very useful tool for the community to analyze and improve the performance of different optimization algorithms, and therefore it supposes an innovative and relevant proposal in this arena.

The three linear functions presented in this work were considered taking into account the size of the attraction basin of the global optimum. Nevertheless, we could consider other linear functions paying attention to the size of the attraction basin of other local optimum, or bearing in mind the sizes of the attraction basins of two or more local optima at the same time, or even considering other different criteria. In summary, our generator is a tool flexible enough to allow the user to define any kind of instance by setting the appropriate parameter values.

As we have seen, the weakest point of our algorithm is the expensive cost of evaluating a solution, that is basically conditioned by m (the number of local optima). Therefore, in order to be able to work with a high number of local optima, a key issue would be to think of different processes that help to reduce the time complexity of one fitness evaluation. Starting by sorting, from high to low, the local optima according to their fitness value, helps significantly to reduce this time. For example, in the case of population-based algorithms, when assigning a fitness to a given solution, we could stop the process of looking for the maximum value given by the GM models, when we find a value higher than the next fitness of the consensus permutation to test. According to a preliminary analysis, with this process we reduced the time to 10%. In the case of local search algorithms, a kind of incremental evaluation could be carried out, and in order to evaluate the neighbors it would not be necessary to evaluate all the models, because their change in fitness is limited by the new distance (+1 or -1). Other techniques such as parallelism could be also applied to make the computation times affordable. It is important to delve into this aspect in order to be able to produce, with our generator, instances with a large number of local optima, for which any algorithm could find a solution in reasonable time.

Part III

General conclusions, Future Work and Publications

6 General conclusions

Many different algorithms have been proposed in the literature to solve artificial as well as real-world instances of COPs. However, in most cases it is difficult to know beforehand if a particular algorithm will perform well on a given instance. That is, it is possible to have an algorithm that is successful in some instances and shows poor behavior for other instances of the same problem. Indeed, given a set of instances of COPs and different algorithms used to solve them, the behavior of these algorithms will be different depending on the properties of the instances. So, it would be useful to extract information from the instance in order to decide which algorithm or operator would be the most appropriate to solve it. In this sense, we find authors who are worried about establishing complexity measures associated to the type of problem, the algorithm used to solve it, or the operator chosen.

In this dissertation we worked in the direction of establishing complexity measures for local search algorithms when solving instances of permutation-based COPs. The complexity found by the local search algorithms comes derived from the neighborhood used, and, therefore, we focus on the properties that this neighborhood provokes on the instances. We consider two principal measures of complexity: the proportion of the attraction basin size of the global optimum and the proportion of the number of local optima. Both descriptors are directly related to the probability of finding the global optimum using a local search algorithm. In general, the higher the number of local optima, the more difficult it is to find the global optimum for the algorithm, while the higher the size of attraction basin of the global optimum, the easier it is to reach it. However, we are conscious that they should not be considered separately, as we could find an instance with a low number of local optima that turns out to be difficult for the algorithm because the size of the attraction basin of the global optimum is really small.

We have analyzed the evolution of these complexity measures as the permutation size grows. Focusing on the well-known TSP under the swap neighborhood, we have found that, as n grows, the average proportion of the size of the attraction basin of the global optimum decreases. We can conclude from this result that, according to our first complexity measure, on average, the instances become more difficult as n grows. Moreover, for values of n close to 10, as the variance experimented an increase, there seemed to be a huge difference between the complexity of different instances. We have also observed a fast decrease, as n grows, in the proportion of the number of local optima. So, although this would indicate that instances become easier with n, it is important to also consider the results obtained for the attraction basin of the global optimum. In fact, the proportions of the attraction basin size of the rest of the local optima (which are not the global optimum) increased. An important result derived from this study was the appearance of a phase transition phenomenon in the complexity of the instances, taking as parameters these two complexity measures. That is, the probability of an instance having values of the complexity measures higher than or equal to certain values decreased rapidly. Moreover, the threshold where this sudden decrease occurred appeared for small values of both descriptors as the permutation size increased: between 10^{-3} and 1 for the proportion of the attraction basin size of the global optimum and close to 10^{-5} for the proportion of the number of local optima.

This initial experiments confirmed the validity of the proportions of the attraction basin sizes of the global optima and the number of local optima as complexity measures. However, in practice, the global optimum is not known, and therefore the number of local optima is used as the main complexity descriptor. It is computationally unfeasible to obtain the exact number of local optima for medium-large size instances, as the process requires, in most of the cases, the exhaustive inspection of each solution of the search space. Thus, our next step was devoted to reviewing estimation methods for the number of local optima.

We reviewed the methods for estimating the number of local optima that can be found in the optimization field. Furthermore, we introduced some statistical methods used by ecologists to estimate the number of species in a population. The methods that come from both fields were tested by applying them to three different sets of instances: artificial instances generated by sampling a Dirichlet distribution, random TSP instances, and TSP instances with real distances between world cities as well as PFSP instances from the Taillard's benchmark. We found that the methods incorporated from the ecologists arena, and which have never been used before in the optimization field, provided the best results for all the datasets. Particularly, when the attraction basins of the different local optima were similar in size, the best method was *ChaoBunge*. However, when the difference between the distinct sizes of the attraction basins was high, the best results were given by *ChaoLee2*. Moreover, for small sample sizes (with respect to the number of local optima) *ChaoBunge* and *ChaoLee2* methods were the best-performers. However, we observed an instability behavior in *ChaoBunge*, so we concluded that both methods should be executed independently, and then compared. If the difference between both results is large, we should rely on the estimation provided by *ChaoLee2*. Otherwise, *ChaoBunge* will be chosen.

As seen, the performance of these estimation methods depends on properties such as the sizes of the attraction basins of the local optima. Improving the estimation of these attraction basin sizes, we would be able to improve the quality of the estimation methods for the number of local optima. So, we have proposed two methods for estimating the sizes of the attraction basins of the different local optima. Both methods started from the local optimum π^* for which we wanted to estimate its attraction basin size. The first method consisted of taking solutions at random from the whole search space. In the second method, the search space was divided in different subsets, which correspond to the sets of permutations at different distances. Three different sample strategies have been used to sample these subsets. They were applied to instances of two different problems and considering the adjacent swap and swap neighborhoods. From the experiments, we concluded that the second method with the right sampling strategy provides, in general, more accurate estimations. However, it is important to take into account that this choice of the sample strategy depends on the neighborhood considered. For the adjacent swap neighborhood, we saw that the DM-SD was the best performer, while for the second neighborhood, they were DM-ES and DM-SP. In fact, for the swap neighborhood the DM-SD provided very poor estimations.

Having information about the properties (complexity) of the instances is useful when designing efficient algorithms, as we can study their performance when applied to specific instances. So, it is important to have at our disposal instances with very different properties, and moreover, have knowledge about these properties beforehand. Thus, having a generator that is able to create instances with fixed and known properties is an important and very useful tool. We can find proposals of generators for the continuous and binary space, but little about the permutation space.

In the second part of this dissertation, we focused on the generation of instances of permutation-based COPs. We have proposed a generator for the space of permutations that depends on a number of parameters (the number of local optima and their distribution, among others). By tuning these parameters, the user is able to control the properties of the output instances. We have proved the flexibility of the generator by comparing the attraction basin sizes of the local optima of the generated instances with those obtained in instances of well-known benchmark problems. From this experiment, we have seen that our generator is able to create instances where the sizes of the attraction basins of the local optima are almost identical to those of benchmark problems or, at least, they match with the shape of the landscape. We have also studied the influence of the different input parameters and have given clues as to how to tune them in order to obtain instances with the desired qualitative properties. Generated instances were later used to observe the performance of an EDA, a GA and a local search algorithm. For instances with specific characteristics, the difficulty found by the EDA and the GA is connected with the difficulty experimented by the local search. However, we have seen that there are some properties (such as having local optima with low fitness values close to the global optimum) that have a high influence on the EDA and the GA, but do not affect the local search algorithm. This type of experiments prove that the generator is an important and useful tool that can help to analyze the performance and efficiency of many metaheuristics, not only local search algorithms.

7 Future Work

In this dissertation we have proposed and discussed the validity of the number of local optima and the sizes of attraction basins as complexity measures of instances, we have reviewed methods to estimate the number of local optima, incorporating new methods from the statistical arena used by ecologists and finally we have proposed and designed a generator of instances of permutation-based COPs. Related to each contribution, there are different aspects that could be addressed as future work.

