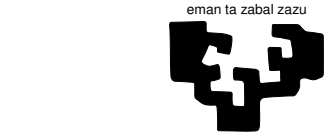


Universidad del País Vasco
Euskal Herriko Unibertsitatea

Understanding Non-Convex Optimization Problems and Stochastic Optimization Algorithms

by

Etor Arza

Supervised by Aritz Perez and Ekhiñe Irurozki

Donostia - San Sebastián, September 2023

Once you've got a task to do, it's better to do it than live with the fear of it.
–Joe Abercrombie, The Blade Itself

This Ph.D. thesis was carried out at the Basque Center for Applied Mathematics (BCAM) within the Intelligent Systems Group (ISG) and has been partially supported by the Basque Government (Elkartek KONFLOT KK-2022/00100, BERC 2018-2021 and 2022-2025 programs), and by the Spanish Ministry of Science, Innovation and Universities trough the BCAM Severo Ochoa accreditation SEV-2017-0718. Etor Arza has been supported by a grant of the Spanish Ministry of Economy and Competitiveness (PRE2019-088164).

Acknowledgments

I would like to start expressing my appreciation to my supervisors, Aritz Perez, Ekhiñe Irurozki and Josu Ceberio. Thank you for improving my papers and teaching me how to do science. I learned a lot from you and for that I will always be grateful. I am very sad that Josu Ceberio will not be officially recognized as my third co-supervisor, due to some rule in the uni. Sorry about that.
Thank you to Emma Hart and Leni Le Goff for kindly hosting me in the Edinburgh Napier University. I had a really good time there, I learned a lot, and it was very cool that I got to play with real robots (sadly, only for a short time), present my work at the conference in York and write a paper. We will hopefully be able to collaborate in the future. I also want to thank Kyrre Glette, Frank Veenstra and Tønnes Nygaard for hosting me at the University of Oslo. This was a great opportunity to learn about co-optimization of design and control of robots, and I think the paper we are doing will be pretty cool when finished. Hopefully this will not be our last collaboration. Thank you to Judith Echevarrieta as well, for being a beta tester of my supervision skills and collaborating with me on a paper.
I also want to thank Ander Carreño for all the lifehacks on the PhD and academia in general, and Verónica Álvarez for all the help regarding the official paperwork. It really made my life easier. I wish you both the best.
Last but not least, I want to thank the BCAM staff and specially Miguel Benitez, Agurtzane Hurtado and Irantzu Elespe for the administrative work they have done on my behalf.
I have had the difficult task of selecting people who explicitly appear in this section. Nevertheless, if you feel that you should have been here and you are not, I sincerely apologize, and please, take a part of this last Thank You.

Contents

0

Background

This chapter introduces the key concepts considered in the rest of the dissertation, and gives a brief overview of the research carried out.

0.1 Introduction

We begin by introducing the concept of optimization problem [Boyd and Vandenberghe, 2004a]. An optimization problem involves finding the best solution among a set of candidate solutions.

Definition 1 *(Optimization Problem) We define an optimization problem* $\boldsymbol{\mathcal{P}}$ *as*

$$\underset{\sigma \in S}{\operatorname{argmin}} f(\sigma)$$

where f *is the objective function and* S *is the solution space.*

Without loss of generality, a minimization setting can be assumed (otherwise consider the problem with the inverse objective function $-f$). In addition, in this thesis, we only consider *non-convex* [Boyd and Vandenberghe, 2004a, Hiriart-Urruty and Lemaréchal, 1996], optimization problems.

Depending on the properties of the solution space, optimization problems can be classified into continuous and combinatorial problems. The solution space in continuous problems is a subset of $\mathbb{R}^n$, and the solution space in combinatorial problems is a countable set. For instance, permutation based optimization problems are optimization problems where the solution space is the set of all permutations of a certain size n ($n!$ permutations in total). Optimization problems can also be

mixed, for example when the solution space has both continuous and discrete components. In this thesis, we considered these three types of optimization problems, and in the following, we give a few example problems that are also featured in the thesis.

Examples

Discrete. The Traveling Salesman Problem [Flood, 1956] (TSP) is a well known combinatorial optimization problem. Given a set of n cities and the distances between each pair of cities, the goal of the TSP is to find the shortest route that visits all the cities exactly once and returns to the initial city. The solution space is the set of all possible routes.

Continuous. A neural network [Goodfellow et al., 2016] is a mathematical function that can be trained to map inputs into outputs with examples. The training process of a neural network involves optimizing its performance in classification, prediction, or other applications. Specifically, the optimization problem is to find a set of weights and biases (continuous parameters) that minimize a previously defined loss function.

Mixed. Optimizing a wind turbine involves finding a design with desirable properties, such as damage resistance, low mass and the amount of energy it can produce. Zarketa-Astigarraga et al. [2023] proposed a wind turbine design problem with 6 continuous variables—corresponding to the length and the width of the blades of the turbine, among other things—and a discrete variable representing the number of blades. The objective value is computed via simulation, and is equal to the amount of energy that the wind turbine obtained in a specific set of environments.

0.1.1 Optimization Algorithm

Definition 2 *(Optimization Algorithm) Let $\mathfrak{P}$ be an optimization problem, and f the objective function of optimization problem $\mathfrak{P}$. An optimization algorithm is a procedure to find a solution σ that minimizes the objective value $f(\sigma)$.*

An optimization algorithm chooses a solution σ in a solution space S that minimizes an objective function f. When we apply the optimization algorithm in an optimization problem, the result is the best found solution σ_{best} (an item from the solution space S). The objective value of σ_{best} varies depending on many factors, including how much computation resources are used. Specifically, the optimization algorithm is executed until a stopping criterion halts the optimization process, usually defined as a maximum runtime in seconds or a maximum number of evaluated solution (denoted as t_{max}). We denote the objective value of the best found solution as the *Performance of an Optimization Algorithm*:

Definition 3 *(Performance of an Optimization Algorithm) Let $\mathfrak{P}$ be an optimization problem, t_{max} a computation budget, f the objective function of an optimization problem $\mathfrak{P}$. The performance of the Optimization Algorithm is defined as the objective value we obtain when applying the optimization algorithm in the problem $\mathfrak{P}$ with the computation budget t_{max} as the stopping criterion.*

In general, the performance of an optimization algorithm is not a deterministic value, instead, a different result (σ_{best}) might be observed each time it is executed (although it depends on the type of optimization algorithm). There are different factors that can make the results of an algorithm stochastic, such as the initialization. This is the case for estimation of distribution algorithms [Larrañaga and Lozano, 2001a] and gradient descent algorithms [Glorot and Bengio, 2010], for example.

The Performance of an Optimization Algorithm A can be modeled as a random variable, such that observing a sample of the random variable is the result of applying the optimization algorithm to the problem. In practice, and for every algorithm considered in this thesis, this randomness is usually generated by random number generator. Setting an initial random seed is sufficient to guarantee a deterministic result when applying the heuristic. The random seed needs to determine both the initialization and the internal state of the heuristic. Tying the randomness of an heuristic to the initial seed makes the results reproducible.

0.1.2 Taxonomy of Optimization Algorithms

There are many types of optimization algorithms, and they can be classified according to several criteria. In the following, we go over these classifications and we provide examples of each. We also state what type of algorithms are considered in the rest of the thesis.

0.1.2.1 Exact & Heuristic

Firstly, we can classify optimization algorithms based on wether they are able guarantee the optimality of the best found solution. Exact algorithms are able to find the optimal solution σ_{opt} such that, for every other solution σ in the solution space, $f(\sigma) \geq f(\sigma_{opt})$ [Wolsey, 2020]. Exact solvers guarantee that every solution in the search space is worse than the current best found solutions, which might not be possible to do in a limited of time. In addition, exact algorithms often are based on the special properties of the problems (for instance, the simplex algorithm takes advantage of the convexity in linear programming problems [Bertsimas and Tsitsiklis, 1997]).

As an alternative, heuristic algorithms are designed to find a good solution in a short period of time. Heuristic algorithms are usually considered on problems in which an exact solver would take too long. However, they are unable to guarantee the optimality of the solution. In this thesis, we only consider heuristic algorithms.

0.1.2.2 Constructive heuristics & Iterative heuristics

Constructive algorithms sequentially build a solution by combining solution components. In contrast, iterative algorithms iteratively propose candidate solutions to be evaluated. A heuristic algorithm is a procedure that proposes the next candidate solution to be evaluated, given the previously evaluated solutions and their objective values. Therefore, iterative algorithms work with complete solutions. In this thesis, we focus on iterative algorithms.

0.1.2.3 Gradient-based heuristics & Gradient-free heuristics (for continuous problems)

The gradient is a vector defined for differentiable functions that points towards the direction in the solution space with the steepest ascent. Gradient-based heuristics take repeated steps in the direction of the steepest descent (opposite of the gradient) and they require the objective function to be explicitly defined and differentiable [Boyd and Vandenberghe, 2004b]. Gradient-free heuristics, on the other hand, do not require the gradient to optimize, and are useful for problems in which the gradient is unavailable. In this thesis, we considered both gradient-based and gradient-free algorithms, although we primarily focused on the latter.

0.1.2.4 Deterministic heuristics & Stochastic heuristics

We use the term "stochastic heuristic" to refer to the fact that some heuristic algorithms will produce a different result each time they are executed, even if the same optimization problem and stopping criterion are considered. In contrast, deterministic optimization algorithms obtain the same result every time they are applied to the same problem. In this thesis, we consider only stochastic optimization algorithms.

0.1.2.5 Examples

Exact. The *Concorde TSP solver* [Applegate et al., 2001] is one of the best exact TSP solvers. This solver operates using the integer programming formulation of the TSP, and is based on the branch and cut algorithm [Wolsey, 2020].

Stochastic Iterative Heuristic. The *2-opt local search heuristic* [Johnson and McGeoch, 1997] is a classical and intuitive heuristic for the TSP. Starting with a random solution, the heuristic checks if the tour length would decrease by exchanging a pair of cities in the route. This is done for every possible city pair, unless an exchange improves the previous route. If none of the exchanges improve the tour, then the solution is reinitialized randomly.

Deterministic Constructive Heuristic. The *nearest neighbor greedy constructive heuristic* for the Traveling Salesman Problem starts with a city and iteratively adds the closest by city to the route, until all the cities have been visited [Goodman and Hedetniemi, 1977].

Gradient-based Stochastic Iterative Heuristic The *gradient descent algorithm* is one of the most simple gradient-based algorithm for continuous optimization problems. It involves starting with a random initial solution and iteratively taking steps in the direction of the steepest descent. A more complex version of this simple algorithm is currently used in Deep Learning to train neural networks [Goodfellow et al., 2016] for a variety of tasks such as classification and reinforcement learning.

Stochastic Iterative heuristic for direct policy learning The covariance matrix adaptation evolution strategies algorithm is an iterative stochastic optimization algorithm for continuous optimization problems that, given a set of initial solutions, samples new solutions from a distribution that is adjusted with the objective value of the previous solutions. Heidrich-Meisner and Igel [2008] proposed using this optimization algorithm for direct policy search, and showed it to be more robust than gradient-based approaches for the double pole balancing task Gomez et al. [1999].

0.2 Motivation, objectives and overview

The principal objective of this thesis is to compare and improve stochastic iterative heuristics. In this sense, we study these types of algorithms from different angles.

A. Chapter 1

In order to ensure a fair comparison between two different algorithms, it is necessary to ensure that an equal amount of computational resources is allocated to both. Two algorithms use the same amount of computational resources when they have the same runtime in the same machine. Usually, this is achieved by executing both algorithms on the same machine and setting a common maximum runtime as stopping criterion.

B. Probability type I error

Suppose the aim of the comparison is to determine whether a new algorithm is better than a previously published one with certain guarantees. In that case, it is essential to control the probability of falsely identifying the new algorithm as performing better than the other one. This probability is known as the probability of type I error in hypothesis testing, and it refers to the rejection of a true null hypothesis (finding that the performance of the new algorithm is significantly better than the other one when it is not the case). By controlling the probability of type I error, we can ensure that the conclusions are based on reliable evidence, and we can avoid erroneous conclusions.

C. Reproducibility in optimization

Even though reproducibility is easier in optimization and computer science—when compared to other fields like psychology or biology—it is not always possible to reproduce previous results and execute them in the same machine. For instance, certain optimization algorithms are very costly to execute [Sharir et al., 2020] (such as training the GTP-3 [Brown et al., 2020] language model). Furthermore, as stated by Hutson [2018], many works do not include the code to reproduce the experiments. These and other issues limit the reproducibility of the comparison of algorithms, as it is not always possible to execute all the algorithms being compared in the same machine.

Addressing this issue, Domínguez and Alba [2012] proposed adjusting the runtimes of the algorithms by assigning a shorter CPU runtime to the algorithms executed in machines with faster CPUs, thus making the execution of algorithms in different machines comparable. To estimate the CPU capabilities of each machine, they proposed using the *dhrystone2* score Weicker [1988]. However, the methodology introduced by Domínguez and Alba [2012] is limited in three ways. Firstly, there is neither theoretical nor experimental justification for the prediction model of the equivalent runtime. Secondly, their prediction model is fixed and cannot be adjusted control the probability of predicting a runtime that is longer than intended. And thirdly, their methodology does not take into account the probability of predicting an equivalent runtime longer or shorter than the true equivalent runtime, and thus, may introduce undesired biases to the comparison of the performance of algorithms.

With this in mind, these are the contributions of Chapter 1:

Contributions Chapter 1:
We develop a methodology to adjust the runtime to account for the difference in machine speed. We propose a comparison method for algorithms executed on different machines that can control the probability of observing a false difference in the performance of the algorithms (due to the algorithms being executed on different machines).

D. Chapter 2

Stochastic optimization algorithms obtain a different result each time they are executed. As a consequence, simple statistics like the median or standard deviation can sometimes not be enough to summarize the performance of an algorithm. Relying on simple statistics like the mean and the variance may lead to erroneous conclusions, as different datasets can still have the exact same statistics, but still be very different distributions [Matejka and Fitzmaurice, 2017, F. J. Anscombe, 1973].

Showing that the performance of one of the algorithms stochastically dominates [Quirk and Saposnik, 1962] the other one is enough to decide that one of the algorithms is better than the other. However, this is a very strong requirement and it is often not fulfilled in practice. Therefore, going beyond summary statistics and comparing the performances as random variables can often lead to more insightful conclusions.

Existing methods considered for the comparison of two samples of the performance of two algorithms have certain limitations. For instance, in null hypothesis tests, the p-value does not give information about the magnitude of the difference [Benavoli et al., 2017, Calvo et al., 2019a]. In addition, rejecting the null hypothesis does not always mean that there is evidence in favor of the alternative hypothesis: it just means that the observed statistic (or a more extreme statistic) is very unlikely when the null hypothesis is true. As an alternative [Benavoli et al., 2017, Calvo et al., 2019a] to the limitations of null hypothesis test, Bayesian analysis has been proposed in the context of algorithm performance comparison. Bayesian analysis [Gelman, 2014, Bernardo and Smith, 2009] estimates the probability distribution of an hypothesis being true, conditioned to the observed data. This probability can then be summarized in the simplex plot [Benavoli et al., 2017, 2014], through the probabilities of $A > B$, $A = B$ or $B > A$, where A and B are the performances of two algorithms respectively. However, the simplex plot also has some limitations, for instance, this probabilities are once again summarizing the comparison through summary statistics, disregarding the distribution of the performances.

This motivates to the second contribution of the thesis, which is addressed in Chapter 2:

Contributions Chapter 2:
We illustrate with examples that comparing two samples with summary statistics can be misleading. We propose eight desirable properties for random variable measures in terms of suitability to compare two samples through stochastic dominance. We propose a measure of stochastic dominance for random variables. We propose a graphical representation to compare two samples through stochastic dominance with three desirable properties: i) it contains a graphical decomposition of the probability of $A > B$ and stochastic dominance, ii) unlike statistical tests, it measures and distinguishes between a high uncertainty and low magnitude in difference; and iii) it models the uncertainty of the estimate through a 95% confidence band.

E. Chapter 3

Certain heuristics for permutation problems such as local search algorithms (like the 2-opt heuristic for the TSP mentioned above [Johnson and McGeoch, 1997]) or estimation of distribution algorithms [Larrañaga and Lozano, 2001a], rely on the definition of a solution (permutation) neighborhood. Intuitively, the neighborhood of a permutation is the set of solutions that are close to that permutation, i.e., the set of solutions that can be obtained when applying an operator (such as exchanging the position of two elements in the permutation) to the solution [Schiavinotto and Stützle, 2007]. The concept of neighborhood is also closely related to distances for permutations, such as the Ulam/Cayley distances [Ceberio et al., 2015a] (the minimum number of inserts/exchanges required to transform a permutation into another permutation). However, some distances for permutations are not defined via an operator, e.g., the Hamming distance (the number of items in a permutation that need to be replaced transform a permutation into another permutation).

In this context, the performance of optimization algorithms that rely on operators/distances/neighborhoods for permutation problems (from now on denoted as *distances*) is influenced by which *distance* is considered. Moreover, by studying the changes on the objective function induced by the *distance*, it is possible to improve optimization algorithms for permutation problems. For instance, Ceberio et al. [2015b] discovered that certain solutions in the insert neighborhood [Schiavinotto and Stützle, 2007] in the Linear Ordering Problem [Martí and Reinelt, 2011] are guaranteed to not be local optima, reducing the size of the neighborhood that needs to be evaluated by a local search procedure.

Contributions Chapter 3:
We analyze and compare the properties of permutation distances on the objective function of the Quadratic Assignment Problem [Koopmans and Beckmann, 1957] (QAP), both theoretically and experimentally. We look at how the Hamming distance induces more suitable neighborhoods for assignment type problems, when compared with other distances for permutations. We also carry out a comparison of estimation of distribution algorithms for the QAP.

F. Chapter 4

Problem analysis methods such as Fitness Landscape Analysis [Ochoa and Malan, 2019] and Local Optima Networks [Ochoa et al., 2014] can help us understand optimization problems (or at the very least, gain some intuition). Analyzing optimization problems can also help us choose the best algorithm for an optimization problem, which is known as the algorithm selection problem [Rice, 1976]. It can also be useful to choose wether to switch to another optimization algorithm during the optimization process [Kostovska et al., 2022, Vermetten et al., 2023].

One of the limitations of the problem analysis methods in the literature is that they are particular to a solution space. For example, local optima networks assume a combinatorial optimization problem [Ochoa et al., 2014], or exploratory landscape analysis [Mersmann et al., 2011a] assumes a continuous optimization problem. A way to overcome this limitation is to study the transferability via hyper-heuristics[1] Burke et al. [2003], as proposed by Hong et al. [2018]. Transferability Hong et al. [2018] is defined as the loss of performance observed when a hyper-heuristic is initially trained on one optimization problem and then applied to a different problem, when compared to training and applying the hyper-heuristic on the same problem. Hong et al. [2018] proposed measuring transferability as the average objective value of the hyper-heuristic when training and testing on two continuous optimization problems.

[1] A hyper-heuristic is a methodology that focuses on selecting/designing heuristics to solve a given problem. It operates at a higher level compared to traditional heuristics, which typically apply heuristics to optimization problems directly Burke et al. [2003]. In this context, we refer to hyper-heuristic algorithms that "learn" to adapt to a optimization problem after an initial training phase.

Contributions Chapter 4:
We propose two multi-domain problem analysis method based on a multi-domain hyper-heuristic framework. The introduced analysis methods are the same regardless of the problem domain, and we show that they are able to identify similar and different problems within a set of optimization problems, both in the continuous and discrete domains. In addition, we improve Hong et al. [2018]'s methodology in four key aspects. Firstly, by defining transferability with ranks instead of objective values, we can compare across different problems (the magnitude of transferability is the same for every problem). Secondly, by repeating several measurements of transferability and computing the average ranks, we can distinguish between noise and the actual difference in performance (this is not possible with average objective values). Thirdly, we experimentally show that the analysis carried out with our approach is correlated with the properties of the optimization problems, which suggests that the proposed technique is useful for finding similarities and differences in the properties of optimization problems. Finally, we show that our approach works in both combinatorial and continuous domains, while Hong et al.'s approach Hong et al. [2018] was only shown to work in the continuous domain.

G. Chapter 5

In optimization problems, and particularly in policy learning tasks involving evaluations in the physical world or costly simulations, lengthy evaluation times are common. To address this limitation, it is possible to stop the evaluation early when we suspect that the objective value is unlikely to improve. For example, when learning to control a robot in a simulation, if the robot gets stuck (it does not move) it is useful to stop the evaluation early [Le Goff et al., 2021]. However, most early stopping methods in policy learning are problem specific (in the previous example, we need to be able to detect that the robot is not moving, which might not be trivial).

General (using no domain specific knowledge) early stopping has been successfully applied on Hyperparameter tuning [Jamieson and Talwalkar, 2016, Hutter et al., 2019, Li et al., 2017, Falkner et al., 2018a]. In addition, Bongard [2011] proposed a general early stopping method for policy learning, however, Bongard [2011]'s method assumes a specific definition of the objective function, as well as the use of a specific optimization algorithm.

We develop the following contributions in Chapter 5:

Contributions Chapter 5:
We propose propose GESP: a general early stopping criterion for direct policy search that does not require problem specific information and only takes into account the objective function to stop the evaluations early. We compare general early stopping with problem specific early stopping, and find it to perform similarly, while being more general.

0.3 Notation

In the following table, we summarize the most important terms and notation considered in the thesis.

Concept	Notation	Explanation
Optimization Problem	$\mathfrak{P}$	Defined in Definition 1.
Performance of an Optimization Algorithm	A	Defined in Definition 3.
Computational Resources	t	Computational resources used so far during the optimization of a problem. It is measured in either seconds, simulation steps or function evaluations.
Computational Resources	t_{max}	The stopping criterion (computation budget) for the optimization algorithm.
Samples of the Performance	$a_1, a_2, ...$	Samples of the random variable A.
Solution	σ	A candidate solution to an optimization problem.
Best Solution	σ_{best}	The candidate solution with the best objective value found so far during an execution of an optimization algorithm.
Objective Function	f	The objective function of an optimization problem.
Objective Value	$f(\sigma)$	The result of evaluating a solution σ with the objective function f.
Probability Density of the Performance	g_A	The probability density function of the random variable A.
Cumulative distribution	G_A	The cumulative distribution of the random variable A.

Table 0.1: A summary of the notation in the thesis.

1

On the Fair Comparison of Optimization Algorithms in Different Machines

The computational resources used by an optimization algorithm can be understood as the number of computational operations that are carried out. A higher number of operations implies a higher number of function evaluations, which in turn implies a better or equal objective value. Hence, a fair comparison of two or more optimization algorithms requires the same computational resources to be assigned to each algorithm.

When a maximum runtime is set as the stopping criterion, all algorithms need to be executed in the same machine if they are to use the same resources. Unfortunately, the implementation code of the algorithms is not always available, which means that running the algorithms to be compared in the same machine is not always possible. And even if they are available, some optimization algorithms might be costly to run, such as training large neural-networks in the cloud. Consequently, it is not always possible to execute all the algorithms in the same machine.

On the other hand, in the field of optimization, new optimization methods are constantly being proposed. To demonstrate the effectiveness of a *new algorithm*, a comparison is made against the state-of-the-art approaches. In order to assert that the *new algorithm* surpasses the state-of-the-art, it is necessary to control the probability of type I error, which refers to the probability of falsely asserting that the *new algorithm* is better that the state-of-the-art, when in fact, it is not. Executing the algorithms in different machines can increase this probability, as the algorithms will not be compared with the same computational resources.

In order to avoid an increase of the probability of type I error in this setting, it is crucial to ensure that the computational resources allocated to the *new algorithm* are lower than or equal to the resources considered in the state-of-the-art-approach. This control is necessary to provide robust evidence of the *new methods*'s superiority over the existing approaches.

More specifically, in this chapter, we introduce a methodology to compare the performance of a new optimization algorithm B with a known algorithm A in the literature if we only have the results (the objective values) and the runtime in each instance of algorithm A. The proposed methodology has two parts. Firstly, we propose a model that, given the runtime of an algorithm in a machine, estimates the runtime of the same algorithm in another machine such that the slower machine is compensated with extra runtime. Although this idea has already been explored before [Domínguez and Alba, 2012], our model presents two improvements with respect to previous work. Firstly, the prediction model can be adjusted so that the probability of estimating a runtime longer than what it should be is arbitrarily low. Secondly, we introduce an adaptation of the one-sided sign test that uses a modified p-value and takes into account that probability. These two improvements allow the proposed model to avoid increasing the probability of type I error associated with executing algorithms A and B in different machines.

1.1 Introduction

A different use of computational resources can lead to observing false differences between the performance of optimization algorithms. Appendix 7.1.1 presents an illustrative example. We say that two algorithms use the same amount of computational resources when they both take the same time to complete in the same machine. Usually, this is achieved by executing both algorithms on the same machine and setting a common maximum runtime as stopping criterion.

Over the last few decades, the computational capabilities of computers have significantly increased [Nordhaus, 2007]. This means that code executed a decade ago is expected to run faster on current hardware. Additionally, in some fields of optimization such as natural language processing, good performing models like the GTP-3 [Brown et al., 2020] are currently being trained with a lot of computational power. Unfortunately, training these types of models is often economically unviable for most researchers [Sharir et al., 2020]. Furthermore, as stated by Hutson [2018], many works do not include the code to reproduce the experiments. These and other issues limit the reproducibility of the comparison of algorithms, as it is not always possible to execute all the algorithms being compared in the same machine.

One way to overcome this limitation is to adjust the runtime of one of the algorithms being compared such that the algorithm executed on the slower machine is compensated with extra computational time. Let us now look at a practical example. Let us imagine that a researcher reads a paper in which algorithm A is executed in machine M_1, taking time t_1. Now, the researcher wants to compare a new algorithm, B, with A, but has no access to algorithm A nor to machine M_1. Instead, the researcher only has access to machine M_2 and algorithm B. In this case, the researcher can execute algorithm B in machine M_2 for time t_2 and compare the result with the known results of algorithm A. The runtime t_2 needs to be set in such a way that both algorithms are given the same computational resources. To achieve this, t_2 needs to be equal to the equivalent runtime: the time it takes to replicate in machine M_2 the exact optimization process that was carried out in machine M_1 (with algorithm A).

The exact equivalent runtime t_2 can be obtained if the exact optimization process that was carried out in machine M_1 is replicated in machine M_2. This implies executing algorithm A in machine M_2, which defeats the purpose of using an equivalent runtime. Fortunately, an estimation of the equivalent runtime $\hat{t}_2$ can be used instead. The estimation is carried out taking into account the computational capabilities of machines M_1 and M_2, denoted as s_1 and s_2 in the rest of the chapter. Table 1.1 offers a brief overview of the terms used in the chapter.

A summary of the notation of the chapter

Name	Notation	Explaination
Optimization algorithm A	A	The optimization algorithm that is not executed in the comparison. Instead, already published results of this algorithm are used in the comparison.
Optimization algorithm B	B	The optimization algorithm that is executed to obtain the results to be compared.
Machine M_1	M_1	The machine in which algorithm A was executed. We have no access to this machine.
Machine M_2	M_2	The machine in which algorithm B is executed to obtain the results used in the comparison.
The score of machine M_1	s_1	A measure proportional to the computational capability of machine M_1.
The score of machine M_2	s_2	A measure proportional to the computational capability of machine M_2.
The runtime of A in machine M_1	t_1	The stopping criterion (in terms of maximum runtime) that was used in the execution of algorithm A in machine M_1.
Equivalent runtime of A in machine M_2	t_2	The time it takes to replicate in machine M_2 the exact optimization process (with algorithm A) that took time t_1 in machine M_1.
Estimated equivalent runtime of algorithm A in machine M_2	$\hat{t}_2$	An estimation of the equivalent runtime t_2. This value is used as the stopping criterion of algorithm B, which is executed in machine M_2.

Table 1.1: A summary of the concepts considered in the chapter.

A. Related work:

Domínguez and Alba [2012] proposed adjusting the runtimes of the algorithms by assigning a shorter CPU runtime to the algorithms executed in machines with faster CPUs, thus making the execution of algorithms in different machines comparable. To estimate the CPU capabilities of each machine, they proposed using the *dhrystone2* score [Weicker, 1988].

The methodology introduced by Dominguez et al. is limited in three ways. Firstly, there is neither theoretical nor experimental justification for the prediction model of the equivalent runtime. Secondly, their prediction model is fixed and cannot be adjusted control the probability of type I error. And thirdly, their methodology does not take into account the probability of predicting an equivalent runtime longer or shorter than the true equivalent runtime, and thus, may introduce undesired biases to the comparison of the performance of algorithms. Without a corrected statistical model, it is not possible to take into account this probability

and control the probability of type I error. In the context of performance comparison of optimization algorithms (with null-hypothesis statistical tests), a type I error is defined as finding a statistically significant difference in the performance of the algorithms, when in reality, there is none. Making a type II error, on the other hand, means not finding a statistically significant difference in the performance of the algorithms, when in reality, the performances are different. In this context, making a type II error is preferred to a type I error: falsely concluding a nonexistent difference in the performance of the algorithms is worse than not finding a statistically significant difference in the performance of the algorithms.

In fact, failing to reject the null hypothesis does not imply evidence in favor of the null hypothesis. Instead, it only shows a lack of evidence against it. In the context of algorithm benchmarking, failing to reject the null hypothesis does not mean that the performance of the algorithms is the same. When the null hypothesis is not rejected, the correct conclusion is that there is not enough evidence to show a statistically significant difference between the performance of the algorithms. Therefore, a type II error just means that additional experimentation is needed to verify an existing difference in the performance of the algorithms, which is not an erroneous conclusion in itself.

B. Proposed methodology:

Inspired by the work of Domínguez and Alba [2012], we propose a methodology to statistically assess the difference in the performance of optimization algorithms executed in different machines. Specifically, the proposed methodology can be used to show that an algorithm B performs statistically significantly better than another algorithm A, without executing A and, instead, using the available results of A in terms of the objective function value and the runtime in each instance. To that end, we propose a conservative methodology in which the probability of giving algorithm B an unfairly longer time is kept in check by i) proposing a two-parameter estimation[1] of the equivalent runtime with an arbitrarily low probability of estimating an unfairly longer runtime and ii) by modifying the one-sided sign test [Conover, 1980] so that it takes this probability into account.

Alongside this chapter, we present a tutorial on how to apply the proposed methodology. This tutorial and the code of all the experimentation is available in our GitHub repository[2]. Besides, we also give two examples of how the methodology

[1] The estimation considered in this chapter is intentionally conservative and tends to estimate shorter than equivalent runtimes. With this, we are able to keep the probability of type I error in check, while increasing the probability of type II error. In the context of algorithm benchmarking, a type I error is worse than a type II error.

[2] Repository available in https://github.com/EtorArza/RTDHW.

is applied. It is noteworthy that applying the proposed methodology to compare algorithms in different machines does not involve executing any additional code.

The rest of the chapter is organized as follows: The next section describes and motivates a two-parameter model proposed to estimate the equivalent time. Section 1.3 presents the modifications made to the sign test to overcome the limitations introduced by the execution of the algorithms in different machines. Afterward, in Section 1.4 we introduce two examples in which we apply the proposed methodology. Finally, Section 1.6 concludes the chapter and proposes some research lines for future investigation.

1.2 The estimation model of the equivalent runtime

Given i) an optimization algorithm, ii) a machine, iii) a problem instance, iv) a stopping criterion and v) a random seed number, executing the optimization algorithm will produce a specific sequence of computational instructions. This sequence is completely determined by these five parameters. We call this sequence of instructions that is reproducible in any machine the *optimization process*. By recording the optimization process carried out with these parameters, we can later reproduce the exact optimization process in another machine. Notice that reproducing the optimization process will take a different time in each machine, even though the final result is the same (because the executed sequence of instructions is the same). We say that the runtimes required to replicate the same optimization process in different machines are equivalent.

Definition 4 *(Optimization process)*
Let M be a machine, A an optimization algorithm, i a problem instance, t_1 a stopping criterion, and r a positive integer (the seed for the random number generator). We define the optimization process $\rho(M, A, i, t_1, r)$ as the sequence of computational instructions carried out when optimizing instance i with algorithm A and seed r in machine M_1 with stopping criterion t_1.

The aim is to compare algorithm A executed in machine M_1, with algorithm B, executed in machine M_2. A fair comparison can be carried out by estimating the time it takes to replicate $\rho(M_1, A, i, t_1, r)$ in machine M_2 and using the estimated value as the stopping criterion for algorithm B in machine M_2. We will sometimes denote the optimization process $\rho(M_1, A, i, t_1, r)$ as ρ, for the sake of brevity.

Definition 5 *(Runtime of an optimization process)*
Let M be a machine and ρ an optimization process. We define the runtime of ρ in

M, denoted as $t(M, \rho)$, as the time it takes to carry out the optimization process ρ in machine M.

Considering the above definitions, it follows that, $t(M_1, \rho) = t_1$.

Definition 6 *(Equivalent runtime)*
Let M_1, M_2 be two machines, ρ an optimization process and $t(M_1, \rho)$ and $t(M_2, \rho)$ the times required to run ρ in M_1 and M_2 respectively. Then, we say that $t(M_2, \rho)$ is the equivalent runtime of $t(M_1, \rho)$ for machine M_2.

From here on, we will denote $t(M_2, \rho)$, the equivalent runtime of $t_1 = t(M_1, \rho)$ in machine M_2, as t_2. Given t_1 (the runtime of optimization process ρ in a machine M_1), in the following, we will propose a model to estimate t_2 (the equivalent runtime in another machine M_2).

Assumption 1 *(Constant ratio of the runtime of two optimization processes)*
Let ρ, ρ' be two optimization processes. Then, we assume that:

$$\frac{t(M_2, \rho)}{t(M_2, \rho')} \approx \frac{t(M_1, \rho)}{t(M_1, \rho')}$$

for any two machines M_1 and M_2.

We assume that the ratio of the runtime of two different optimization processes is constant with respect to the machine in which it is measured. In Appendix 7.1.2, we justify why this assumption is reasonable. This assumption is critical to the estimation model that will later be proposed. By using this assumption in the model, a prediction error is introduced. Therefore, we will later propose a correction to the model to control this prediction error.