Regarding the complexity measures, the two descriptors analyzed in the first part of the thesis are relevant to determine the complexity that a metaheuristic based on a local search finds when solving the instances of COPs. However, we have seen that they should not be studied independently, as they complement each other. Therefore, a way to measure complexity based on a combination of both of them should be further studied.

Focusing on the estimation of the number of local optima, we plan several future research lines. A first line is to improve the quality of some of the presented methods. For example, methods such as *ChaoBunge* or *ChaoLee2* depend on a cut-off value that determines the border between rare and abundant species (hard-to-find and easy-to-find local optima). This cut-off value could be properly tuned for each instance and sample size instead of being a fixed parameter. Another line of research has to do with the design of completely new methods to estimate the number of local optima. Statistical methods used by ecologists have not been explicitly designed to calculate the number of local optima but the number of species, and these are two different problems. In fact, these methods do not use all the information that is at hand. In this sense, and in order to design new estimation methods, we need to consider the specific characteristics of our problem. We should include the size of the search space, as well as the size of the attraction basins of the local optima that have been encountered in the sample. We can also use the fact that the search space is structured and, therefore, it could be divided based on a certain criterion, performing estimations for each region. This could help us to make different estimations according to the distinct areas and provide an accurate estimation for the total number of local optima of the whole search space.

Another way to improve the methods to estimate the number of local optima is by improving the method to estimate the attraction basins. In this problem, we plan to incorporate more information to the estimation of the attraction basins, such as the number of steps (solutions) traversed from the initial solution to the local optima. It can provide valuable information about the relative sizes of the attraction basins and their shapes.

Regarding the last contribution, i.e., the generator of instances, we have seen that its weakest point is the cost of evaluating a solution, which is basically conditioned by m (the number of local optima). Therefore, in order to be able to work with a large number of local optima, a key issue would be to think of different processes that help to reduce the time complexity of each fitness evaluation. A first step in this direction is to sort, from high to low, the local optima according to their fitness value. That could significantly help to reduce this time. For example, in the case of population-based algorithms, when assigning a fitness value to a given solution, we could stop the process of looking for the maximum value given by the GM models, when we find a value higher than the next fitness of the consensus permutation to test. According to a preliminary experimentation, using this process we were able to reduce the execution time to 10%. In the case of local search algorithms, a kind of incremental evaluation could be carried out and, in order to evaluate the neighbors, it would not be necessary to evaluate all the models, because their change in fitness is bounded by the distance (+1 or -1). Other techniques, such as parallelism, could be also applied to reduce the execution times.

Apart from the cost of evaluating a solution, we find it important to deal with an issue that has already been mentioned. When generating the instances we assume that two local optima can not have the same objective function value. This has been done by forcing the distance between two different local optima to be higher than or equal to two. This framework does not always reflect the reality, because we can find many instances of real COPs with two neighboring local optima (and therefore with the same objective function value). This situation can cause, for example, maximal or minimal flat regions, and this could have different consequences in the distribution of the attraction basins along the search space. So, a more detailed analysis should be done to explore this case, according to the neighborhood chosen. Moreover, we have not considered the possibility of generating instances of symmetric optimization problems, such as the Symmetric TSP. One of the implications of this type of problems is the appearance of several solutions with the same fitness value. Furthermore, in these problems, we have to be careful with the definition of neighborhood and distance. These characteristics have important consequences in our generator, and therefore should be taken into consideration in order to be able to design instances for symmetric optimization problems.

8

Publications

The research work carried out during this thesis has produced the following publications:

8.1 Referred journals

- **L. Hernando**, A. Mendiburu, & J. A. Lozano (2015). A tunable generator of instances of permutation-based combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*. Accepted.
- **L. Hernando**, A. Mendiburu, & J. A. Lozano (2013). An evaluation of methods for estimating the number of local optima in combinatorial optimization problems. *Evolutionary Computation*, 21(4), 625-658.

8.2 Book chapters

- R. Martí, J. A. Lozano, A. Mendiburu & **L. Hernando** (2015). Multi Start Methods and Local Optima. In R. Martí, P. Pardalos and M. Resende, editors, *Handbook of heuristics*. Springer. *In Press*.

8.3 International conference communications

- **L. Hernando**, A. Mendiburu, & J. A. Lozano (2013). Generating Customized Landscapes in Permutation-based Combinatorial Optimization Problems. In the *International Workshop on Intelligent Perception and Image Understanding*, Xi'an, China.
- **L. Hernando**, A. Mendiburu, & J. A. Lozano (2013). Generating Customized Landscapes in Permutation-based Combinatorial Optimization Problems. *Learning and Intelligent OptimizatioN Conference (LION 7)*, Catania, Italy. In Volume 7997 of Lecture Notes in Computer Science, pages 299-303, Springer Berlin Heidelberg. [BEST PAPER AWARD].

- **L. Hernando**, J. A. Pascual, A. Mendiburu & J. A. Lozano (2011). A study on the complexity of TSP instances under the swap neighbor system. *IEEE Symposium on Foundations of Computational Intelligence (FOCI 2011)*, Paris, France. In *Proceedings of IEEE Symposium on FOCI2011, part of the IEEE Symposium Series on Computational Intelligence 2011*, pages 15-21.

8.4 National conference communications

- **L. Hernando**, J. A. Pascual, A. Mendiburu & J. A. Lozano (2010). Estudio preliminar sobre la complejidad de las instancias del TSP bajo el sistema de vecinos 2-opt. *VII Congreso Español sobre Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2010)*, Valencia, Spain.
- **L. Hernando**, A. Mendiburu & J. A. Lozano (2013). Generador de instancias de problemas de optimizacion combinatoria basados en permutaciones. *IX Congreso Español sobre Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2013)*, Madrid, Spain. [FINALIST OF THE GENIL AWARD].

8.5 Collaborations

- J. Ceberio, **L. Hernando**, A. Mendiburu & J. A. Lozano (2013). Understanding Instance Complexity in the Linear Ordering Problem. *14th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL-2013)*, Hefei, China. In Volume 8206 of Lecture Notes in Computer Science, pages 479-486, Springer Berlin Heidelberg.

References

[1] Achlioptas, D., Naor, A., and Peres, Y. (2005). Rigorous location of phase transitions in hard optimization problems. *Nature*, 435(7043):759–764.

[2] Albrecht, A., Lane, P., and Steinhofel, K. (2008). Combinatorial landscape analysis for k-SAT instances. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2498–2504, Hong Kong. IEEE Press.

[3] Albrecht, A., Lane, P., and Steinhofel, K. (2010). Analysis of Local Search Landscapes for k-SAT Instances. *Mathematics in Computer Science*, 3(4):465–488.

[4] 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.

[5] Alyahya, K. and Rowe, J. (2014). Phase Transition and Landscape Properties of the Number Partitioning Problem. In Blum, C. and Ochoa, G., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 8600 of *Lecture Notes in Computer Science*, pages 206–217. Springer Berlin Heidelberg.

[6] Angel, E. and Zissimopoulos, V. (1998). Autocorrelation coefficient for the graph bipartitioning problem. *Theoretical Computer Science*, 191(1-2):229–243.

[7] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G., and Stewart Jr, W. R. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32.

[8] Bianchi, L. and Campbell, A. M. (2007). Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. *European Journal of Operational Research*, 176(1):131–144.

[9] Bierwirth, C. and Mattfeld, D. C. (1999). Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation*, 7(1):1–17.

[10] Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA, USA.

[11] Bunge, J. and Fitzpatrick, M. (1993). Estimating the Number of Species: A Review. *Journal of the American Statistical Association*, 88(421):364–373.

[12] Burnham, K. P. and Overton, W. S. (1978). Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals. *Biometrika*, 65(3):625–633.

[13] Caragiannis, I., Procaccia, A. D., and Shah, N. (2013). When Do Noisy Votes Reveal the Truth? In *Proceedings of the Fourteenth ACM Conference on Electronic Comerce*, pages 143–160, New York, USA. ACM.

[14] Caruana, R. and Mullin, M. (1999). Estimating the Number of Local Minima in Big, Nasty Search Spaces. In *In Proceedings of IJCAI-99 Workshop on Statistical Machine Learning for Large-Scale Optimization*.

[15] Ceberio, J., Irurozki, E., Mendiburu, A., and Lozano, J. A. (2012). A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1(1):103–117.