Based on this assumption, we propose a model to estimate the equivalent runtime of an optimization process in a machine, given its runtime in another machine, as well as the scores[1] (relative to the computational capabilities) of both machines. Notice that in Assumption 1, we use a reference optimization process ρ' to estimate the equivalent runtime of the optimization process ρ. Any optimization process ρ' can be used as a reference. In the following, we will define an optimization process ρ' whose runtime we will be able to estimate with the scores s_1 and s_2 of the machines. This will allow the estimation of the equivalent runtime t_2 without executing any reference optimization processes, as shown in Figure 1.1.

[1] In this chapter, we denote an observed sample of the performance of an algorithm as "score".

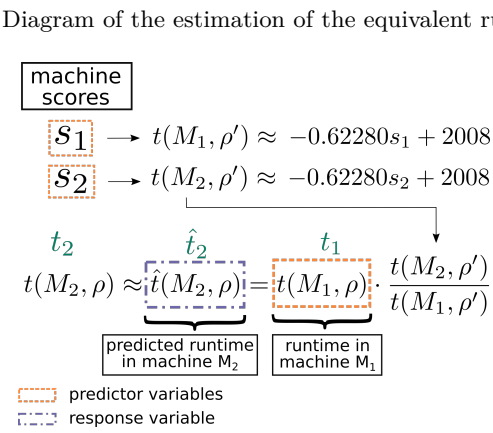


Fig. 1.1: Estimated equivalent runtime of ρ in machine M_2 (the response variable $\hat{t}(M_2, \rho)$). The estimation is carried out with three predictor variables: the machine scores s_1 and s_2 and $t(M_1, \rho)$.

Let us now define the optimization process ρ', whose runtime can be estimated. Recall that an optimization process is just a sequence of computational instructions that can be reproduced in any machine. Aiming to obtain a more diverse sequence of computational instructions, we define the optimization process ρ' as the computational instructions generated by consecutively executing 4 different optimization algorithms in 16 problem instances. Each of the 64 executions involves solving a permutation problem with an optimization algorithm, with a stopping criterion of a maximum of $2 \cdot 10^6$ evaluations (see Appendix 7.1.3 for details on the optimization problems and algorithms used).

The runtime of the optimization process ρ' in a machine can be estimated with its machine score. In this chapter, we measure the score of a machine (its computational capability) in terms of its PassMark single thread CPU score [1], although adapting the proposed methodology to other benchmarks is also possible. The advantage of using the PassMark score is that the PassMark website has a large collection of CPUs with their scores. In Figure 1.2, we show the machine score and runtime of ρ' for each of the 8 machines considered in this chapter (the list of

[1] The PassMark CPU score is one of the most popular CPU benchmark scores, with over 3500 CPUs listed on their website. We use the single thread score of this benchmark. A higher value of the score is associated with a better relative performance of the CPU. The PassMark scores change over time, as new CPUs and consumer demand for computation evolves. The scores considered in this chapter can be looked up in the file cpu_scores.md in our GitHub repo.

PassMark single thread score and the runtime ρ'

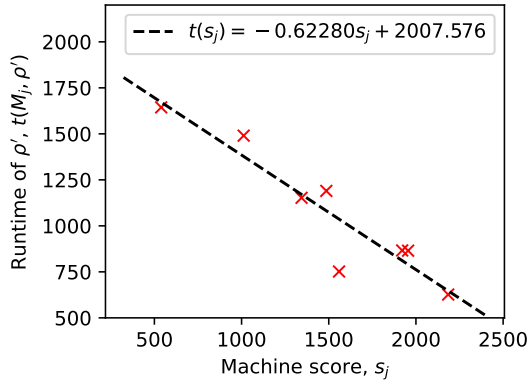


Fig. 1.2: The runtime of ρ', the optimization process used as a reference to define the regression model. Every point represents a different CPU, each with a different machine score and runtime of ρ' in this machine.

machines is available in Appendix 7.1.3). We see that the relationship between the runtime of ρ' and the machine score is linear. Every point in Figure 1.2 represents a different machine.

Based on the figure shown, we infer that i) linear regression is suitable to model the runtime of ρ' with respect to the machine score, and ii) the machine score has a good correlation with the runtime of ρ'. Note that using a linear estimation of ρ' has some limitations in its applicability that will be addressed in Section 1.5. With this in mind, we define the estimation of the runtime in a machine:

Definition 7 *(Prediction of the runtime of ρ' in a machine)*
Let M_j be a machine and s_j its machine score. Then, the runtime of ρ' in M_j is modeled as

$$t(M_j, \rho') \approx -0.62280s_j + 2008$$

where s_j is the score of machine M_j.

Considering together Assumption 1 and Definition 7, the equivalent runtime can be estimated as:

$$t_2 \approx \hat{t}_2 = t_1 \cdot \frac{-0.62280s_2 + 2008}{-0.62280s_1 + 2008} = t_1 \cdot \frac{3223.49 - s_2}{3223.49 - s_1} \tag{1.1}$$

where s_1 and s_2 are the PassMark single thread scores of the CPUs on machines M_1, M_2, respectively. Due to the approximation errors in Assumption 1 and Definition 7, the estimated equivalent runtime $\hat{t}_2 = \hat{t}(M_2, \rho)$ will often differ from the true equivalent runtime value $t_2 = t(M_2, \rho)$. This means that when using the estimated equivalent runtime as the stopping criterion for algorithm B, sometimes, the runtime will be longer or shorter than the runtime used by algorithm A.

1.2.1 Controlling the probability of predicting a runtime longer than the true equivalent runtime

To statistically assess the uncertainty associated to the comparison of the performance of algorithms A and B, in this methodology, we propose using a one-sided statistical test. Under this test, the alternative hypothesis states that the performance of algorithm B is better than the performance of algorithm A. As a result, a type I error (erroneously finding a statistically significant difference in the performance of A and B) can only be made when algorithm B performs better than A.

When a shorter runtime is estimated, algorithm B has an "unfairly" shorter stopping criterion for the optimization. This implies that the measured performance of B will be worse than or equal to the performance that would have been measured if the actual equivalent runtime were used. Consequently, taking into account the one-sided nature of the test, estimating a lower than actual runtime will not increase the probability of type I error (estimating a lower than actual runtime can never help algorithm B perform better than algorithm A). It might, however, increase the probability of type II error.

Making a type II error is better than making a type I error when comparing algorithm performance, because it is better to miss evidence that can adequately discriminate between two algorithms than to observe a false difference. For example, let us assume that someone publishes algorithm A with a certain performance. Another researcher proposes an algorithm B that is a variation of algorithm A. If a type II error is made, then B is actually better than A, but the researcher is not able to find enough evidence to support this, which is not in itself an erroneous conclusion. However, in a type I error, algorithm B is actually worse or equal to A but the researcher incorrectly identified algorithm B as the better algorithm, and this can be more detrimental to scientific progress.

To avoid drawing erroneous conclusions, we present a modification to Equation (1.1) so that the probability of estimating a longer time than the actual equivalent runtime stays under a percentage chosen by the user. We reformulate the unbiased estimator shown in Equation (1.1) to reduce the probability of estimating a runtime longer than the true equivalent runtime. The new biased estima-

tor is defined by multiplying the unbiased estimator with a correction parameter $\gamma \in (0, 1]$:

Definition 8 *(estimation of the equivalent runtime in machine M_2) Let ρ be an optimization process, M_1, M_2 two machines and t_1 the runtime of ρ in machine M_1. We compute $\hat{t}_2$, the estimate equivalent runtime of ρ for machine M_2 as:*

$$t_2 = t(M_2, \rho) \approx \hat{t}_2 = t_1 \cdot \frac{3223.49 - s_2}{3223.49 - s_1} \cdot \gamma$$

By adjusting γ, the probability of estimating a longer runtime than the equivalent runtime, $\mathcal{P}[\hat{t}_2 > t_2]$, can be reduced. However, adjusting γ implies that on average, a shorter runtime is predicted. With $\gamma = 1.0$, the original, unbiased estimator is obtained. A lower value of γ is associated to a lower probability of estimating a runtime longer than the true runtime. Specifically, the parameter γ is equal to $\mathbb{E}[\frac{\hat{t}_2}{t_2}]$: how much shorter the estimated equivalent runtime is than the true equivalent runtime on average. For example, when $\gamma = 0.5$, then the equivalent runtime used will be half of the equivalent runtime predicted by the unbiased estimator.

We estimated the relationship between γ and $\mathcal{P}[\hat{t}_2 > t_2]$ and we show the result in Figure 1.3. We computed this probability empirically in a cross validation setting. The exact procedure carried out to generate this figure is available in the file show_linear.py in our GitHub repository. In the following, we give additional details on the process carried out to compute the relationship between γ and $\mathcal{P}[\hat{t}_2 > t_2]$ shown in the figure.

The pseudocode of the following process is shown in Algorithm 1. Given a value of γ, we start by iterating over all the optimization processes and every possible pair of CPUs (Lines 3-5) and we leave them out of the training data (Lines 6-7). Then, we fit the linear regression in Definition 7 and Equation (1.1), but only with the CPUs and optimization processes in the training data (Line 8). The runtime of the left out optimization process in the machine with cpu1 is t_1 (Line 11). Now, we predict the equivalent runtime of the optimization process left out in the machine with cpu2 with the formula $\hat{t_2} \leftarrow t_1 \cdot \frac{\alpha - s_2}{\alpha - s_1} \cdot \gamma$, where α was fitted with the training data (Line 12). Finally, we measure the proportion of times in which the predicted equivalent runtime for the machine with cpu2, $\hat{t_2}$, was longer than the runtime of the optimization process in that machine, t_2.

Instead of considering the parameter γ, we can also think of the parameter $p_\gamma = \mathcal{P}[\hat{t}_2 > t_2]$. p_γ is the probability of estimating a longer than equivalent runtime associated to γ. It is probably more useful for the user to think of p_γ, because this is what is directly related to the increase of probability of making a type I error in algorithm comparison: a lower p_γ has an associated lower probability of

Algorithm 1: Compute the probability of estimating a runtime longer than the true equivalent runtime given γ in a cross validation setting

Input:
cpu_list: The list of all the CPUs used to fit the linear regression.
process_list: The list of all the optimization processes.
γ: The correction coefficient.
Output:
$\mathcal{P}[\hat{t_2} > t_2]$: The probability of estimating a runtime longer than the true equivalent runtime for the given γ.

```
1  total ← 0
2  longer_runtime_predicted ← 0
3  forall test_process in process_list do
4  |  forall cpu1 in cpu_list do
5  |  |  forall cpu2 in cpu_list \ {cpu1} do
6  |  |  |  train_cpus ← cpu_list \ {cpu1, cpu2}
7  |  |  |  train_processes ← process_list \ {test_process}
8  |  |  |  fit the linear regression in Definition 7 and Equation (1.1) with the
   |  |  |    cpu scores of train_cpus and the runtimes of the optimization
   |  |  |    processes in train_processes when executed in train_cpus.
9  |  |  |  s_1 ← cpu score of cpu1
10 |  |  |  s_2 ← cpu score of cpu2
11 |  |  |  t_1 ← runtime of test_process in the machine with cpu1
12 |  |  |  t̂_2 ← t_1 · (α − s_2)/(α − s_1) · γ, where α was adjusted in Line 8.
13 |  |  |  total ← total +1
14 |  |  |  if t̂_2 > t_2 then
15 |  |  |  |  longer_runtime_predicted ← longer_runtime_predicted +1
16 return longer_runtime_predicted / total
```

predicting an unfairly longer equivalent runtime. To obtain an unbiased prediction of the equivalent runtime, it is enough to consider the parameter $p_\gamma = 0.5$.

To make the computation of the equivalent runtime convenient for the user, we created a standalone (no dependencies) python script equivalent_runtime.py available in our GitHub repository. This script predicts the equivalent runtime with the formula in Definition 8, but considering the parameter p_γ instead of γ. To achieve this, the γ associated to p_γ is calculated first. Given p_γ the desired probability of estimating a runtime longer than the true equivalent runtime, s_1, s_2 the PassMark single thread score of machines M_1, M_2 respectively and t_1 the runtime in machine M_1, we can use this script to estimate the equivalent runtime. For example if $p_\gamma = 0.1$, $s_1 = 1540$, $s_2 = 1643$ and $t_1 = 15.0$, then we can get $\hat{t_2}$ with

```
python equivalent_runtime.py 0.1 1540 1643 15.0
```

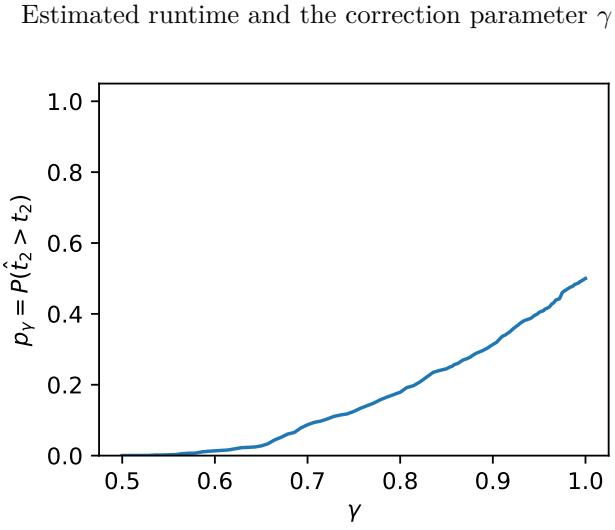


Fig. 1.3: The probability of estimating a runtime longer than the true equivalent runtime with respect to γ.

```
>> 11.461295
```

Even though the proposed model has an arbitrarily low probability of estimating a longer than actual equivalent runtime, this probability is not zero. In Section 1.3, we will propose a modification of the sign test that takes into account this probability and avoids an increase in the probability type I error.

1.2.2 Validation

We have introduced a methodology to predict the equivalent runtime for single thread optimization processes—a sequence of computer instructions that can be replicated in different machines—based on the PassMark single thread score. In the following, we experimentally validate that the proposed methodology works as intended. To this end, we try to answer the following two questions: A) Is using the equivalent runtime better (in the sense that it makes the comparison fairer) than using the same runtime in two machines? And B) does the method still work when considering other machines and optimization processes that were not present when the prediction model was fitted?

A. Predicting the equivalent runtime vs. using the same runtime

In this chapter, we proposed predicting the equivalent runtime with the PassMark score. In the following, we show that it is better than just using the same runtime in two machines. To do so, we compare the prediction error, measured as the ratio with respect to the true equivalent runtime, when using the centered estimator in Equation (1.1) of the equivalent runtime ($\hat{t}_2 = t_1 \cdot \frac{3223.49 - s_2}{3223.49 - s_1}$) vs. when estimating it as the same runtime as in the other machine ($\hat{t}_2 = t_1$).

We estimate the prediction error with these two methods for the 64 optimization processes and 8 machines considered in the calibration of the linear regression (see Appendix 7.1.3 for details). Once we have measured the ratio between the estimated runtime and the true equivalent runtime, we apply the loss function $f(x) = \text{abs}(\text{Log}_2(x))$, obtaining the *log deviation ratio*. With this loss function, a prediction that was three times the true equivalent runtime is assigned the same loss as a prediction that was a third of the true equivalent runtime. In addition, the log deviation ratio is easier to interpret: for example, a log deviation ratio of 0 means that the prediction was perfectly accurate, and a log deviation ratio of 1 denotes that the prediction was double or half the true value etc.

In Figure 1.4 we show the empirical distribution function of the log deviation ratio for *equivalent runtime* and *same runtime*. The results clearly point out that *equivalent runtime*, predicting the equivalent runtime with the centered estimator in Equation (1.1), consistently produces a lower (better) error when compared to using the same runtime in two machines *same runtime*.

B. Validation in other optimization processes and CPUs

Each time we predict the equivalent runtime with the centered estimator in Equation (1.1), we expect the prediction to be either higher or lower than the true equivalent runtime. Controlling this prediction error is one of the key challenges of the proposed methodology, and allows the user to predict an equivalent runtime with a desired probability of being higher than the true equivalent runtime. However, since we fitted this estimator with a set of optimization problems and CPUs (described in detail in Appendix 7.1.3), we need to validate the prediction with a different set of CPUs and optimization processes.

Validation CPUs and optimization processes: The four optimization processes for the validation experiment are enumerated below. These four optimization processes are very different from the optimization processes used to fit the estimator.

1. Find the first 10^6 prime numbers.

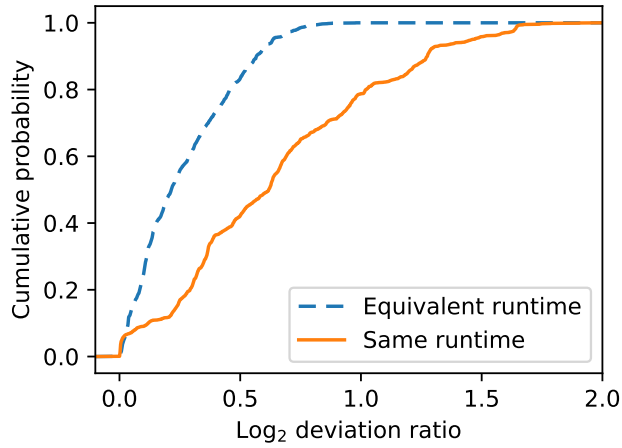


Fig. 1.4: A comparison in estimation error of the predicted equivalent runtime (Equivalent runtime) and simply using the same runtime in both machines (Same runtime) with respect to the true equivalent runtime. The estimation error is measured as the log deviation ratio of the prediction of the equivalent runtime with respect to the true equivalent runtime. For example, a log deviation ratio of 0 means that the prediction was perfectly accurate, and a log deviation ratio of 1 denotes that the prediction was double or half the true value etc.

2. Finding magic squares [Dougherty, 2014].
3. Solving the knapsack problem [GeeksforGeeks, 2022].
4. Solving the N queens problem [GeeksforGeeks, 2022].

The CPUs of the machines considered in the validation experiment are listed below:

CPU model name	**PassMark score**
Intel(R) Xeon(R) CPU @ 2.20GHz	1383
Intel(R) Core(TM)2 Duo CPU P9600 @ 2.53GHz	1075
Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz	1779
Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz	1840
Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz	1511

Now, we compare the log deviation ratio of the centered estimator (Equation (1.1)) both for the optimization processes and CPUs used to fit the estimator, and these new validation optimization processes and CPUs. The empirical distribution function of the log deviation ratio is shown in Figure 1.5. The error obtained with the

CPUs and optimization processes used to fit the estimator (from now on *Train*) is a lot smoother than with the validation CPUs and optimization processes (from now on *Validation*). However, this is to be expected because *Train* contains a larger amount of both optimization processes and CPUs. In addition, notice that for most of the x-axis, the error of *Validation* is lower (better) than the error of *Train*. This can also be explained by the variance of the error *Validation* being larger due to the smaller amount of CPUs and optimization processes.

In any case, both errors are very similar and close to each other, and this implies that the proposed methodology is indeed applicable to different CPUs and optimization processes.

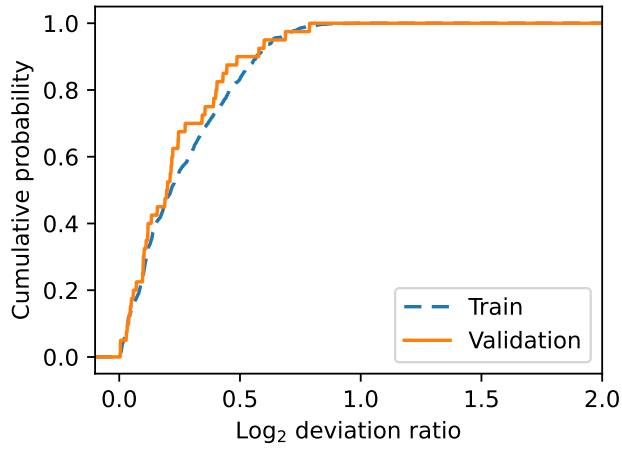


Fig. 1.5: A comparison in estimation error of the equivalent runtime with the centered estimator. The estimation error for the optimization processes and CPUs used to fit the estimator (Train), and these new validation optimization processes and CPUs (Validation) is compared. The estimation error is measured as the log deviation ratio of the prediction of the equivalent runtime with respect to the true equivalent runtime. For example, a log deviation ratio of 0 means that the prediction was perfectly accurate, and a log deviation ratio of 1 denotes that the prediction was double or half the true value etc.

1.3 Modifying the one-sided sign test

In the previous sections, we proposed an estimator of the equivalent runtime of an algorithm in a machine. Specifically, we proposed a biased estimator with an arbitrary probability of estimating a runtime longer than the true equivalent runtime. By using this estimator, we can adjust the expected percentage of samples of the performance of algorithm B computed with a runtime longer than the true equivalent runtime. When a runtime longer than the true equivalent time is estimated, the probability of making a type I error is higher than if the comparison were carried out in the same machine. Therefore, in this section, we propose a correction of the one-sided statistical test that takes into account the probability of estimating a longer than the true equivalent time and its subsequent increase in the probability type I error.

Given algorithms A, B, a one-sided hypothesis test in algorithm performance comparison is as follows[1]:

H_0: The performance of algorithm B is worse than or equal to A.
H_1: The performance of algorithm B is better than A.

We believe that the sign test [Conover, 1980] is a suitable hypothesis in the context of this chapter and, in general, for comparing the performance of optimization algorithms (see Appendix 7.1.4 for details). We limit the statistical test to the one-sided sign test, with the alternative hypothesis being that the algorithm whose equivalent runtime was estimated has a higher performance. In the following, we propose a correction for the sign test that does not increase the probability of type I error.

1.3.1 One-sided sign test

The sign test [Conover, 1980] is a special case of the binomial test, for $p = 0.5$. In the context of algorithm performance comparison, the sign test statistically assesses if the paired performance of two algorithms is the same or not. Performing this statistical test involves first executing the optimization algorithms A and B in the same machine, with the same stopping criterion, in a set of n problem instances ($i \in \{1, ..., n\}$), obtaining the scores a_i and b_i for each algorithm-instance pair.

[1] It is noteworthy that failing to reject H_0 does not imply statistical evidence that H_0 is true, instead it suggests a lack of evidence against H_0. Therefore, in this case, it would not be correct to conclude that "the algorithms compared perform the same with a statistical significance of $1 - \alpha$".

These scores depend on which random seed was chosen (this seed represents all the non-deterministic parts of the algorithms, such as solution initialization). Thus, the performance of an algorithm in an instance can also be seen as a random variable that is completely determined, given a certain seed. We denote the random variables that represent the performance of algorithms A and B in an instance i as A_i and B_i, respectively.

The statistical test allows us to draw conclusions about the algorithms on a larger set of problem instances based on the observed results in the set of n instances. The sign test is based on these three assumptions [Conover, 1980]:

- Each of the sample pairs A_i, B_i are mutually independent of the rest of the pairs.
- Any observable pair a_i, b_i can be compared, that is, we can say that $a_i < b_i$, $b_i < a_i$ or $a_i = b_i$.
- The pairs are internally consistent, or if $\mathcal{P}[A_i > B_i] > \mathcal{P}[A_i < B_i]$ for one pair, then the same is true for all pairs.

In the context of algorithm comparison, the most problematic assumption is the first one. The reason is that in real-life benchmarks, some problem instances may share similarities, which means that there is no complete independence among all sample pairs A_i, B_i. The Mann-Whitney and the Wilcoxon signed rank test also contain this first assumption [Conover, 1980]. However, in practice, this limitation is usually ignored. This is why, in general, it is a good idea to use a set of benchmark problems with many kinds of different instances. As future work, the proposed methodology could be adapted to be applicable to multiple benchmark sets, where the instances in each benchmark have similar properties.

From now on, without loss of generality[1], we assume that the algorithms deal with a minimization problem, (i.e., a_i is better than $b_i \iff a_i < b_i$). We define $\#\{A_i < B_i\}$ as a random variable that counts the number of cases[2] that $A_i < B_i$ in n instances. Then, the following hypothesis test corresponds to the one-sided sign test [Conover, 1980]:

$$\boxed{\begin{array}{l} H_0 : \mathcal{P}[A_i < B_i] \geq \mathcal{P}[A_i > B_i] \\ H_1 : \mathcal{P}[A_i < B_i] < \mathcal{P}[A_i > B_i] \end{array}}$$

[1] A maximization problem can be converted into a minimization problem by multiplying the objective function with -1.

[2] Without loss of generality, we can assume that $a_i \neq b_i$, because samples in which $a_i = b_i$ are removed when performing the sign test [Conover, 1980].

Under H_0, the null distribution of $\#\{A_i < B_i\}$ is $Bin(n, 0.5)$, where $Bin(n, 0.5)$ is the binomial distribution of size n and rate of success 0.5 [Conover, 1980]. The p-value for this hypothesis test is

$$p(k) = \mathcal{P}[\#\{A_i < B_i\} \leq k \mid H_0] = \mathcal{P}[Bin(n, 0.5) \leq k] \tag{1.2}$$

where k is substituted by the statistic of the sign test: the number of cases that $a_i < b_i$ in all $i \in \{1, ..., n\}$ samples, denoted as $\#\{a_i < b_i\}$. By definition [Wasserstein and Lazar, 2016], the p-value can be interpreted as the probability of obtaining a more extreme (lower) statistic than the observed, assuming H_0 is true. If we reject the null hypothesis when $p(\#\{a_i < b_i\}) \leq \alpha$, then the probability of type I error is less than or equal to α.

1.3.2 The corrected p-value

In practice, the statistic $\#\{a_i < b_i\}$ cannot be computed because the true equivalent runtime t_2 is unobservable. The equivalent runtime is approximated with $\hat{t}_2$ (see Definition 8). As a result, each b_i is substituted with its corresponding $\hat{b}_i$, which is computed by using $\hat{t}_2$ instead of t_2 as the stopping criterion. This means that the statistic $\#\{a_i < b_i\}$ is replaced by $\#\{a_i < \hat{b}_i\}$, which counts the number of times that $a_i < \hat{b}_i$ (without loss of generality, minimization is assumed) is observed. Therefore, we need to define the function to obtain the p-value associated to the statistic $\#\{a_i < \hat{b}_i\}$:

$$\hat{p}(k) = \mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \mid H_0] \tag{1.3}$$

where the p-value is obtained by substituting k with the observed statistic $\#\{a_i < \hat{b}_i\}$. The p-value is the probability of obtaining a statistic that is lower than or equal to the observed when H_0 is true.

Notice that if we reject the null hypothesis when $\hat{p}(\#\{a_i < \hat{b}_i\}) \leq \alpha$, then the probability of type I error is less than or equal to α. However, for this to hold, we need to assume that $\hat{b}_i < b_i$: a longer optimization time produces a lower or equal objective value (in a minimization setting). In general, we assume that a longer optimization time can only produce a lower or equal objective value.

Let p_γ be the desired upper bound of the probability of predicting a runtime longer than the true equivalent runtime for each instance i. Then, in more than $1 - p_\gamma$ of cases, b_i is obtained with a longer runtime than $\hat{b}_i$ and therefore, the probability of $\hat{b}_i \geq b_i$ is greater than $1 - p_\gamma$. When a small p_γ is chosen, we expect that $\#\{a_i < \hat{b}_i\}$ is higher than or equal to $\#\{a_i < b_i\}$, but it will not always be

so. To overcome this limitation, we need to define a corrected p-value $\hat{p}_c$, an upper bound of the actual p-value associated with statistic $\#\{a_i < \hat{b}_i\}$, that takes into account the probability $p_\gamma = \hat{t}_2 > t_2$. Specifically, we define this upper bound as

$$\hat{p}_c(k) = \sum_{v=0}^{n} (1 - \mathcal{P}[Bin(n, p_\gamma) < \max(0, v - k)])\mathcal{P}[Bin(n, 0.5) = v] \tag{1.4}$$

such that

$$\hat{p}_c(k) > \mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \mid H_0] = \hat{p}(k) \tag{1.5}$$

is satisfied (we prove this inequality in Appendix 7.1.5), where H_0 implies that statistic $\#\{A_i < B_i\}$ follows the null distribution $Bin(n, 0.5)$ [Conover, 1980]. Figure 1.6 shows p and $\hat{p}_c$ side by side. Notice that $\hat{p}_c$ is slightly higher, because it needs to account for the probability that $\hat{t}_2 > t_2$. The corrected p-value $\hat{p}_c$ is interesting because rejecting H_0 when $\hat{p}_c(\#\{a_i < \hat{b}_i\}) < \alpha$ has also an associated probability of type I error lower than α. The reason is that $\hat{p}_c(\#\{a_i < \hat{b}_i\}) > \hat{p}(\#\{a_i < \hat{b}_i\})$, and therefore, $\hat{p}_c(\#\{a_i < \hat{b}_i\}) < \alpha \Rightarrow \hat{p}(\#\{a_i < \hat{b}_i\}) < \alpha$.

To generate the corrected p-values conveniently, we created a standalone (only dependencies in the standard library) python script corrected_p_value.py available in our GitHub repository. This script uses an efficient and precise implementation of Equation (1.4). To calculate the corrected p-value, we need the probability of predicting a runtime longer than the true equivalent runtime p_γ, the sample size n, and the number of observations k in which algorithm A outperforms algorithm B. For example if $p_\gamma = 0.1$, $n = 20$, and $k = 3$, then we can get $\hat{p}_c(3)$ with

```
python corrected_p_value.py 0.1 20 3
>> 0.043596000
```

1.4 Applying the methodology

In this section, we illustrate how to apply the proposed methodology with two examples. The proposed methodology is very simple to use and does not require any additional software. Further details and material are available in our GitHub repository.

1.4.1 Example I

In this example, we will compare a simple random initialization local search procedure with a memetic search algorithm by Benlic and Hao [2015a] for the QAP.

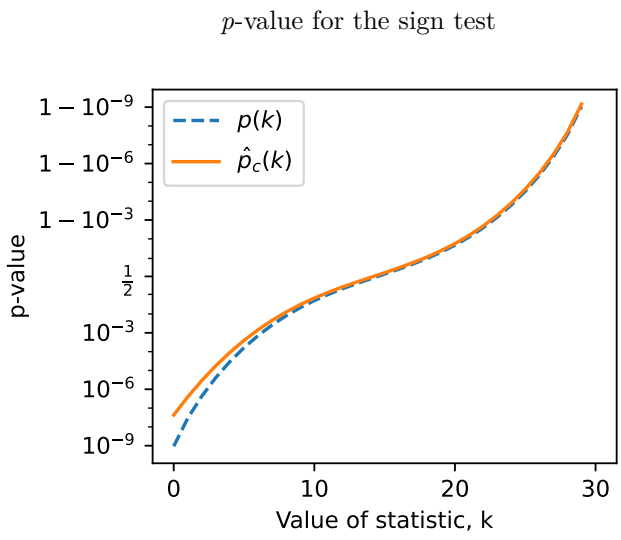


Fig. 1.6: This figure shows the p-value and the corrected p-value $\hat{p}_c$ for the sign test with a sample size of $n = 30$ and $p_\gamma = 0.01$. Under the null hypothesis H_0, the p-value represents the probability of $\#\{a_i < b_i\} \leq k$, while the corrected p-value represents an upper bound of the probability of $\#\{a_i < \hat{b}_i\} \leq k$. Under the null hypothesis, $\#\{a_i < b_i\}$ follows a binomial distribution of size n and probability of success 0.5.

Benlic and Hao [2015a] run the code sequentially, without any parallel or multi threaded execution. Using the proposed methodology, we will statistically assess the difference in the performance between these two algorithms, without having to execute the code of the memetic algorithm. In this case, the memetic search algorithm is algorithm A, because this is the algorithm of which we already have the results, and the local search algorithm is algorithm B, because this is the algorithm whose runtime is going to be estimated. For this experiment, we choose a probability of predicting a longer than true equivalent runtime of $p_\gamma = 0.01$.

Step 1: Obtaining the data

To apply the proposed methodology, we need to find certain information about the execution of the memetic algorithm. The required data includes the list of instances to be used in the experimental comparison, the average objective value obtained by the memetic search algorithm, and the runtime of the memetic search algorithm in each of the instances. The information extracted from the article by Benlic and Hao [2015a] is listed in the first three columns of Table 1.2. Also, we

need to find the CPU model of the machine in which the memetic search was run (machine M_1), which is "Intel Xeon E5440 2.83GHz" as specified in their article. Finally, the machine score of this CPU, measured as PassMark single thread score, is $s_1 = 1219$ (as seen on the file cpu_scores.md in the GitHub repo).

	Data obtained from the paper by [Benlic and Hao, 2015a]		**Data corresponding to the execution of B in machine M_2**	
instance	runtime in seconds, t_1	objective value, a_i	estimated equivalent runtime, $\hat{t}_2$	objective value, $\hat{b}_i$
tai40a	486	**3141222**	313.68	3207604
tai50a	2520	**4945266**	1626.50	5042830
tai60a	4050	**7216339**	2614.02	7393900
tai80a	3948	**13556691**	2548.18	13840668
tai100a	2646	**21137728**	1707.82	21611122
tai50b	72	**458821517**	46.47	459986202
tai60b	312	**608215054**	201.37	609946393
tai80b	1878	**818415043**	1212.13	824799510
tai100b	816	**1185996137**	526.67	1195646366
tai150b	4686	**499195981**	3024.52	505187740
sko100a	1338	**115534**	863.59	153082
sko100b	390	**152002**	251.72	155218
sko100c	720	**147862**	464.71	149076
sko100d	1254	**149584**	809.37	150568
sko100e	714	**149150**	460.84	150638
sko100f	1380	**149036**	890.70	150006

Table 1.2: This table shows all the data in the first example. The first three columns correspond to the QAP instances in which the memetic search algorithm by Benlic and Hao [2015a] was tested, the runtime of the memetic search algorithm in each instance, and the best objective value they obtained in each execution, averaged in 10 executions per instance. The information in these three columns was directly obtained from the paper by Benlic et al., without any additional executions. The next two columns show the estimated equivalent runtimes and the average objective value scores that the local search algorithm obtained with this runtime as the stopping criterion. The local search algorithm was executed in machine M_2.

Step 2: Estimating the equivalent runtime

With the data already gathered, the next step is to estimate the equivalent runtime of each instance for the machine in which the local search algorithm will be executed (machine M_2). Estimating the runtime requires the score s_2 of this

machine to be known. The CPU model of M_2 is "Intel Celeron N4100", with a PassMark single thread score of $s_2 = 1012$. With this information, we are ready to estimate the equivalent runtime $\hat{t}_2$ for each instance in machine M_2. We run the script

```
python equivalent_runtime.py 0.01 1219 1012 t_1
```

where t_1 is substituted with the runtime of the memetic search algorithm in each instance, listed in Table 1.2.