[16] 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., editors, *2011 International Conference on Neural Information Processing (ICONIP-2011)*, volume 7063 of *Lecture Notes in Computer Science*, pages 461–470. Springer Berlin Heidelberg.

[17] Chao, A. (1984). Nonparametric Estimation of the Number of Classes in a Population. *Scandinavian Journal of Statistics*, 11(4):265–270.

[18] Chao, A. and Bunge, J. (2002). Estimating the Number of Species in a Stochastic Abundance Model. *Biometrics*, 58(3):531–539.

[19] Chao, A. and Lee, S.-M. (1992). Estimating the Number of Classes via Sample Coverage. *Journal of the American Statistical Association*, 87(417):210–217.

[20] Chao, A. and Yang, M. C. K. (1993). Stopping Rules and Estimation for Recapture Debugging with Unequal Failure Rates. *Biometrika*, 80(1):193–201.

[21] Cheng, W. and Hullenmeier, 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.

[22] Chicano, F., Whitley, L. D., and Alba, E. (2011). A Methodology to Find the Elementary Landscape Decomposition of Combinatorial Optimization Problems. *Evolutionary Computation*, 19(4):597–637.

[23] Cirasella, J., Johnson, D. S., McGeoch, L. A., and Zhang, W. (2001). The Asymmetric Traveling Salesman Problem: Algorithms, Instance generators, and Tests. In Buchsbaum, A. and Snoeyink, J., editors, *Algorithm Engineering and Experimentation*, volume 2153 of *Lecture Notes in Computer Science*, pages 32–59. Springer Berlin Heidelberg.

[24] Collard, P., Gaspar, A., Clergue, M., and Escazut, C. (1998). Fitness Distance Correlation, as Statistical Measure of Genetic Algorithm Difficulty, Revisited. In *Proceedings of the European Conference on Artificial Intelligence*, pages 650–654. John Witley & Sons, Ltd.

[25] De Jong, K. A., Potter, M. A., and Spears, W. M. (1997). Using Problem Generators to Explore the Effects of Epistasis. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 338–345. Morgan Kaufmann.

[26] De Lima, T. and Ayala-Rincon, M. (2012). Complexity of Cayley Distance and other General Metrics on Permutation Groups. In *Proceedings of the Seventh Colombian Computing Congress (CCC)*, pages 1–6, Medellin. IEEE Press.

[27] Draskoczy, B. (2010). Fitness Distance Correlation and Search Space Analysis for Permutation Based Problems. In Cowling, P. and Merz, P., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6022 of *Lecture Notes in Computer Science*, pages 47–58. Springer Berlin Heidelberg.

[28] Drugan, M. M. (2013). Generating QAP instances with known optimum solution and additively decomposable cost function. *Journal of Combinatorial Optimization*, pages 1–35.

[29] Engelsa, C. and Manthey, B. (2009). Average-case approximation ratio of the 2-exchange algorithm for the TSP. *Operations Research Letters*, 37(2):83–84.

[30] Eremeev, A. V. and Reeves, C. R. (2002). Non-parametric Estimation of Properties of Combinatorial Landscapes. In Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., and Raidl, G., editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 31–40. Springer Berlin Heidelberg.

[31] Eremeev, A. V. and Reeves, C. R. (2003). On Confidence Intervals for the Number of Local Optima. In Cagnoni, S., Johnson, C., Cardalda, J., Marchiori, E., Corne, D., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G., and Hart, E., editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 224–235. Springer Berlin Heidelberg.

[32] Esty, W. W. (1985). Estimation of the Number of Classes in a Population and the Coverage of a Sample. *Mathematical Scientists*, 10:41–50.

[33] Fligner, M. and Verducci, J. S. (1988). Multistage ranking models. *Journal of the American Statistical Anssociation*, 83(403):892–901.

[34] Fligner, M. A. and Verducci, J. S. (1986). Distance based ranking models. *Journal of the Royal Statistical Society*, 48(3):359–369.

[35] Fonlupt, C., Robilliard, D., Preux, P., and Talbi, E.-G. (1999). Fitness Landscapes And Performance of Meta-Heuristics. In Voß, S., Martello, S., Osman, I., and Roucairol, C., editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 257–268. Springer US.

[36] Gallagher, M. (2001). Fitness Distance Correlation of Neural Network Error Surfaces: A Scalable, Continuous Optimization Problem. In De Raedt, L. and Flach, P., editors, *Machine Learning: ECML 2001*, volume 2167 of *Lecture Notes in Computer Science*, pages 157–166. Springer Berlin Heidelberg.

[37] Gallagher, M. and Yuan, B. (2006). A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation*, 10(5):590–603.

[38] Garey, M. and Jonhson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman & Co., New York, USA.

[39] Garnier, J. and Kallel, L. (2001). How to detect all maxima of a function. In Kallel, L., Naudts, B., and Rogers, A., editors, *Theoretical Aspects of Evolutionary Computing*, Natural Computing Series, pages 343–370. Springer Berlin Heidelberg, London, UK.

[40] Gent, I. P. and Walsh, T. (1996). The TSP Phase Transition. *Artificial Intelligence*, 88(1-2):349–358.

[41] Gomes, C. P. and Selman, B. (2005). Computational science: Can get satisfaction. *Nature*, 435(7043):751–752.

[42] Good, I. J. (1953). The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 40(3-4):237–264.

[43] Grundel, D., Krokhmal, P., Oliveira, C., and Pardalos, P. (2007). On the Number of Local Minima for the Multidimensional Assignment Problem. *Journal of Combinatorial Optimization*, 13(1):1–18.

[44] Harris, B. (1959). Determining bounds on integrals with applications to cataloging problems. *Annals of Mathematical*, 30(2):521–548.

[45] Hartman, A. K. and Weight, M. (2005). *Phase Transitions in Combinatorial Optimization Problems*. Wiley-VCH.

[46] Hasegawa, M., Ikeguchi, T., and Aihara, K. (1997). Combination of Chaotic Neurodynamics with the 2-exchange Algorithm to Solve Traveling Salesman Problems. *Physical Review Letters*, 79(12):2344–2347.

[47] 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.

[48] Hertz, A., Jaumard, B., and de Aragão, M. P. (1994). Local optima topology for the k-coloring problem. *Discrete Applied Mathematics*, 49(1-3):257–280.

[49] Hogg, T., Huberman, B. A., and Williams, C. P. (1996). Phase transitions and the search problem. *Artificial Intelligence*, 81(1-2):1–15.

[50] Holland, J. H. (2000). Building Blocks, Cohort Genetic Algorithms, and Hyperplane-Defined Functions. *Evolutionary Computation*, 8(4):373–391.

[51] Hossain, M. M., Abbass, H. A., Lokan, C., and Alam, S. (2010). Adversarial Evolution: Phase transition in non-uniform hard satisfiability problems. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE Press.

[52] Irurozki, E. (2014). *Sampling and learning distance-based probability models for permutation spaces*. PhD thesis, University of the Basque Country.

[53] Irurozki, E., Calvo, B., and Lozano, J. A. (2014a). An R package for permutations, Mallows and Generalized Mallows models. Technical report, University of the Basque Country.

[54] Irurozki, E., Calvo, B., and Lozano, J. A. (2014b). Sampling and learning the Mallows and Generalized Mallows models under the Cayley distance. Technical report, University of the Basque Country.

[55] Kirkpatrick, S. and Toulouse, G. (1985). Configuration space analysis of Travelling Salesman Problems. *Journal de Physique*, 46(8):1277–1292.

[56] Lafferty, G. and John, L. (2002). Conditional Models on the Ranking Poset. In Thrun, S. and Obermayer, K., editors, *Advances in Neural Information Processing Systems*, pages 415–422. MIT Press, Cambridge, MA.

[57] Lagaris, I. E. and Tsoulos, I. G. (2008). Stopping rules for box-constrained stochastic global optimization. *Applied Mathematics and Computation*, 197(2):622–632.

[58] Lebanon, G. and Mao, Y. (2008). Non-parametric Modeling of Partially Ranked Data. *Journal of Machine Learning Research*, 9(81):2401–2429.

[59] Link, W. A. (2003). Nonidentifiability of Population Size from Capture-Recapture Data with Heterogeneous Detection Probabilities. *Biometrics*, 59(4):1123–1130.