Step 3: Running the experiments

Now, we execute the local search algorithm in the instances listed in Table 1.2, using the estimated runtimes $\hat{t}_2$ as the stopping criterion. This execution is carried out in machine M_2, and the best objective function values $\hat{b}_i$ are listed in Table 1.2. Following the procedure by Benlic et al., these best objective values are averaged over 10 executions.

Step 4: Obtaining the corrected p-value

Once all the results have been computed, the next step is to compute the statistic $\#\{a_i < \hat{b}_i\}$, which counts the number of times that $a_i < \hat{b}_i$. In this case, $a_i < \hat{b}_i$ happens 15 times, and therefore, $k = \#\{a_i < \hat{b}_i\} = 15$. Now we can compute the corrected p-value of the one sided sign test. To do so, we use the script corrected_p_value.py with the chosen $p_\gamma = 0.01$, $n = 15$ and $k = 15$.

```
python corrected_p_value.py 0.1 15 15
>> 1.0000000
```

Step 5: Conclusion

Since the observed corrected p-value is not lower than the chosen $\alpha = 0.05$, we cannot reject H_0. In this case, the conclusion is that with the amount of data that we have and the chosen target probability of type I error of $\alpha = 0.05$, we can not say that the local search algorithm has a statistically significantly better performance than the memetic search algorithm[1].

It is important to note that, if we had considered the original runtimes t_1 as the stopping criterion for algorithm B in machine M_2 (longer than the estimated equivalent runtime $\hat{t}_2$), the local search would have had an unfairly longer runtime. In other words, the comparison would have been biased towards the local search.

[1] It would not be correct to conclude that the two algorithms perform (statistically significantly) the same, or that the memetic search performs statistically significantly better than the local search.

1.4.2 Example II

In this second example, we will compare the same simple random initialization local search procedure with an estimation of distribution algorithm (EDA) for the QAP [Arza et al., 2020]. The estimation of distribution algorithm (EDA) was executed sequentially, without parallel or multi threaded execution. In this case, the EDA is algorithm A, because this is the algorithm of which we already have the results, and the local search algorithm is algorithm B, because this is the algorithm whose runtime is estimated. For this experiment, we choose a probability of predicting a longer than true equivalent runtime of $p_\gamma = 0.01$.

Step 1: Obtaining the data

To apply the proposed methodology, we need to find certain information about the execution of the EDA. The required data includes the list of instances to be used in the comparison, the average objective value obtained by the EDA, and the runtime used in each instance. The information extracted from the paper by Arza et al. [2020] is listed in Table 1.3. In addition, we need to find the CPU model of the machine in which the EDA search was run (machine M_1), which is "AMD Ryzen 7 1800X", as specified in the paper. Finally, the machine score of this CPU, measured as PassMark single thread score is $s_1 = 2182$, as seen on the cpu_scores.md file.

Step 2: Estimating the equivalent runtime

With the data already gathered, the next step is to estimate the equivalent runtime of each instance for the machine in which the local search algorithm will be executed (machine M_2). The CPU model of M_2 is "Intel Celeron N4100" (the same as in the previous example), with a PassMark single thread score of $s_2 = 1012$. To estimate the runtime for each instance, we run

```
python equivalent_runtime.py 0.01 2182 1012 t_1
```

where t_1 is substituted with the runtime of the EDA algorithm in each instance, listed in Table 1.3.

Step 3: Running the experiments

Now, we execute the local search algorithm in the instances listed in Table 1.3, using the estimated runtimes $\hat{t}_2$ as the stopping criterion. This execution is carried out on machine M_2, and the best objective function values $\hat{b}_i$ are listed in Table 1.3. Following the procedure by Arza et al., these best objective values are averaged over 20 executions.

Step 4: Obtaining the corrected p-value

	Data obtained from the paper by [Arza et al., 2020]		**Data corresponding to the execution of B in machine M_2**	
instance	runtime in seconds, t_1	objective value, a_i	estimated equivalent runtime, $\hat{t}_2$	objective value, $\hat{b}_i$
bur26a	1.45	5432374	1.80	**5426670**
bur26b	1.45	3824798	1.80	**3817852**
bur26c	1.43	5427185	1.77	**5426795**
bur26d	1.44	3821474	1.78	**3821239**
nug17	0.44	1735	0.54	**1734**
nug18	0.51	1936	0.63	1936
nug20	0.68	2573	0.84	**2570**
nug21	0.77	2444	0.95	2444
tai10a	0.12	135028	0.14	135028
tai10b	0.12	1183760	0.14	1183760
tai12a	0.18	224730	0.22	**224416**
tai12b	0.19	39464925	0.23	39464925
tai15a	0.31	388910	0.38	**388214**
tai15b	0.31	51768918	0.38	**51765268**
tai20a	0.69	709409	0.85	**703482**
tai20b	0.68	122538448	0.84	**122455319**
tai40a	5.41	**3194672**	6.72	3227894
tai40b	5.41	644054927	6.72	**637470334**
tai60a	19.23	**7367162**	23.88	7461354
tai60b	19.21	**611215466**	23.86	611833935
tai80a	50.09	**13792379**	62.22	13942804
tai80b	50.1	836702973	62.23	**830729983**

Table 1.3: This table shows all the data in the second example. The first three columns correspond to the QAP instances in which the EDA algorithm by Arza et al. [2020] was tested, the runtime of the EDA in each instance, and the best objective value they obtained in each execution, averaged in 10 executions per instance. The information in these three columns was directly obtained from this paper, without any additional executions. The next two columns show the estimated equivalent runtimes and the average objective value scores that the local search algorithm obtained with this runtime as the stopping criterion. The local search algorithm was executed in machine M_2.

After the executions, the statistic $k = \#\{a_i < \hat{b}_i\}$ is computed, which counts the number of times that $a_i < \hat{b}_i$. In this case, $a_i < \hat{b}_i$ happens 4 times, and therefore, $\#\{a_i < \hat{b}_i\} = 4$. Now we compute the corrected p-value with with the chosen $p_\gamma = 0.01$, $n = 17$ and $k = 4$.

```
python corrected_p_value.py 0.01 17 4
>> 0.033192784
```

Step 5: Conclusion

The observed corrected p-value is lower than the chosen $\alpha = 0.05$, and therefore we reject H_0. The conclusion is that with a probability of type I error of $\alpha = 0.05$, the performance of the local search procedure is statistically significantly better than the performance of the EDA.

In this case, machine M_1 is more powerful (in terms of computational capabilities) than machine M_2. If we had considered the original runtimes t_1 as the stopping criterion for algorithm B in machine M_2 (shorter than the estimated equivalent runtime $\hat{t}_2$), it would have been more difficult for the local search to perform better than the EDA. In that case, H_0 might not have been rejected.

1.5 Limitations, applicability and future work

Below, we discuss the limitations, applicability, and potential future developments of the proposed methodology.

1.5.1 Multiple threads/cores

The purpose of the presented work is to compare two algorithms with the same computational resources. One of the limitations of the presented model is that it should only be applied with optimization processes that run on a single thread. This is because the single thread PassMark score and the optimization process ρ' (see Appendix 7.1.3) that were used to calibrate the linear regression in Definition 7 are also single threaded.

However, many of the optimization algorithms in the literature today are not single threaded. For example, many problems involve linear algebra operations that can benefit from a multi threaded speedup, and most consumer CPUs today have at the very least two cores. Hence, the single thread PassMark score is not suitable for optimization algorithms that involve these types of operations. Although, theoretically, it should always be possible to execute parallel code sequentially.

The multi thread PassMark score does take into account the multi threaded nature of the CPUs, and could therefore be used to re-calibrate the linear regression. This re-calibration would also involve defining another optimization process ρ' that makes use of the multi thread capabilities of the CPU, such as solving linear problems or other processes that involve matrix multiplications.

There is, however, an additional limitation inherent to parallel executing algorithms that makes their comparison in different machines difficult. Suppose we have two algorithms that run in parallel and their maximum speedup is obtained when executed in four cores, and additional threads/cores offer negligible improvement. Now let us assume that we have two machines, M_1 with four cores and M_2 with sixteen. Let us also assume that the cores in these two machines are similar in speed. Then, roughly speaking, we expect that the algorithm executed in machine M_2 should have an equivalent runtime 4 times shorter. However, since the two algorithms can only take advantage of the speedup provided by, at most, four cores; the algorithm executed in machine M_2 would have a huge disadvantage.

In general, it is difficult to know the maximum potential speedup of the execution in parallel of an algorithm presented in the literature. This makes the prediction of the equivalent runtime of parallel algorithms more challenging than for single thread algorithms. Taking into account the additional difficulty associated to the comparison of parallel algorithms, we think that the comparison of optimization algorithms that run on single thread is a reasonable starting point. In future work, it would be interesting to extend the proposed methodology for multi thread algorithms.

1.5.2 CPU as the only bottleneck

The PassMark single thread score measures the computing capability of the CPU, disregarding other components like the hard disk or the speed and the size of the RAM memory. Therefore, the prediction of the equivalent runtime is only applicable to optimization algorithms that are CPU intensive, or in other words, optimization algorithms that have their execution speed limited by the CPU.

There are many optimization algorithms whose runtime is determined mainly by the speed of the memory, instead of the CPU. Specially, when optimization algorithms require large amounts of data to be loaded to memory repeatedly. Conversely, when an optimization algorithm does not use too much memory, we can expect its runtime to be more CPU dependent.

In addition, many optimization algorithms in machine learning today are executed in GPUs and sometimes even on specialized hardware. Compared to CPU execution, GPU offers speedups when similar operations are applied to large amounts of data. As in the multi core case, predicting the runtime of algorithms in GPUs

is more challenging than in single threaded execution in CPU. GPUs themselves have several processing cores and integrated memory that varies in size and speed from model to model.

The proposed methodology could be adapted for algorithms whose runtime depends on memory or GPU speed. In fact, PassMark has a benchmarks for RAM and GPUs. Therefore, it could be possible to re-calibrate the linear regression for either RAM or GPU dependent tasks. This re-calibration would also involve defining another optimization process ρ'. Taking into account the additional difficulty associated to measuring the runtime of algorithms that depend on RAM and GPU, we think that the comparison of optimization algorithms whose runtime depends primarily on the CPU is a reasonable starting point for this chapter. In future work, it could be interesting to adapt the methodology for algorithms whose runtime depends on RAM or GPU.

1.5.3 Efficiency of the implementation

In addition to the limitations related to the hardware, the implementation of the algorithms can also have an impact in the runtime. For instance, if the same algorithm is implemented in both Python and C, the runtime in C is likely to be shorter. But even within the same programming language, the runtime could change depending on the compiler flags (i.e. the -O3 will probably outperform no optimizations) or the configuration of the interpreter. In addition to the previous factors, the implementation itself could also be more or less efficient, depending on the skill of the programmer and the time it invests in designing efficient code.

Even though there are quite a few factors that depend on the implementation, we argue that by implementing the code in the same programming language the results should be comparable. In any case, even when the methodology proposed in this chapter is not used, it is the responsibility of the researcher to make sure that the comparison is fair in terms of the implementation. This limitation is not inherent to the proposed methodology, but to the comparison of two algorithms in general.

1.5.4 The variance in the PassMark single thread score

The PassMark single thread score is a score for CPUs that is correlated with their single thread performance. However, the performance of CPUs is not the same even within the same model. This is known as the *silicon lottery*, and is caused by the manufacturing process of CPUs. In addition, the performance of the CPU will also be limited by the cooling system used. With better cooling, we can expect a better CPU performance.

The PassMark single thread score takes into account this variance, and the scores are the averages of several users' submitted results. Still, the cooling setup and the silicon lottery of the researcher that wants to apply the proposed methodology will inevitably introduce a variance to the predictions of the equivalent runtime.

The presented method models the probability of predicting an equivalent runtime that is longer than the true equivalent runtime. And by doing so, it takes into account this variance because the machines used in the calibration process of the linear regression inevitably have different cooling systems and are also affected by the silicon lottery.

1.5.5 Very high and low PassMark scores

Finally, there is a limitation regarding the chosen machine score: the PassMark single thread score. In Section 1.2, we saw that a linear function is a suitable function to model the relationship between the machine score and the runtime of the reference optimization process ρ' (the definition of ρ' is given in Appendix 7.1.3). The reference optimization process ρ' is used to calibrate the linear regression in Definition 7 so that the equivalent runtime of other optimization processes ρ can later be predicted based on this formula. The formula of the fitted linear regression is $t(M_j, \rho') \approx -0.62280 s_j + 2008$ where $t(M_j, \rho')$ is the equivalent runtime of ρ' in machine M_j, and s_j is the score of machine M_j. With this formula, a PassMark single thread score higher than 3223 produces a negative estimated equivalent runtime, which does not make sense. However, for the 8 machines used to fit the data, as Figure 1.2 shows, the linear model seems to be suitable.

To overcome this limitation, we recommend applying the proposed methodology only in machines with PassMark single thread scores in the interval $(411, 2185)$. These values correspond to the highest and lowest values used in the fitting of the linear regression. More than 70% of the CPUs in the provided list (see the file cpu_scores.md in the GitHub repo) have their PassMark score in this interval. In addition, the CPUs that do not have their score in this interval are either very new orvery old, which means that the proportion of the user base with the PassMark single thread score in this interval is probably way higher than 70%. As future work, and especially when more powerful processors are available, the methodology can be updated to incorporate these new processors or even change the machine score to other benchmark scores beyond the PassMark single thread score.

1.5.6 Assumptions of the corrected sign test

The corrected sign test is based on certain assumptions and should only be used taking into account certain limitations that will be addressed in this section. First, we will address the assumption related to the probability of predicting a runtime longer than the true equivalent runtime. Let $(\hat{t}_2 > t_2)_i$ be a random variable that represents if the estimated equivalent runtime for algorithm A, instance i in machine M_2, is longer than the true equivalent runtime or not. The required assumption is similar to assuming that $\hat{t}_2 > t_2$ is mutually independent for each instance i. Specifically, it is required[1] that $\mathcal{P}[(\hat{t}_2 > t_2)_i | \cap_{i \neq j} (\hat{t}_2 > t_2)_j] < p_\gamma$.

One could argue that this assumption is false because $(\hat{t}_2 > t_2)$ depends on many factors, such as the machines used in the experimentation. The same two machines are used to compute all samples a_i, b_i suggesting that all $(\hat{t}_2 > t_2)_i$ can never be truly independent among each other. However, even though we can not ensure that $\mathcal{P}[(\hat{t}_2 > t_2)_i | \cap_{i \neq j} (\hat{t}_2 > t_2)_j] < p_\gamma$, by choosing a suitable correction coefficient γ, in Section 1.2, we estimated that $\mathcal{P}[(\hat{t}_2 > t_2)_i] < p_\gamma$.

In addition to the previous assumption, the proposed methodology only considers one side hypothesis testing. In this regard, it should only be applied to show a statistically significantly superior performance of the algorithm whose equivalent runtime was estimated (denoted as algorithm B in this chapter). The reason is that algorithm B has a high probability of having a lower runtime, thus, it is easy that B performs worse than A, while the opposite is difficult. Failing to reject H_0 only indicates a lack of evidence against H_0, and in our context, it only indicates that there is not enough evidence to say that B performs better than A (it tells us nothing about A performing better than B).

1.6 Conclusion

Usually, a fair comparison of optimization algorithms with a maximum runtime as a common stopping criterion requires the algorithms to be executed in the same machine. Unfortunately, it is not always possible to do this. An alternative is to adjust the runtime of the algorithms relative to the speed of the machine in which they are executed. In this chapter, we proposed a methodology to statistically compare the performance of two optimization algorithms in two different machines, when the results of one of the algorithms are already known and without having to execute this algorithm again. The methodology ensures that the probability of type I error does not increase due to the algorithms being executed in different

[1] This assumption is required in the proof of Equation (1.5) in Appendix 7.1.5. Specifically, it is used in Lemma 3.

machines. To achieve this, first, the runtime of the executed algorithm is adjusted based on the speed of the CPUs of both machines. Then, a modified one-sided sign test is used so that the probability of using an unfairly longer runtime is taken into account. We illustrate the application of the proposed methodology with two examples.

Alongside this chapter, a tutorial with examples is presented in our GitHub repository. In addition, we offer two standalone scripts (also in the same repository): one to estimate the equivalent runtime and another one to generate the corrected p-values for the one sided statistical test. This will hopefully make it simple for people to apply the proposed methodology.

Supplementary Material

Code to Reproduce the Results

The code to reproduce the results in this chapter are available in the GitHub repository https://github.com/EtorArza/RTDHW.

Scripts to Apply the Methodology

The standalone scripts equivalent_runtime.py and corrected_p_value.py required to apply the methodology are also available in the GitHub repository.

2

Comparing Two Optimization Algorithms Through Stochastic Dominance

The performance of an stochastic optimization algorithm (the best objective value observed) will vary in each execution, and thus can be modeled as a random variable. Therefore, comparing the performance of two stochastic optimization algorithms involves repeated executions, and the comparison can be carried out via the averages or more sophisticated tools such as statistical null hypothesis statistical tests. Existing methods considered for the comparison of two samples of the performance of two algorithms have certain limitations.

For instance, in statistical null hypothesis tests, the p-value does not separate between the effect size and the sample size [Benavoli et al., 2017, Calvo et al., 2019a]. In short, statistical null hypothesis tests allow us to reach a *yes-no* type conclusion, although a lot of information is lost. Bayesian analysis [Benavoli et al., 2017, Calvo et al., 2019a] overcomes this limitation, but still disregard the "different distributions same summary statistics" [Matejka and Fitzmaurice, 2017, F. J. Anscombe, 1973] problem: two very different distributions can still have the same Bayesian analysis result.