[60] Mallows, C. L. (1957). Non-null ranking models. *Biometrika*, 44(1-2):114–130.

[61] Mandhani, B. and Melia, M. (2009). Tractable Search for Learning Exponential Models of Rankings. In Dyk, D. V. and Welling, M., editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS-09)*, volume 5, pages 392–399. Journal of Machine Learning Research.

[62] Matsuura, T. and Ikeguchi, T. (2008). Chaotic Search for Traveling Salesman Problems by Using 2-exchange and Or-opt Algorithms. In Kurkova, V., Neruda, R., and Koutnik, J., editors, *19th International Conference on Artificial Neural Networks*, volume 5769 of *Lecture Notes in Computer Science*, pages 563–572. Springer Berlin Heidelberg.

[63] Mattfeld, D. C. and Bierwirth, C. (1999). A Search Space Analysis of the Job Shop Scheduling Problem. *Annals of Operations Research*, 86(0):441–453.

[64] Mézard, M., Parisi, G., and y Zecchina, R. (2002). Analytic and Algorithmic Solution of Random Satisfiability Problems. *Science*, 297(5582):812–815.

[65] Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1999). Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(6740):133–137.

[66] Morgan, R. and Gallagher, M. (2010). When Does Dependency Modelling Help? Using a Randomized Landscape Generator to Compare Algorithms in Terms of Problem Structure. In Schaefer, R., Cotta, C., Kolodziej, J., and Rudolph, G., editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 94–103. Springer Berlin Heidelberg.

[67] Morgan, R. and Gallagher, M. (2012). Using Landscape Topology to Compare Continuous Metaheuristics: A Framework and Case Study on EDAs and Ridge Structure. *Evolutionary Computation*, 20(2):277–299.

[68] Naudts, B. and Kallel, L. (2000). A Comparison of Predictive Measures of Problem Difficulty in Evolutionary Algorithms. *IEEE Transactions On Evolutionary Computation*, 4(1):1–15.

[69] Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc.

[70] Rardin, R. L. and Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304.

[71] Reeves, C. and Aupetit-Bélaidouni, M. (2004). Estimating the Number of Solutions for SAT problems. In Yao, X., Burke, E., Lozano, J., Smith, J., Merelo-Guervós, J., Bullinaria, J., Rowe, J., Tino, P., Kabán, A., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 101–110. Springer Berlin Heidelberg.

[72] Reeves, C. R. (2001). Statistical Properties of Combinatorial Landscapes: An Application to Scheduling Problems. In *Proceedings of the fourth Metaheuristic International Conference*, pages 691–695.

[73] Reeves, C. R. (2002). The "Crossover Landscape" and the Hamming Landscape for Binary Search Spaces. In *Foundations of Genetic Algorithms 7*, pages 81–98. Morgan Kaufmann.

[74] Reeves, C. R. and Eremeev, A. V. (2004). Statistical analysis of local search landscapes. *Journal of the Operational Research Society*, 55(7):687–693.

[75] Rönkkönen, J., Li, X., Kyrki, V., and Lampinen, J. (2011). A framework for generating tunable test functions for multimodal optimization. *Soft Computing*, 15(9):1689–1706.

[76] Schiavinotto, T. and Stützle, T. (2005). The Linear Ordering Problem: Instances, earch space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402.

[77] Schiavinotto, T. and Stutzle, T. (2007). A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153.

[78] Schwarz, C. J. and Seber, G. A. F. (1999). Estimating Animal Abundance: Review III. *Statistical Science*, 14(4):427–456.

[79] Seber, G. A. F. (1986). A Review of Estimating Animal Abundance. *Biometrics*, 42(2):267–292.

[80] Seber, G. A. F. (1992). A Review of Estimating Animal Abundance II. *International Statistical Review*, 60(2):129–166.

[81] Solomon, A., Barnes, J. W., Dokov, S. P., and Acevedo, R. (2003). Weakly Symmetric Graphs, Elementary Landscapes, and the TSP. *Applied Mathematics Letters*, 16(3):401–407.

[82] 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, pages 78–163. Springer Berlin Heidelberg.

[83] Stadler, P. F., Hordijk, W., and Fontanari, J. F. (2003). Phase transition and landscape statistics of the number partitioning problem. *Physical Review E*, 67(5):1–7.

[84] Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

[85] Tayarani-N, M. and Prugel-Bennett, A. (2014). On the Landscape of Combinatorial Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 18(3):420–434.

[86] Tomassini, M., Vérel, S., and Ochoa, G. (2008). Complex-network analysis of combinatorial spaces: The NK landscape case. *Physical Review E*, 78(6):66–114.

[87] Van Hemert, J. and Urquhart, N. (2004). Phase Transition Properties of Clustered Travelling Salesman Problem Instances Generated with Evolutionary Computation. In Yao, X., Burke, E., Lozano, J., Smith, J., Merelo-Guervós, J., Bullinaria, J., Rowe, J., Tiňo, P., Kabán, A., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 151–160. Springer Berlin Heidelberg.

[88] Verhoeven, M., Aarts, E., and Swinkels, P. (1995). A parallel 2-exchange algorithm for the Traveling Salesman Problem. *Future Generation Computer Systems*, 11(2):175–182.

[89] Vose, M. D. and Wright, A. H. (1995). Stability of Vertex Fixed Points and Applications. In *Foundations of Genetic Algorithms 3*, pages 103–113. Morgan Kaufmann.

[90] Wang, J.-P. (2010). Estimating the species richness by a Poisson-Compound Gamma model. *Biometrika*, 97(3):727–740.

[91] Wang, J.-P. and Lindsay, B. (2008). An exponential partial prior for improving NPML estimation for mixtures. *Statistical Methodology*, 5(1):30–45.

[92] Wang, J.-P. Z. and Lindsay, B. G. (2005). A Penalized Nonparametric Maximum Likelihood Approach to Species Richness Estimation. *Journal of the American Statistical Association*, 100(471):942–959.

[93] Whitley, D., Rana, S., Dzubera, J., and Mathias, K. E. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1–2):245 – 276.

[94] Whitley, L. D., Sutton, A. M., and Howe, A. E. (2008). Understanding Elementary Landscapes. *Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO '08*, pages 585–592.

[95] Yuan, B. and Gallagher, M. (2003). On Building a Principled Framework for Evaluating and Testing Evolutionary Algorithms: A Continuous Landscape Generator. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 451–458. IEEE Press.

[96] Zhang, W. (2004). Phase Transitions and Backbones of the Asymmetric Traveling Salesman Problem. *Journal of Artificial Intelligence Research*, 21:471–497.