The stochastic dominance is a property for random variables. If the performance of an optimization algorithm A stochastically dominates another algorithm's performance B, then this means that for all objective value $v \in \mathbb{R}$, the probability that A produces a value better than v is higher than for B. If this property holds, then it is clear that algorithm A is better. Unfortunately, it is not always the case that one of the algorithms stochastically dominates the other one. In such case, it is necessary to consider other alternatives.

In this chapter, we propose framework to compare two samples of the performances of two algorithms through the first order stochastic dominance. First, we introduce a dominance measure for two random variables that quantifies the proportion in which the cumulative distribution function of one of the random variables stochas-

tically dominates the other one. In addition, based on this measure, we present a graphical method that i) is directly estimated from observed samples, ii) includes a confidence band that estimates the uncertainty, iii) visually shows the introduced dominance measure and iv) can differentiate the effect size and the sample size.

With illustrative purposes, we re-evaluate the experimentation of an already published work with the proposed methodology and we show that additional conclusions—missed by the rest of the methods—can be inferred. Additionally, the software package *RVCompare* was created as a convenient way of applying and experimenting with the proposed framework.

2.1 Introduction

The objective value returned by an optimization algorithm may be non-deterministic. For example, in stochastic algorithms, the objective value returned depends on the seed used in the random number generator. In these kinds of scenarios, we can think of them as the observations of random variables with unknown distributions. Based on these measurements, we sometimes need to choose the random variable that takes the lowest (or largest) values[1]. The expected values of the random variables—usually estimated as an average of several repeated observations—can be used for this purpose. However, many statisticians have claimed that summarizing data with simple statistics such as the average or the standard deviation is misleading, as very different data can still have the same statistics [Matejka and Fitzmaurice, 2017, Chatterjee and Firat, 2007].

Motivating example 1)

Let us first consider a real-world motivation. Suppose we need to choose the best option between two stochastic gradient-based methods for optimizing the parameters of a neural network. A neural network classifier trained with a gradient-based method will produce different error rates [Goodfellow et al., 2016] each time it is trained-tested, even if the same train-test dataset is used in each repeated measurement. One of the reasons is that the learned classifier depends on the initialization of its weights (before applying a gradient-based optimizer), which are often initialized randomly [Glorot and Bengio, 2010].

To illustrate the previous scenario, we trained and tested a neural network[2] in the MNIST dataset, and we compared the performance (the error rate on the test set) of gradient-based optimizers in this data set: *adam* and *RMSProp* [Goodfellow et al., 2016]. The error rate in the test set depends on the seed used to train the neural network, and therefore, we can model the error rate of each of the algorithms in this problem as a random variable. Figure 2.1 shows the kernel density estimations of these random variables using the uniform kernel. As we see in the figure, the error rate is not the same in each measurement and ranges between 0.022 and 0.04. This shows that, in this context, it makes sense to model the error rate as a random variable rather than a constant: a unique value cannot represent the error rate without a significant amount of information loss.

[1] Without loss of generality, minimization is assumed in this chapter.

[2] We follow an example in the Keras [Chollet et al., 2015] library, and train the neural network for one epoch.

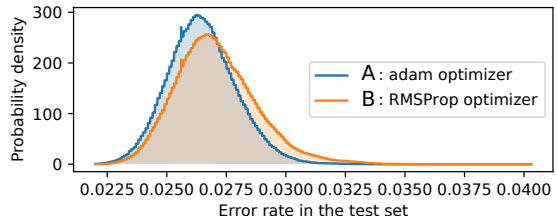


Fig. 2.1: Density estimates of the error rates produced by the optimizers *adam* and *RMSProp* in the MNIST dataset. The sci-kit learn[Pedregosa et al., 2011] package was used in the estimation.

Motivating example 2)

In the following, we present another example with synthetic data. Let us consider the two random variables A and B shown in Figure 2.2. B has a lower expected value than A, $\mathbb{E}[B] < \mathbb{E}[A]$. If we use the expected value as the only criterion, then B takes lower values than A. However, notice that with a low but nonzero probability, B will take very large values that are undesirable in the context of minimization. Without loss of generality, in this chapter, we assume that lower values are preferred.

An error with low variance is very important in an environment where reliability is key, even if it means a slightly worse expected value. Some examples include breast cancer detection [Cruz-Roa et al., 2017], or some reinforcement learning tasks [François-Lavet et al., 2018, Mnih et al., 2013] like self-driving cars [Badue et al., 2021].

In other circumstances, obtaining the lowest possible error can be more important than reliability. One could argue that reliability is less important in sentiment analysis [Zhang et al., 2018], or in certain real-world optimization problems [Regnier-Coudert et al., 2016], where obtaining the best possible solution is key. When obtaining the best possible score is more important than reliability, it may even be worth running an optimization algorithm several times and choosing the best solution out of all the runs. In that case, A would also be preferred to B, as A has a higher probability of taking a value lower than 0.01 (see Figure 2.2).

A. Related work

In these two examples, we have seen that summarizing and comparing random variables with only the expected value can leave important information out (such as which of the random variables can take lower values), especially when neither

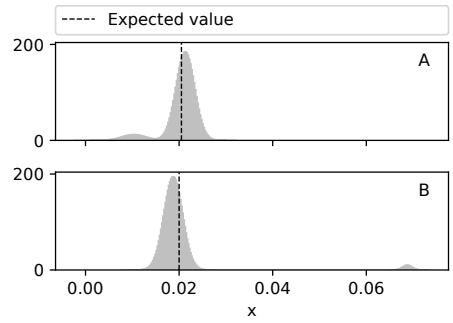


Fig. 2.2: The probability density of two random variables A and B, with probability density functions $g_A = 0.925 \cdot g_{\mathcal{N}(0.210325,0.002)} + 0.075 \cdot g_{\mathcal{N}(0.010325,0.025)}$ and $g_B = 0.975 \cdot g_{\mathcal{N}(0.01875,0.002)} + 0.025 \cdot g_{\mathcal{N}(0.06875,0.001)}$ where $\mathcal{N}(\rho, \sigma)$ is the normal distribution with mean ρ and standard deviation σ. Their expected values are $\mathbb{E}[A] = 0.0205$ and $\mathbb{E}[B] = 0.02$ respectively.

random variable clearly takes lower values than the other one. Many works in the literature use statistical null hypothesis tests [Conover, 1980, Mann and Whitney, 1947, Wilcoxon, 1945] to analyze observed samples and choose one of the random variables accordingly. Nonetheless, as claimed by Benavoli et al. [2017], statistical null hypothesis tests have their limitations too: when the null hypothesis is not rejected—this will happen often when the random variables being compared take similar values—, we get no information. Not only that but even when the null hypothesis is rejected, it does not quantify the amount of evidence in favor of the alternative hypothesis [Benavoli et al., 2017].

B. Contribution

In this chapter, we propose a graphical framework that compares two random variables using their associated cumulative distribution functions, in the context of choosing the one that takes lower values. Specifically, we first propose 8 desirable properties for *dominance measures*: functions that compare two random variables in this context of stochastic dominance. From the measures in the literature, we find that the *probability that one of the random variables takes a lower value than the other random variable* satisfies most of these properties. In addition, we propose a new dominance measure, the *dominance rate*, that also satisfies most of the properties and is related to the first-order stochastic dominance [Quirk and Saposnik, 1962]. Then, we propose a graphical method that involves visually comparing the random variables through these two dominance measures. The graphical

method, named *cumulative difference-plot*, can also be used to compare the quantiles of the random variables, and it models the uncertainty associated with the estimate. By re-evaluating the experimentation of a recently published paper with the proposed methodology, we demonstrate that this new plot can be useful to compare the performance of stochastic optimization algorithms, especially in the case when the random variables take similar values.

Finally, an *R* package named *RVCompare*, available in CRAN, is distributed alongside this chapter. With this package, the *cumulative difference-plot* can be conveniently computed. The source code of the package and the supplementary material for the chapter are available at github.com/EtorArza.

The rest of the chapter is organized as follows: in the next section, we propose eight desirable properties for dominance measures. Then, in Section 2.3, we study two dominance measures that satisfy most of these properties. Section 2.4 introduces a graphical method to compare random variables. In Section 2.5, we discuss related methods in the literature and compare them to the proposed approach. Section 2.6, evaluates the proposed graphical method and other alternatives in an already published experimentation. In Section 2.7, we state the assumptions and limitations of the proposed *cumulative difference-plot*. Finally, Section 2.8 concludes the chapter.

2.2 Desirable properties for dominance measures

2.2.1 Background

When we have two random variables and we need to choose the one that takes the lowest values, we usually take i) the random variable with the lowest expected value or ii) the random variable with the lowest median. The *median* [Conover, 1980] of a continuous random variable A, denoted as m_A, is the value that satisfies $\mathcal{P}(A < m_A) = \mathcal{P}(A > m_A)$. In other words, if m_A is the median of A, a sample of A is as likely to be lower than m_A as it is to be higher.

Interestingly enough, the median and the expected value have their strengths and weaknesses when it comes to choosing the random variable that takes the lowest values. In the following, we elaborate on this point with two particular cases of study. The first case is shown in Figure 2.3, with two random variables A and B. Each of the random variables is a mixture of two Gaussian distributions with the same shape and similar weight in the mixture. It is clear that A tends to take values lower than B, as the Gaussian distributions of A are centered in 0.05 and 0.07, while the Gaussian distributions of B are centered in 0.06 and 0.08. While the expected values of A and B are aligned with this intuition, the medians are

not; as $\mathbb{E}[A] < \mathbb{E}[B]$ and $m_A > m_B$. However, the expected value does a poor job of summarizing the bimodal shape of A or B: both of these random variables usually take much higher or much lower values than their expected values.

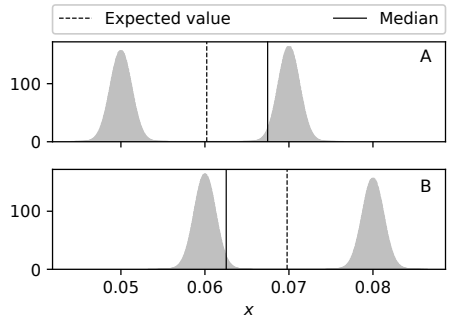


Fig. 2.3: Case 1. The probability density functions of A and B: $g_A = 0.489 \cdot g_{\mathcal{N}(0.05,0.00125)} + 0.511 \cdot g_{\mathcal{N}(0.07,0.00125)}$ and $g_B = 0.511 \cdot g_{\mathcal{N}(0.06,0.00125)} + 0.489 \cdot g_{\mathcal{N}(0.08,0.00125)}$ where $g_{\mathcal{N}(\rho,\sigma)}$ is the density function of the normal distribution with mean ρ and standard deviation σ.

The second case is shown in Figure 2.4. With a very high probability, A takes lower values than B, even though B will rarely take really low values, which might prove useful in some particular applications. In this case, $m_A < m_B$ and $\mathbb{E}[A] > \mathbb{E}[B]$, hence, the comparison of the medians are aligned with the intuition that A takes lower values than B, while the expected values are not. In the presence of outliers [Carreño et al., 2020], the median is considered more robust than the expected value [Rousseeuw and Hubert, 2011].

Notice that, in the second case, it is not trivial to choose between A and B, as B can take lower values, but A is more likely to be lower than B. So, when can we claim that one of them clearly takes lower values than the other? When the cumulative distribution of A is higher than the cumulative distribution of B in the entire domain of definition: in that case, A has a higher probability than B of taking values lower than x, for all x in the domain of definition. This is known [Mann and Whitney, 1947] as A being stochastically smaller than B. Depending on the field of study, it can also be referred to as "A stochastically dominates B" [Quirk and Saposnik, 1962, Schmid and Trede, 1996, Bennet, 2013]. The stochastic dominance can be further relaxed, obtaining what is known as *first-order stochastic dominance* in the literature [Quirk and Saposnik, 1962, Schmid and Trede, 1996], although, for the sake of brevity, we will call it *stochastic dominance* throughout the chapter.

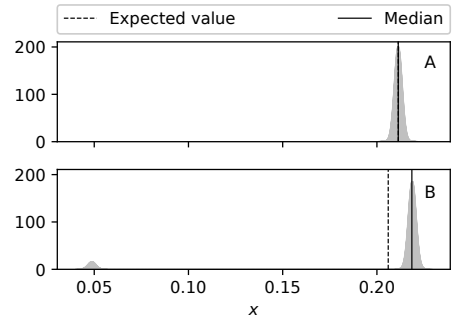


Fig. 2.4: Case 2. The probability density functions of A and B: $g_A = g_{\mathcal{N}(0.211325,0.002)}$ and $g_B = 0.925 \cdot g_{\mathcal{N}(0.21875,0.002)} + 0.075 \cdot g_{\mathcal{N}(0.04875,0.002)}$ respectively.

Definition 9 *(Stochastic dominance) Let A and B be two continuous random variables defined in a connected subset $N \subseteq \mathbb{R}$. We say that A stochastically dominates B, denoted as $A \succ B$, when*

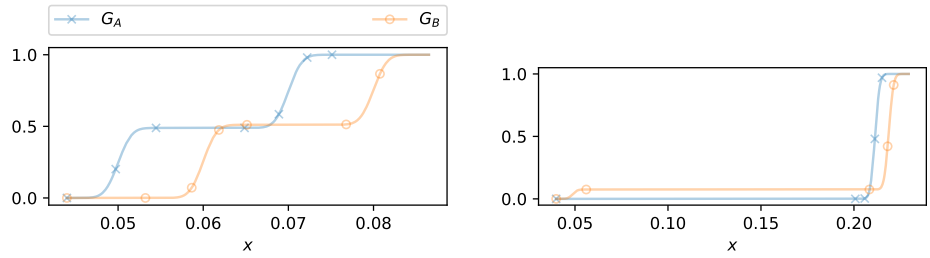
i) $G_A(x) \geq G_B(x)$ *for all* $x \in N$
and
ii) There exists an $x \in N$ *such that* $G_A(x) > G_B(x)$.

where G_A and G_B are the cumulative distributions of A and B respectively.

For A not to stochastically dominate B (denoted as $A \not\succ B$), either condition i) or ii) must be violated. Note that $A \not\succ B$ is not equivalent to $B \succ A$. The special case that $A \not\succ B$ and $B \not\succ A$ at the same time is defined, it is said that A and B *cross* [Bennet, 2013], and we denote it as $A \lesseqgtr B$. In the non trivial ($A \neq B$) case that $A \lesseqgtr B$, there exists two points $\sigma_1, \sigma_2 \in N$ such that $G_A(x_1) < G_B(x_1)$ and $G_A(x_2) > G_B(x_2)$: we cannot say, for all $x \in N$, that one of the random variables has a higher probability of taking values lower than x.

Let us now see how the cumulative distributions can be used to compare random variables in an example. In Figure 2.5a, the cumulative distributions of the random variables described in Figure 2.3 are shown. We can see that $G_A(x) > G_B(x)$ for almost all $x \in N$. But there is at least a point $x \in (0.06, 0.07)$ where $G_A(x) < G_B(x)$, hence, $A \lesseqgtr B$. The same happens in Case 2 shown in Figure 2.5b. As in the previous case, $A \lesseqgtr B$, because even though $G_A(x) > G_B(x)$ for almost all $x \in N$ (in which $g_A(x) \neq 0$ and $g_B(x) \neq 0$), for all $x \in (0.05, 0.2), G_A(x) < G_B(x)$.

In the following, we will study how to quantify the difference between two random variables, emphasizing the degree to which one of the random variables stochastically dominates the other.



(a) Case 1 (b) Case 2

Fig. 2.5: The cumulative distributions of the two cases shown in Figures 2.3 and 2.4.

2.2.2 Desirable properties

There are many ways to compare two random variables, each with a different point of view: some aim to find how dissimilar two random variables are (disregarding which of them takes lower values), while other methods try to guess if one of the random variables stochastically dominates the other one. In the context of this chapter, we are interested in measures that, given two random variables, quantify through the stochastic dominance how much one of the random variables tends to take lower values than the other. We use the term *dominance measure* to refer to functions that quantify the difference between two random variables following this intuition. In this section, we define eight desirable properties for these dominance measures, and we study the suitability of several measures from the literature.

Definition 10 *Let A and B be two continuous random variables. We define a* dominance measure *between two random variables as a function $\mathcal{C}$ that maps two random variables into a real value $\mathcal{C}(A, B)$.*

It is desirable that $\mathcal{C}(A, B)$ quantifies the stochastic dominance. Hence, we want $\mathcal{C}(A, B)$ to be proportional to the portion of the support of A and B in which $G_A(x) < G_B(x)$. Formally, this intuitive idea can be represented as:

Property 1 *$\mathcal{C}$ is defined in the $[0, 1]$ interval, where:*

i)

$$\mathcal{C}(A, B) = 1 \iff A \succ B$$

ii)

$$\mathcal{C}(A, B) = 0 \iff B \succ A$$

iii)

$$\mathcal{C}(A, B) \in (0, 1) \iff B \lesseqgtr A$$

Proposition 1 *If a dominance measure $\mathcal{C}$ satisfies Property 1 i) and ii), then it also satisfies Property 1 iii).*

Proof. By definition, $B \lesseqgtr A$ iff $A \not\succ B$ and $B \not\succ A$. Property 1 i) and ii) implies that $A \not\succ B$ and $B \not\succ A$ iff $\mathcal{C}(A, B) \neq 1$ and $\mathcal{C}(A, B) \neq 0$. From Property 1 i) also $\mathcal{C}(A, B) \in [0, 1]$, thus $B \lesseqgtr A$ iff $\mathcal{C}(A, B) \in (0, 1)$.