9 Appendices

9.1 Appendix I

9.1.1 Estimation of the number of local optima in synthetic instances

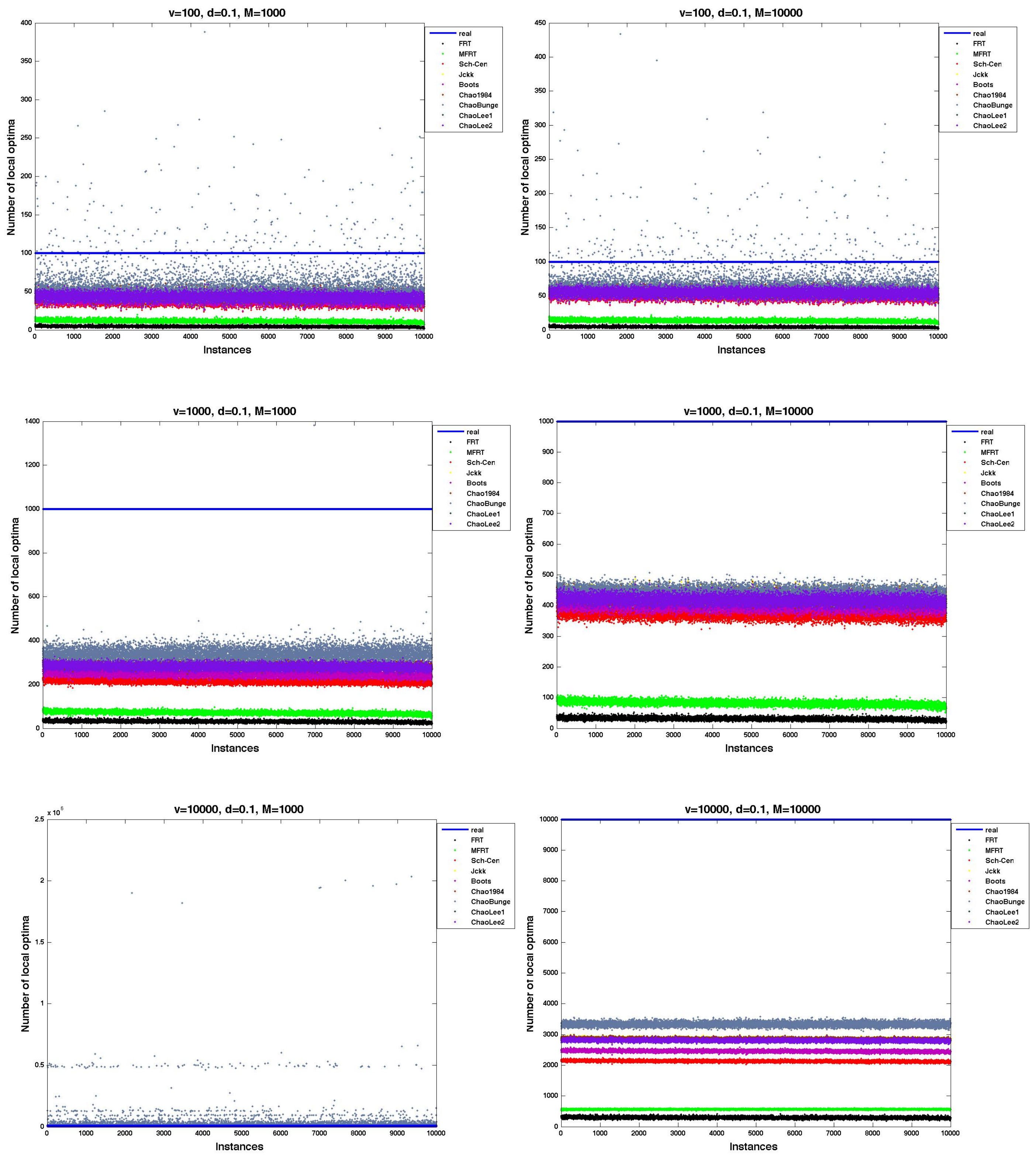


Fig. 9.1. Estimations of the number of local optima provided by the methods reviewed in Chapter 3 for the synthetic instances created by sampling a Dirichlet with parameter $d = 0.1$. The number of local optima are (from top to bottom) $n = 100, 1000, 10000$, and the sample sizes considered are $M = 1000$ (left) and $M = 10000$ (right).

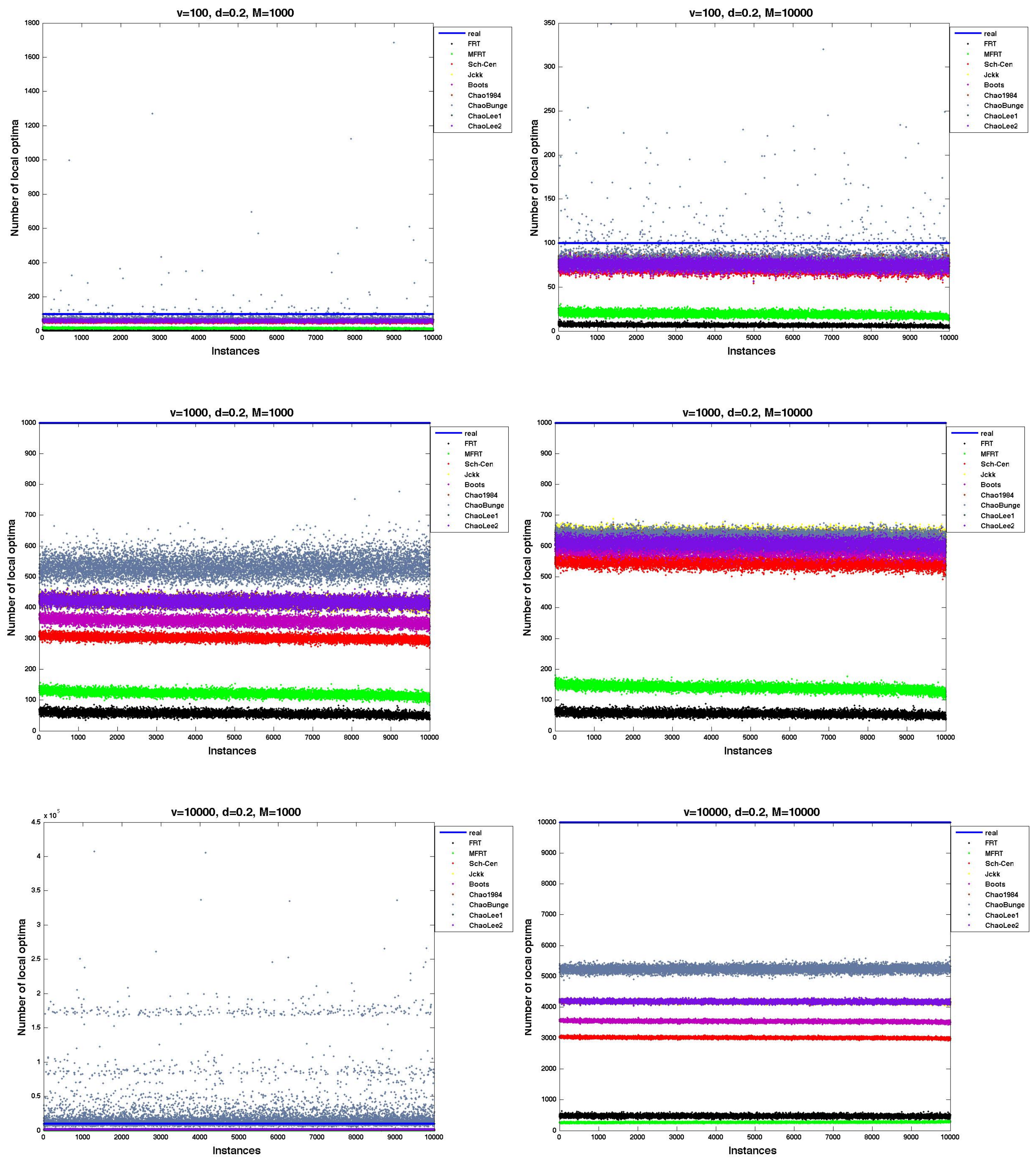


Fig. 9.2. Estimations of the number of local optima provided by the methods reviewed in Chapter 3 for the synthetic instances created by sampling a Dirichlet with parameter $d = 0.2$. The number of local optima are (from top to bottom) $n = 100, 1000, 10000$, and the sample sizes considered are $M = 1000$ (left) and $M = 10000$ (right).

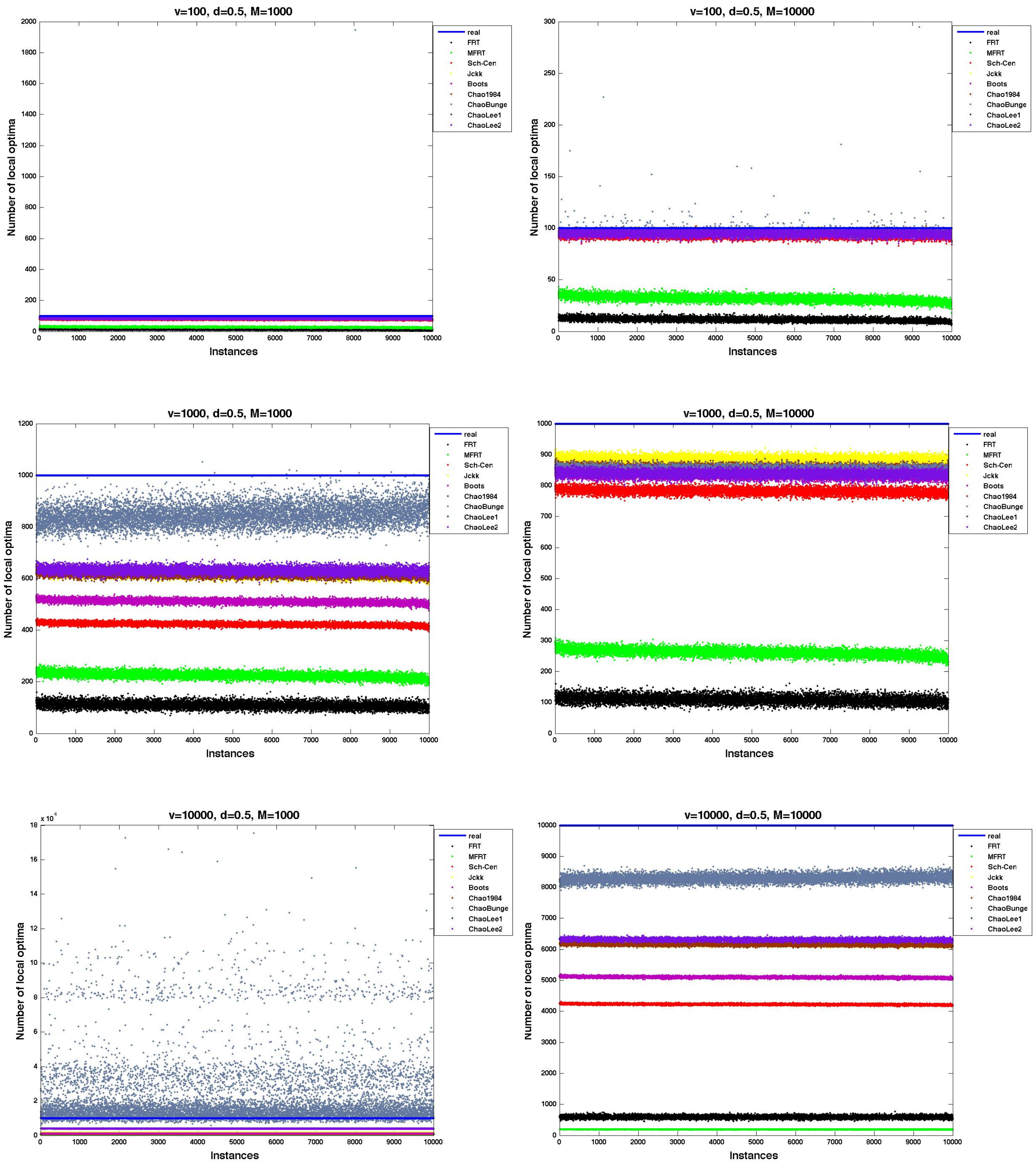


Fig. 9.3. Estimations of the number of local optima provided by the methods reviewed in Chapter 3 for the synthetic instances created by sampling a Dirichlet with parameter $d = 0.5$. The number of local optima are (from top to bottom) $n = 100, 1000, 10000$, and the sample sizes considered are $M = 1000$ (left) and $M = 10000$ (right).

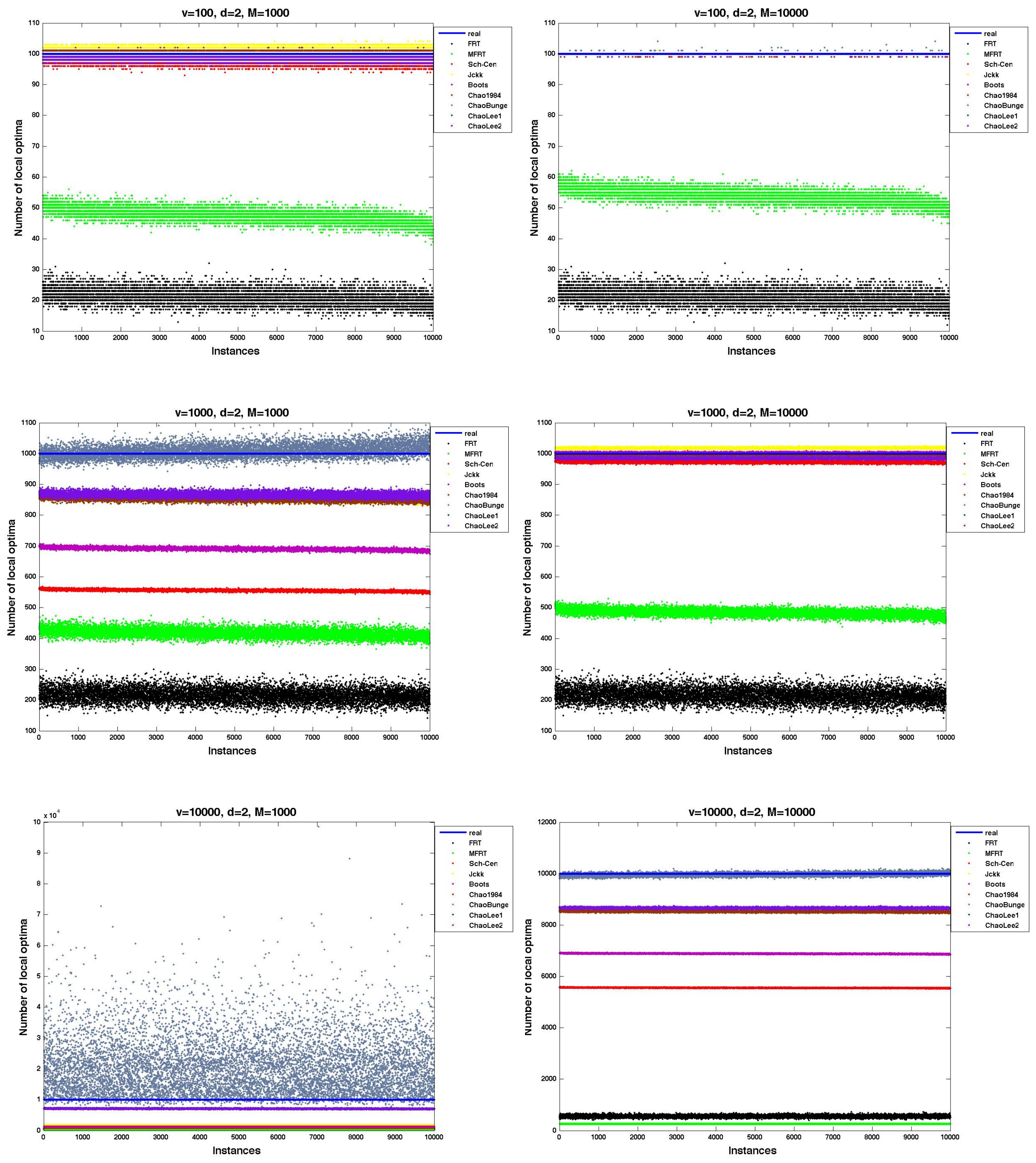


Fig. 9.4. Estimations of the number of local optima provided by the methods reviewed in Chapter 3 for the synthetic instances created by sampling a Dirichlet with parameter $d = 2$. The number of local optima are (from top to bottom) $n = 100, 1000, 10000$, and the sample sizes considered are $M = 1000$ (left) and $M = 10000$ (right).