Property 2 *(Antisymmetry) $\mathcal{C}(A, B)$ and $\mathcal{C}(B, A)$ add up to 1.*

$$\mathcal{C}(A, B) = 1 - \mathcal{C}(B, A)$$

It is noteworthy that Property 1 ii) can be inferred from Property 1 i) and Property 2.

Property 3 *The inversion (under the sum) of the operands of $\mathcal{C}$ equals the inversion of $\mathcal{C}$:*

$$\mathcal{C}(-1 \cdot A, -1 \cdot B) = 1 - \mathcal{C}(A, B)$$

Property 4 *When A and B are equal, $\mathcal{C}$ is symmetric.*

$$A = B \implies \mathcal{C}(A, B) = \mathcal{C}(B, A)$$

Assuming Property 2 holds, we can rewrite the previous property as:

$$A = B \implies \mathcal{C}(A, B) = 0.5.$$

Note that the opposite is not true:

$$\mathcal{C}(A, B) = \mathcal{C}(B, A) \not\Rightarrow A = B$$

Property 5 *(Invariance to translation) Moving the domain of definition of A and B by the same amount does not change $\mathcal{C}$*[1]*.*

$$\text{for all } \lambda \in \mathbb{R}, \quad \mathcal{C}(A + \lambda, B + \lambda) = \mathcal{C}(A, B)$$

Property 6 *(Invariance to scaling) Scaling both A and B by the same positive amount does not change $\mathcal{C}$.*

$$\text{for all } \lambda > 0, \quad \mathcal{C}(\lambda \cdot A, \lambda \cdot B) = \mathcal{C}(A, B)$$

[1] We define $A + \lambda$ as the random variable that is sampled in two steps: first obtain an observation from A and then add λ to this observation. We define $\lambda \cdot A$ in a similar way.

In the following lines, we give an intuition for Property 7. In Case 2, shown in Figure 2.4, we saw that for all $x \in (0.075, 0.2)$, $G_A(x) < G_B(x)$. However, notice that most of the mass of A and B is in the interval $(0.2, 0.23)$, where $G_A(x) > G_B(x)$. This means that most of the observed points of A and B will be in that interval. Therefore, it makes sense that $G_A(x) > G_B(x)$ has a higher weight than $G_A(x) < G_B(x)$ in the computation $\mathcal{C}(A, B)$. In other words, the *small* mass of B centered in 0.05 can only account for a *small* part of $\mathcal{C}(A, B)$. In what follows, this is formalized as B being a mixture of two distributions, where one of the distributions represents this small mass with a small weight in the mixture. Property 7 states that the change in the computation of $\mathcal{C}$ produced by the distribution of small weight in the mixture can be, at most, its weight in the mixture.

Property 7 *Let* $B = \mathcal{M}_{[1-\tau,\tau]}(B1, B2)$ *be the mixture*[1] *distribution of* $B1$ *and* $B2$ *with weights* $1 - \tau$ *and* τ *respectively and let* A *be another random variable. Then,*

$$|\mathcal{C}(A, B) - \mathcal{C}(A, B1)| \leq \tau$$

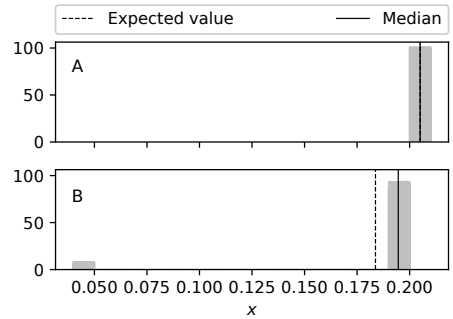


Fig. 2.6: Case 3. The probability density functions of A and B: $g_A = g_{\mathcal{U}(0.2,0.21)}$ and $g_B = 0.925 \cdot g_{\mathcal{U}(0.19,0.2)} + 0.075 \cdot g_{\mathcal{U}(0.04,0.05)}$ respectively, where $\mathcal{U}(0.2, 0.21)$ is the uniform distribution in the interval $(0.2, 0.21)$.

Property 8 explains that, under certain circumstances, $\mathcal{C}(A, B)$ is invariant to the translation/dilatation of one of the random variables. Specifically, it states that the distribution of one of the random variables (B) can change without affecting

[1] The probability density function of $\mathcal{M}_{[1-\tau,\tau]}(B1, B2)$ is defined as $(1 - \tau) \cdot g_{B1}(x) + \tau \cdot g_{B2}(x)$. Note that $\tau \in [0, 1]$.

the value of $\mathcal{C}(A, B)$ as long as the changed part does not overlap with the support of the other random variable (A). Let us assume that the random variable B is defined as mixture distribution $\mathcal{M}_{[1-\rho,\rho]}(B1, B2)$ where the supports of $B2$ and A do not overlap, with $\rho \in (0, 1)$. Property 8 states that a translation and/or dilatation can be applied to $B1$, as long as: i) this transformation does not cause an overlap of the supports of A and $B2$, and ii) partial transformations will also not cause an overlap (hence the need for ξ_1 and ξ_2 in Property 8). In the following, we formalize this property:

Property 8 *Let $B = \mathcal{M}_{[1-\rho,\rho]}(B1, B2)$ be the mixture distribution of $B1$ and $B2$ with weights $1-\rho$, and ρ, respectively and let A be another random variable with $\rho \in (0, 1)$. Suppose that $\text{supp}(B2) \cap \text{supp}(A) = \varnothing$. Let $\lambda_1 \in \mathbb{R}^+, \lambda_2 \in \mathbb{R}$ be two numbers such that for all $\xi_1, \xi_2 \in [0, 1]$, $\text{supp}((1 + (\lambda_1 - 1)\xi_1) \cdot B2 + \xi_2\lambda_2) \cap \text{supp}(A) = \varnothing$. Then,*

$$\mathcal{C}(A, B) = \mathcal{C}(A, \mathcal{M}_{[1-\rho,\rho]}(B1, \lambda_1 \cdot B2 + \lambda_2)$$

This property can be applied to the distributions in Case 3 shown in Figure 2.6. For example, the probability mass in the interval $(0.04, 0.05)$ could have been centered in 0.1 or 0.15 instead of 0.045, without any changes to $\mathcal{C}(A, B)$. In addition to the position, the shape of the mass can also be altered as long as its weight in the mixture stays the same and does not overlap with A.

Unfortunately, it is impossible that a dominance measure satisfies Properties 1 and 7 at the same time. Intuitively, the problem is that, given the distributions A and $B = \mathcal{M}_{[1-\tau,\tau]}(B1, B2)$, it is possible that $A \succ B1$ and at the same time $B \succ A$ with $\tau < 0.5$[1]. We formalize and prove this claim in the following proposition:

Proposition 2 *Let $\mathcal{C}$ be a dominance measure.*

i) If $\mathcal{C}$ satisfies Property 1, then it fails to satisfy Property 7.

ii) If $\mathcal{C}$ satisfies Property 7, then it fails to satisfy Property 1.

Proof. A dominance measure only satisfies a property when that property is true for every possible random variable. Consequently, to prove this proposition, it is enough to find four random variables $A, B, B1$ and $B2$ where

i) $B = \mathcal{M}_{[0.1,0.9]}(B1, B2)$,
ii) $A \succ B1$,
iii) $B \succ A$.

If four random variables can be found that satisfy these three statements, then with Property 1 we obtain that $\mathcal{C}(A, B1) = 1$ and $\mathcal{C}(A, B) = 0$. This contradicts

[1] See https://etorarza.github.io/pages/2021-interactive-comparing-RV.html for an interactive example that illustrates the above point.

Property 7, because $|\mathcal{C}(A,B) - \mathcal{C}(A,B1)| \not\leq 0.1$. The same is true the other way around, Property 7 states that $|\mathcal{C}(A,B) - \mathcal{C}(A,B1)| \leq 0.1$ and this contradicts Property 1, with $\mathcal{C}(A,B1) < 1$ or $\mathcal{C}(A,B) > 0$.

A simple example in which this happens is for the random variables

$A = \mathcal{U}(0,1)$,

$B = \mathcal{M}_{[0.9,0.1]}(\mathcal{U}(0.1,1), \mathcal{U}(-0.5,0))$,

$B1 = \mathcal{U}(0.1,1)$,

$B2 = \mathcal{U}(-0.5,0)$.

The cumulative distribution functions of A, B and $B1$ are shown in Figure 2.7, where it is clear that $B \succ A$ and $A \succ B1$.

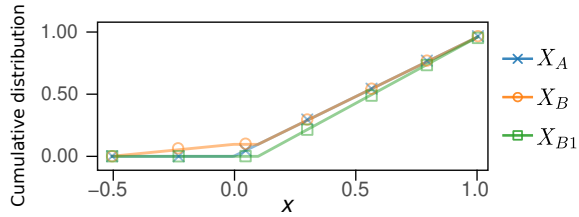


Fig. 2.7: The cumulative distribution functions of A, B and $B1$.

In the following, we will briefly review several measures in the literature and, specifically, which of the proposed properties they satisfy. Many measures describe the difference between A and B, disregarding whether the difference in cumulative density is positive or negative. Consequently, they cannot satisfy Property 1 (see Appendix 7.2.1 for details). This is the case for *f-divergences*—including Kullback-Leibler, Jensen-Shannon, the Hellinger distance and the total variation—and for the Wasserstein distance. These measures also fail to satisfy several other properties (see a summary in Table 2.1).

2.3 Dominance measures

Most of the measures in the literature fail to satisfy the eight properties introduced in Section 2.2.2. However, there is a dominance measure in the literature that overcomes this limitation: the probability that $A < B$ [Conover, 1980].

Table 2.1: Which of the properties in Section 2.2.2 does each measure satisfy?

	1	2	3	4	5	6	7	8
Kullback-Leibler divergence				✓	✓	✓		✓
Jensen-Shannon divergence				✓	✓	✓		✓
Total-Variation				✓	✓	✓	✓	✓
Hellinger distance				✓	✓	✓	✓	✓
Wasserstein distance				✓	✓			
$\mathcal{C}_{\mathcal{P}}$: Probabitlity of $A < B$		✓	✓	✓	✓	✓	✓	✓
$\mathcal{C}_{\mathcal{D}}$: Dominance rate of A over B	✓	✓	✓	✓	✓	✓		✓

A checkmark ✓ indicates that the measure satisfies the property.

2.3.1 $\mathcal{C}_{\mathcal{P}}$: the probability of $A < B$

We can compare A and B with the probability that a value sampled from A is smaller than a value sampled from B. When the random variables are exactly the same, this probability is 0.5. Formally, given two continuous random variables A and B defined in a connected set $N \subseteq \mathbb{R}$, the probability that $A < B$ is defined as:

$$\mathcal{C}_{\mathcal{P}} = \mathcal{P}(A < B) = \int_N g_B(x) G_A(x) dx. \tag{2.1}$$

One of the advantages of $\mathcal{C}_{\mathcal{P}}$ is its easy interpretation. In addition, $\mathcal{C}_{\mathcal{P}}$ is a well behaved dominance measure, as it satisfies Properties 2, 3, 4, 5, 6, 7 and 8. It also satisfies a weak version of Property 1:

$$\mathcal{C}_{\mathcal{P}}(A, B) = 1 \implies A \succ B \implies \mathcal{C}_{\mathcal{P}}(A, B) \in (0.5, 1]$$

and

$$\mathcal{C}_{\mathcal{P}}(A, B) = 0 \implies B \succ A \implies \mathcal{C}_{\mathcal{P}}(A, B) \in [0, 0.5).$$

Note that, when $A \succ B$, $\mathcal{C}_{\mathcal{P}}(A, B) \neq 1$ is still possible, and this is why it does not satisfy Property 1 entirely. For instance, when the probability densities of A and B are two Gaussian distributions with the same variance and the mean of A is lower, then $A \succ B$ but $\mathcal{C}_{\mathcal{P}}(A, B) < 1$.

So far, we have seen that $\mathcal{C}_{\mathcal{P}}$ satisfies most of the properties. Unfortunately, since it does not satisfy Property 1, not all cases of $A \succ B$ can be identified by $\mathcal{C}_{\mathcal{P}}$. We now propose a dominance measure that satisfies Property 1 and, thus, can be used to identify cases in which $A \succ B$.

2.3.2 $\mathcal{C}_{\mathcal{D}}$: dominance rate

Intuitively, the *dominance rate* is a dominance measure that quantifies the extent to which A has a lower cumulative distribution function than B, normalized by the portion of the probability densities with different cumulative distributions.

Definition 11 *(Dominance density function) Let A and B be two continuous random variables defined in a connected set $N \subseteq \mathbb{R}$. We define the dominance density function as follows:*

$$\mathcal{D}_{A,B}(x) = \begin{cases} g_A(x) \cdot k_A & \text{if} \quad G_A(x) > G_B(x) \\ -g_B(x) \cdot k_B & \text{if} \quad G_A(x) < G_B(x) \\ 0 & \text{otherwise.} \end{cases}$$

where $k_A = \left(\int_{\{x \in N \mid G_A(x) \neq G_B(x)\}} g_A(t)dt\right)^{-1}$ is the normalization constant and k_B is defined likewise.

Note that the dominance density function is not correctly defined when $\int_N |g_A(x) - g_B(x)|dx = 0$.

Definition 12 *(Dominance rate) Let A and B be two continuous random variables defined in a connected set $N \subseteq \mathbb{R}$. The dominance rate of A over B is defined as*

$$\mathcal{C}_{\mathcal{D}}(A,B) = \begin{cases} 0.5, & \text{if} \int_N |g_A(x) - g_B(x)|dx = 0 \\ 0.5 \int_N \mathcal{D}_{A,B}(t)dt + 0.5, & \text{otherwise.} \end{cases}$$

Basically, we are measuring the amount of mass of A in which $G_A(x) > G_B(x)$ minus the amount of mass of B in which $G_A(x) < G_B(x)$. This value is then normalized so that all sections in which $G_A(x) = G_B(x)$ are ignored, i.e. $\int_N \mathcal{D}_{A,B}(t)dt =$

$$\frac{\mathbb{E}_A[\mathcal{I}[G_A(x) > G_B(x)]]}{\mathbb{E}_A[\mathcal{I}[G_A(x) \neq G_B(x)]]} - \frac{\mathbb{E}_B[\mathcal{I}[G_A(x) < G_B(x)]]}{\mathbb{E}_B[\mathcal{I}[G_A(x) \neq G_B(x)]]}$$

Finally, we apply the linear transformation $l(x) = 0.5x - 0.5$ ensuring the dominance rate is defined in the interval $[0, 1]$ (instead of $[-1, 1]$), required to comply with Property 1.

From

i) $\mathcal{C}_{\mathcal{D}}(A,B) = 1 \iff A \succ B$ and

ii) $\mathcal{C}_{\mathcal{D}}(A,B) = 0 \iff B \succ A$,

we deduce that the dominance rate satisfies Property 1. Note that the previous deduction is only possible when g_A and g_B are bounded, as this implies that G_A and G_B are continuous. Specifically, it is enough to find a point in N in which $G_A(x) > G_B(x)$ to satisfy that $\int_{x \in \{t \in N \mid G_A(t) > G_B(t)\}} g_A(x)dx > 0$, and this point is guaranteed to exist when $A \succ B$ because of the definition of the dominance. The dominance rate is also a well behaved dominance measure, as it satisfies Properties 1, 2, 3, 4, 5, 6 and 8.

We have seen that the dominance measures $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ satisfy most of the properties listed in Section 2.2.2. As we will see in the next section, their values are related.

2.3.3 The relationship between $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$

In Section 2.2.1 we stated that $\mathcal{C}_{\mathcal{P}} = 1$ is a stronger condition than $\mathcal{C}_{\mathcal{D}} = 1$, because $\mathcal{C}_{\mathcal{P}}(A, B) = 1$ implies that for all x in N that $G_A(x) < 1$, $G_B(x) = 0$. On the other hand, $\mathcal{C}_{\mathcal{D}} = 1$ implies that $A \succ B$ (the two conditions in Definition 9), which is weaker. In the diagram below, we show the values of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ that imply other values of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$. Each arrow can be interpreted as an implication. The implications are transitive: i.g. $\mathcal{C}_{\mathcal{D}}(A, B) = 1$ implies $\mathcal{C}_{\mathcal{P}}(A, B) > 0.5$.

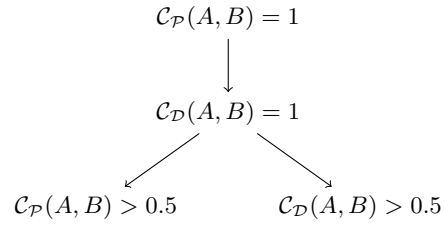


Fig. 2.8: Implications between the values of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$.

2.3.4 Estimating $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$

In the previous sections, we have assumed that the random variables A and B are known, but usually, we only have a few observed values from each random variable. Therefore, it may be interesting to estimate $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ from the observed samples. With this purpose, we propose the following empirical estimates of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$.