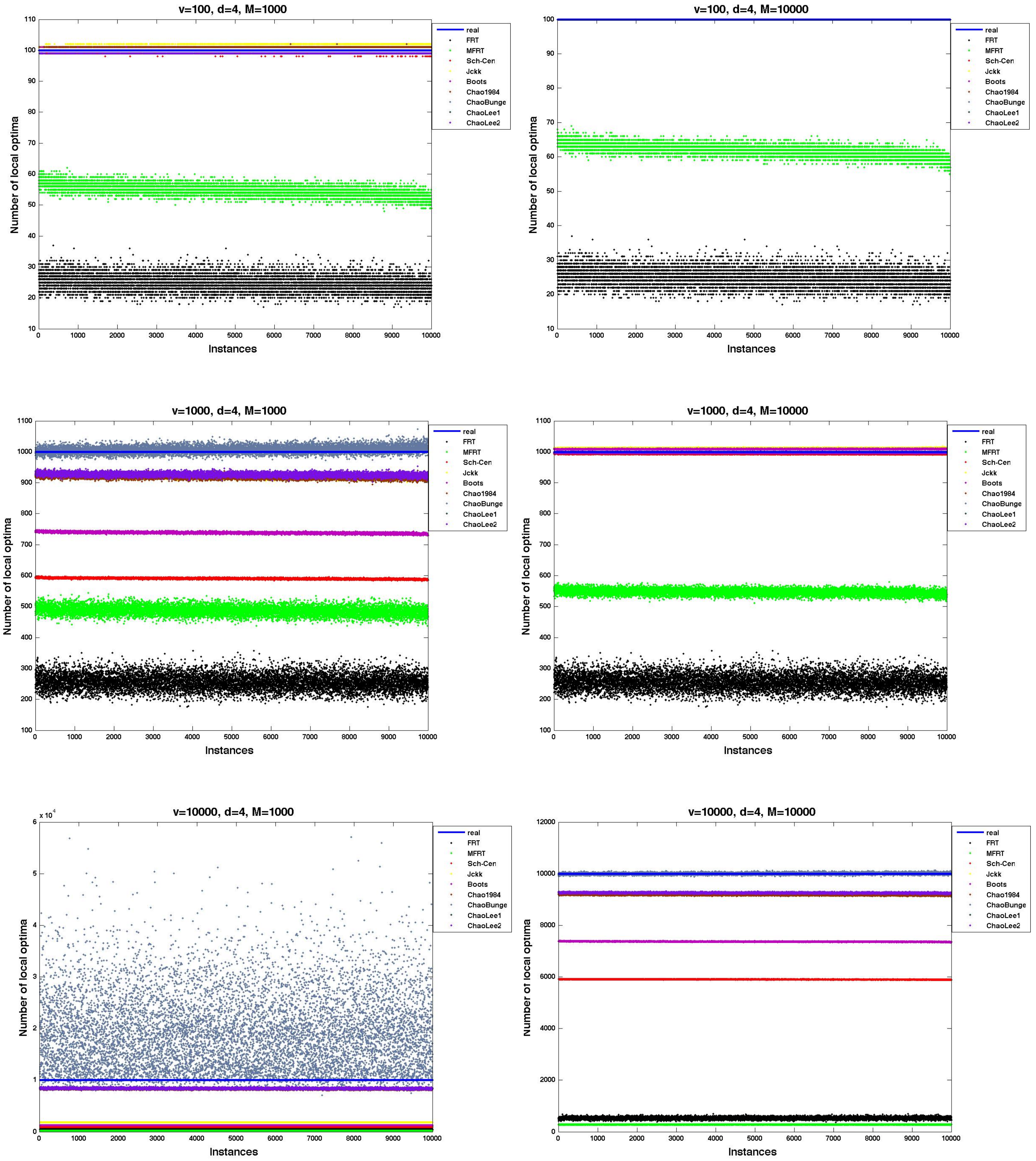


Fig. 9.5. Estimations of the number of local optima provided by the methods reviewed in Chapter 3 for the synthetic instances created by sampling a Dirichlet with parameter $d = 4$. The number of local optima are (from top to bottom) $n = 100, 1000, 10000$, and the sample sizes considered are $M = 1000$ (left) and $M = 10000$ (right).

9.1.2 Estimation of the number of local optima in random instances of TSP

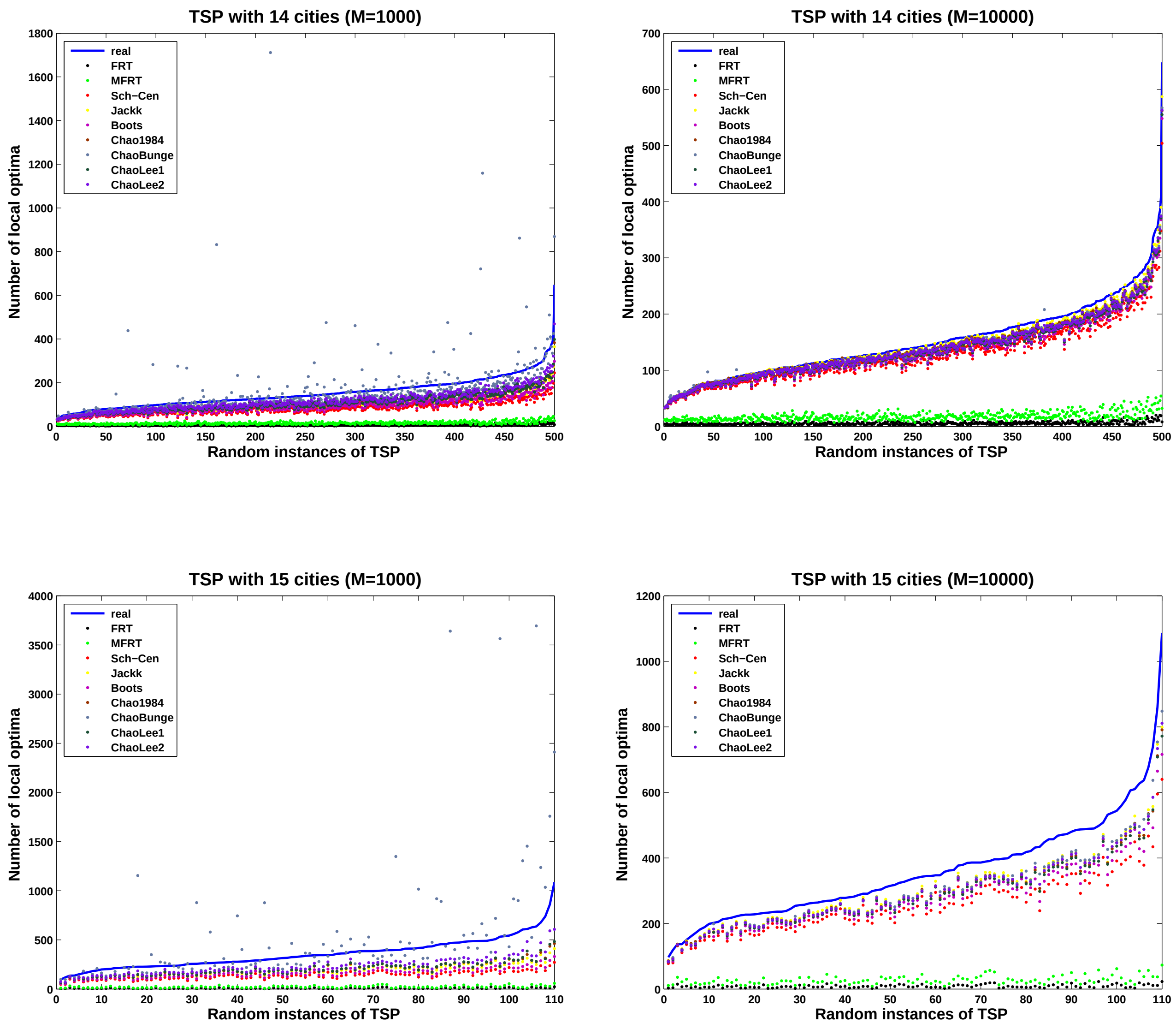
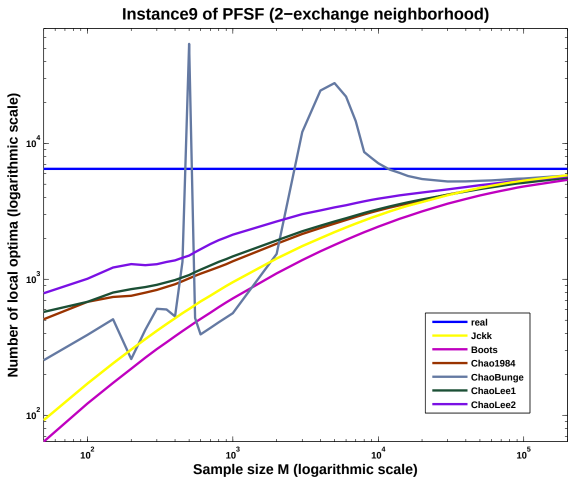


Fig. 9.6. Estimations of the number of local optima provided by the methods reviewed in Chapter 3 for the random instances of TSP. The number of cities considered are $n = 14$ (top) and $n = 15$ (bottom), and the sample size used is $M = 1000$ (left) and $M = 10000$ (right).

9.1.3 Estimation of the number of local optima in benchmark instances of PFSP



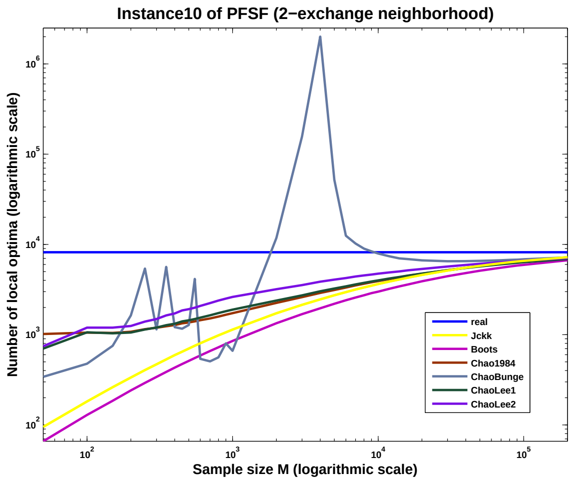


Fig. 9.7. Estimations obtained by the best methods among those reviewed in Chapter 3 for the instance number 9 (top) and instance number 10 (bottom) of PFSP using the swap neighborhood as the sample size grows.

9.2 Appendix II

9.2.1 Figures of the generator of instances for the PFSP

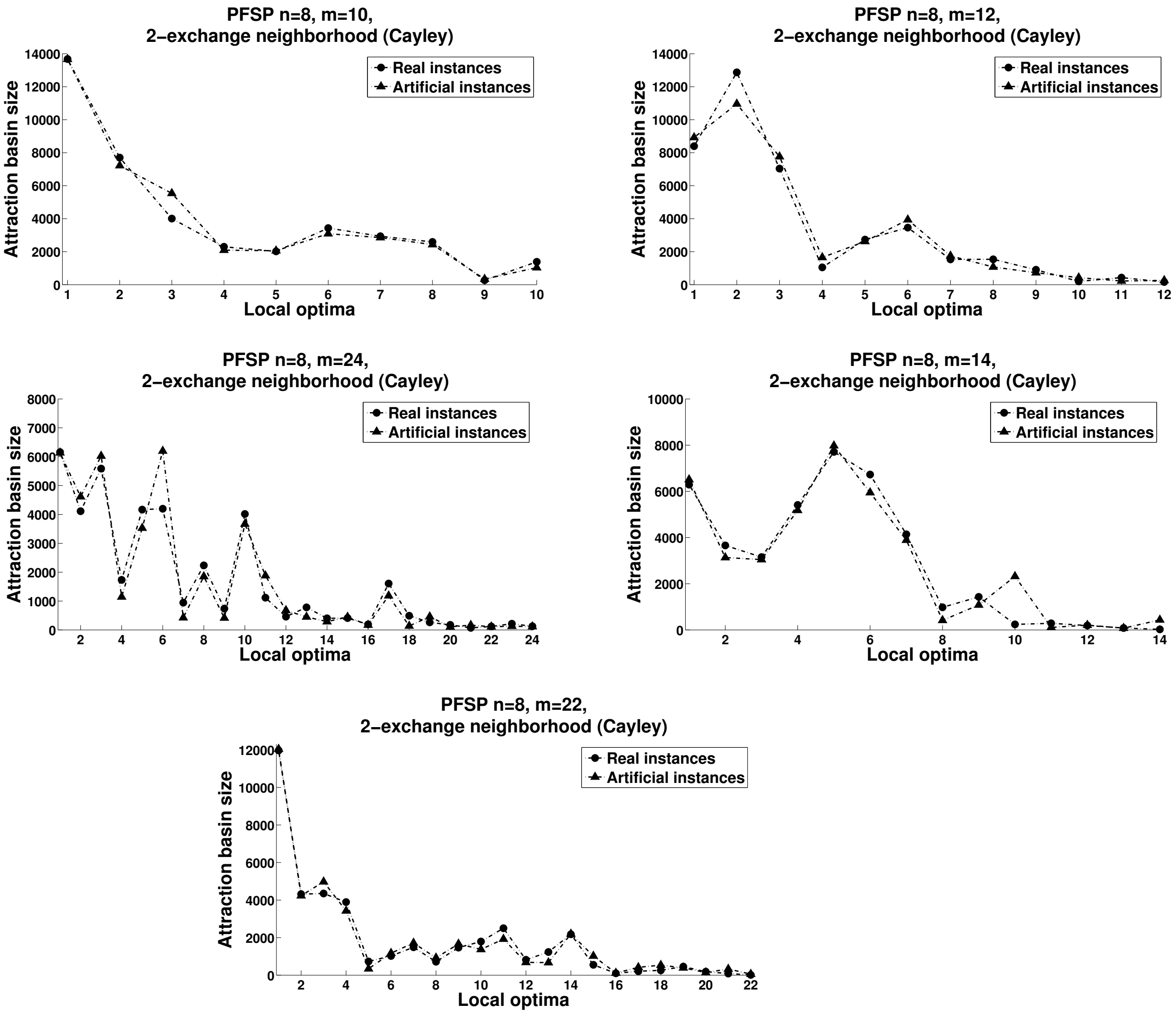


Fig. 9.8. Sizes of the attraction basins of the different local optima found in each of the five instances of the PFSP using the swap neighborhood. We represent those sizes found in the benchmark instance with a circle, and the sizes obtained in the generated instance with a triangle. The instances are, from left to right and from top to bottom, the first, second, third, fourth and fifth.

9.2.2 Figures of the generator of instances for the LOP

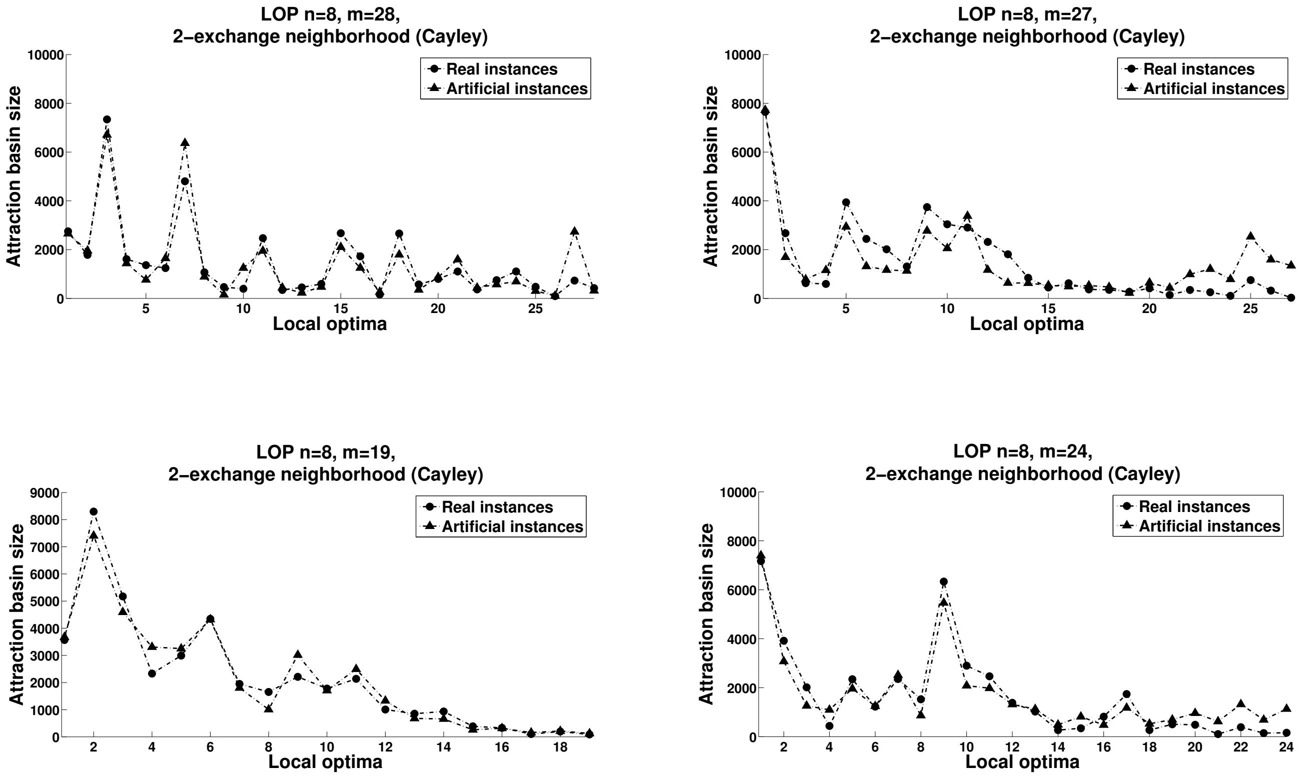


Fig. 9.9. Sizes of the attraction basins of the different local optima found in each of the five instances of the LOP using the swap neighborhood. We represent those sizes found in the benchmark instance with a circle, and the sizes obtained in the generated instance with a triangle. The instances are, from left to right and from top to bottom, the first, second, third, fourth and fifth.