Definition 13 *(estimation of $\mathcal{C}_{\mathcal{P}}$) Let A and B be two continuous random variables and $A^n = \{a_1, ..., a_n\}$ and $B^n = \{b_1, ..., b_n\}$ their n observations respectively. We define the estimation of the probability that $A < B$ as*

$$\widetilde{\mathcal{C}_{\mathcal{P}}}(A^n, B^n) = \sum_{i,k=1...n} \frac{\text{sign}(b_k - a_i)}{2n^2} + \frac{1}{2}.$$

This estimator is well known in the literature because it is the U statistic of the Mann-Whitney test [Mann and Whitney, 1947].

Definition 14 *(estimation of $\mathcal{C}_{\mathcal{D}}$) Let A and B be two continuous random variables and $A^n = \{a_1, ..., a_n\}$ and $B^n = \{b_1, ..., b_n\}$ their n observations respectively. Let $C_{2n} = \{c_j\}_{j=1}^{2n}$ be the list of all the sorted observations of A^n and B^n where c_1 is the smallest observation and c_{2n} the largest. Suppose that $a_i \neq b_k$ for all $i, k = 1, ..., n$. We define the estimation of the dominance rate as*

$$\widetilde{\mathcal{C}_{\mathcal{D}}}(A^n, B^n) = \sum_{j=1}^{2n} \frac{\mathcal{I}(\hat{G}_A(c_j) > \hat{G}_B(c_j) \wedge c_j \in A^n)}{2n} -$$

$$\sum_{j=1}^{2n} \frac{\mathcal{I}(\hat{G}_A(c_j) < \hat{G}_B(c_j) \wedge c_j \in B^n)}{2n} + \frac{1}{2}.$$

where $\mathcal{I}$ is the indicator function and $\hat{G}_A(x)$ and $\hat{G}_B(x)$ are the empirical distributions estimated from A^n and B^n respectively.

For simplicity, this estimator of the dominance rate assumes there are no repeated samples. However, it can be extended to take into account repeated values (see Appendix 7.2.3).

2.4 Cumulative difference-plot

In this section, we propose a graphical method called *cumulative difference-plot* that shows the estimations of $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ decomposed by quantiles: $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ can be visually estimated from the difference plot. In addition, the proposed plot allows a comparison of quantiles of the two random variables. The proposed approach also models the uncertainty associated with the estimation of the *cumulative difference-plot* from the data.

2.4.1 Quantile random variables

From a practical point of view, it is unlikely that the probability densities of the compared random variables A and B are known. Usually, we only have n observations $A^n = \{a_1, ..., a_n\}$ and $B^n = \{b_1, ..., b_n\}$ from each random variable. The proposed *cumulative difference-plot* is based on two random variables Y_A and Y_B that are defined with these observations. Specifically, we define the densities of the two *quantile random variables* Y_A and Y_B as a mixture of several uniform distributions in the interval $[0, 1]$.

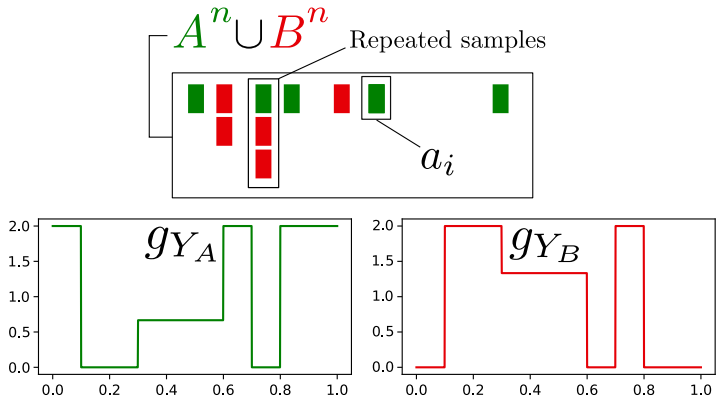


Fig. 2.9: An example of the probability density functions of Y_A and Y_B given the observed samples $A^n \cup B^n$.

The uniform distributions in the quantile random variables are placed according to their rank in $A^n \cup B^n$. Assuming no repetitions, for each value k in $A^n \cup B^n$, its corresponding kernel is centered in $\frac{\text{rank}(k)+0.5}{2n}$ where $\text{rank}(k)$ is the ranking of k in $A^n \cup B^n$. The kernels have a bandwidth of $1/4n$, ensuring that the sum of the densities of Y_A and Y_B is constant. If there are repeated values in $A^n \cup B^n$, their corresponding kernel is placed at the middle of the previous and the next rank, and the width of the kernel is increased proportionally with respect to the number of repetitions. See Figure 2.9 for an example. In Appendix 7.2.2.1 we show how to compute the probability densities of Y_A and Y_B step by step.

A more simple approach would be to estimate and define the quantile random variables through the empirical cumulative distribution functions of the observed samples of A and B. However, the quantile random variables defined through uniform kernels have some interesting properties: they have the same $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ as the kernel density estimations of A and B (shown in Appendix 7.2.2.2). In

addition, $g_{Y_A}(x)+g_{Y_B}(x) = 2$ for all $x \in [0, 1]$. As we will later see, these properties are essential for the interpretation of the *cumulative difference-plot*.

2.4.2 Confidence bands

The *cumulative difference-plot* is based on the cumulative distribution functions of Y_A and Y_B, which are estimated from the observed samples. This means that we need to model the uncertainty associated with the estimations. Confidence bands are a suitable choice in this scenario: a confidence band is a region in which the cumulative distribution is expected to be with a certain confidence. The size of the band is determined by the number of samples and the desired level of confidence: a high number of samples or a low level of confidence are associated with a small band size. There is an extensive literature [Cheng and Iles, 1983, Steck, 1971, Cheng and Lies, 1988, Wang et al., 2013, Faraway and Myoungshic Jhun, 1990, Bickel and A. M. Krieger, 1989, Hall and Horowitz, 2013] on how to estimate the confidence bands of cumulative distributions, and, in this chapter, we use a simple bootstrap approach[1].

To illustrate how to interpret the confidence bands of the cumulative distributions of Y_A and Y_B, we will assume that we have observed $n = 400$ samples from each random variable A and B from Case 2 (see Figure 2.4 in Section 2.2.1). We show the 95% confidence bands of the cumulative distribution functions of Y_A and Y_B in Figure 2.10. The estimated cumulative distribution functions of Y_A and Y_B resemble the cumulative distribution functions of A and B from Figure 2.5b. However, there are several relevant differences. In Figure 2.10, we observe that Y_A and Y_B are defined in the interval $[0, 1]$, while the cumulative distribution functions of A and B are defined in the sample space. Each of the values in this interval can be used to deduce the distribution with the lowest quantile: at $x = 0.5$, the cumulative distribution function of Y_A is larger than the cumulative distribution function of Y_B, hence, the median of A is lower than the median of B. In addition, the sum of the density function of Y_A and Y_B is constant. As a result, unlike A

[1] The bootstrapping [Efron and Tibshirani, 1993] method involves considering the observed values as a population from which random samples with replacement are drawn. These samples are then used to estimate the upper and lower pointwise confidence intervals of the cumulative distribution of Y_A and Y_B. Since a pointwise estimation of the confidence interval is used, we can expect that a portion proportional to α will fall outside the confidence band.
Note that we are interested in having an overall confidence of $1 - \alpha$, thus, we want that the cumulative distributions of Y_A and Y_B are inside their confidence bands at the same time with this level of confidence [Goeman and Solari, 2014, Bauer, 1991]. This means that we have to use a higher confidence level for each band: $\sqrt{(1-\alpha)}$.

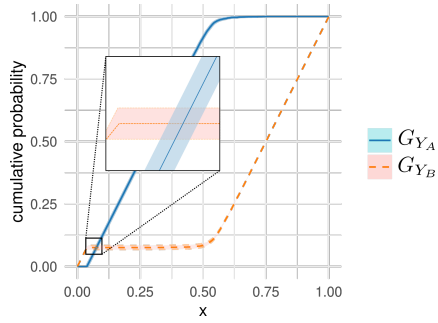


Fig. 2.10: The confidence bands of the cumulative distributions of the quantile random variables Y_A and Y_B corresponding to the distributions A and B in Case 2 (shown in Figure 2.4).

and B, the probability density functions of Y_A and Y_B do not have large areas where the probability density is zero.

2.4.3 The cumulative difference-plot

In this section, we introduce a new graphical method designed to visually analyze the dominance of A and B. Without loss of generality, a minimization[1] setting is assumed: lower values in A and B are preferred to higher values. It builds upon the difference function defined as

$$\begin{aligned} \text{diff}\colon [0,1] &\longrightarrow [-1,1] \\ x &\longmapsto G_{Y_A}(x) - G_{Y_B}(x), \end{aligned} \tag{2.2}$$

where $G_{Y_A}(x)$ and $G_{Y_B}(x)$ are the cumulative distribution functions of Y_A and Y_B, respectively.

[1] Note that if the random variables being compared take values in a maximization setting (higher values are preferred), then the random variables need to be redefined as the inverse with respect to the sum (this simply means the sampled values are multiplied by -1) before generating the cumulative difference-plot. With this change, the interpretation of the cumulative difference-plot is consistent and intuitive: for either minimization or maximization, on the left side of the cumulative difference-plot, the most desirable values that the random variables take are compared. If the difference is positive on the left side of the cumulative difference-plot, then the best values that A takes are better than the best values that B takes. Similarly, the worst values are compared on the right side of the cumulative difference-plot: if the difference is positive on this side, then the worst values of A are better than the worst values of B.

The *cumulative difference-plot* is the plot of the difference function (the difference between the cumulative distributions of Y_A and Y_B), including a confidence band. A positive value in the *cumulative difference-plot* can be interpreted as a quantile in which the cumulative distribution function of A is larger than the cumulative distribution function of B. Hence, if the difference is positive at 0.5, the median of A is lower than the median of B (assuming minimization). In this sense, the best values obtained by both random variables are compared on the left side, and the worst values are compared on the right side.

2.4.3.1 $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ in the cumulative difference-plot

$\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ can be directly obtained from the proposed plot. The integral of the difference between Y_A and Y_B is $\mathcal{C}_{\mathcal{P}} - 0.5$ (we prove this in Appendix 7.2.3). Formally, $\mathcal{C}_{\mathcal{P}} = 0.5 + \int_0^1 \text{diff}(x)dx$. However, in practice, $\mathcal{C}_{\mathcal{P}}$ can be visually estimated by adding 0.5 to the difference in the areas over and under 0. For the example shown in Figure 2.11, $\mathcal{C}_{\mathcal{P}} = 0.5 - \text{Area1} + \text{Area2}$. The difference can only be in the area highlighted in blue in the cumulative difference-plot. When the probability that $A < B$ is 1, the difference is at its maximum: in the cumulative difference-plot we see a line from $(x, f(x)) = (0, 0)$ to $(0.5, 1)$ and from $(0.5, 1)$ to $(1, 0)$. Similarly, when the probability that $A < B$ is 0, the difference between Y_A and Y_B is equal to the lowest possible values inside the light blue area.

By contrast, $\mathcal{C}_{\mathcal{D}}$ is represented in the plot as the total length in which the difference is positive minus the total length in which the difference is negative. Specifically,

$$\mathcal{C}_{\mathcal{D}} = \frac{\frac{\int_0^1 \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx}{2} + \frac{1}{2}}{\int_0^1 \mathcal{I}[\text{diff}(x) \neq 0]dx}, \tag{2.3}$$

where $\mathcal{I}$ is the indicator function (we prove this in Appendix 7.2.3). As an example, $\mathcal{C}_{\mathcal{D}}$ is proportional to Length2 − Length1 in Figure 2.11: it is higher than 0.5, because Length2 > Length1. In this example, there is no need to divide by the total length in which the difference is nonzero because the difference is zero in only a limited number of points. In such cases, $\mathcal{C}_{\mathcal{D}}$ can also be estimated as the total length in which $\text{diff}(x) > 0$. In the example in Figure 2.11, the estimation is $\mathcal{C}_{\mathcal{D}} = \text{Length2} \approx 0.75$. Note that Equation (2.3) is not correctly defined when Y_A and Y_B are equal, but this is an easy case to identify, as the difference is constantly 0.

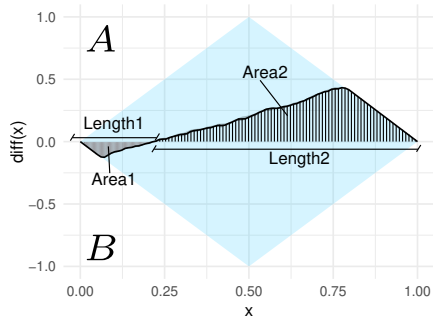


Fig. 2.11: The areas and lengths in the cumulative difference-plot that can be used to deduce $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$.

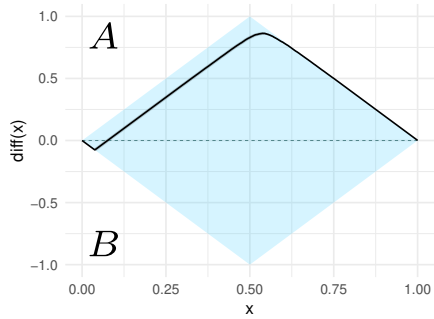


Fig. 2.12: The cumulative difference-plot for Case 2: the difference between the cumulative distribution functions of Y_A minus Y_B corresponding to the distributions A and B in Case 2 (shown in Figure 2.4).

2.4.3.2 Illustrative example

Figure 2.12 shows the cumulative difference-plot for the random variables A and B from Case 2 (their densities were shown in Figure 2.4). First, we see that the difference is both negative and positive, hence, neither random variable dominates the other. The difference is negative when $x = 0.05$ or lower. This can be interpreted as B having a smaller 5% quantile than A. The difference is positive otherwise, thus we deduce from the cumulative difference-plot that the 25%, the 50% (the median), the 75% and 95% quantiles are smaller in A than in B. In other words, the random variable B can take really low values with a small probability, but apart from these really low values, A takes lower values than B. This is also reflected by $\mathcal{C}_{\mathcal{D}}(A, B) > 0.75$, as deduced from $\text{diff}(x) > 0$ for all $x \in (0.25, 1)$.

The difference is also near its maximum value, implying that its integral is high and thus $\mathcal{C}_{\mathcal{P}}$ (the probability that $A < B$) is also near 1.

2.5 Related work

Statistical assessment of experimental results is a very studied research topic. In this section, we locate our proposal in the field and focus on similarities and differences with respect to other random variable comparison methods.

2.5.0.1 Visualizing densities

As mentioned in the introduction, it makes sense to model the performance of stochastic optimization algorithms as random variables. Therefore, statistical tools that compare random variables have become an increasingly important part of the analysis of experimental data. Among these tools, visualization techniques such as histograms or box-plots are usually applied before the rest of the methods. The advantage of these methods is their simplicity. If one of the random variables clearly takes lower values than the other, then these two methods effectively convey this message simply and naturally. Unfortunately, when both random variables have similar probability densities, these two methods might fail to represent the random variables in a way that makes it easy to compare them (example shown in Section 2.6.1).

The simplicity of these methods is also a drawback: for example, they give no information about the uncertainty associated with the estimates. The histogram suffers from the bin positioning bias [Thas, 2010, scikit-learn developers, 2021]. A kernel density estimation with the uniform kernel—considered to be the moving window equivalent of the histogram—overcomes this limitation [Thas, 2010], at the cost of using a more complex model. Similarly, the box-plot has a "non-injectivity" problem: very different data can still have the same box-plot [Matejka and Fitzmaurice, 2017, Chatterjee and Firat, 2007]. The violin-plot is an extension of the box-plot that overcomes the above limitation by combining the kernel density estimate of the random variables with the traditional box-plot [Hintze and Nelson, 1998]. The proposed cumulative difference-plot improves on these methods because it represents the data clearly, even when the two random variables being compared are similar.

2.5.0.2 Statistical null hypothesis tests

Statistical null hypothesis tests can be used to compare random variables without having to visually represent them. In a very general way, carrying out a statistical null hypothesis test involves the following: first, a null hypothesis is proposed. Under certain assumptions, the null hypothesis implies that a given statistic obtained from the data follows a known distribution. Then, assuming the null hypothesis is true, the probability of obtaining data with a more extreme statistic value[1] than the observed [Conover, 1980] is computed. When the probability under the null hypothesis of the observed statistic is lower than a predefined threshold, the null hypothesis is rejected and the alternative hypothesis is accepted [Greenland et al., 2016]. Usually, this threshold is set at an arbitrary but well established [Wasserstein and Lazar, 2016] $p = 0.05$, although recently, further reducing the threshold to $p = 0.005$ has been proposed [Benjamin et al., 2018, Ioannidis, 2018].

In the context of comparing two random variables A and B, in general, we cannot assume that a statistic obtained from the data follows a known distribution under the null hypothesis. In this case, a non-parametric test [Conover, 1980] is a suitable choice. Specifically, the Mann-Whitney test [Mann and Whitney, 1947] is a good choice, as the samples observed from the random variables are i.i.d for each random variable [2]. With this test, the null hypothesis is that $\mathcal{P}(A > B) = \mathcal{P}(B > A)$, and a possible alternative hypothesis is that $A \succ B$ [Mann and Whitney, 1947].

Statistical null hypothesis tests have some limitations: for example, the p-value does not separate between the effect size and the sample size [Benavoli et al., 2017, Calvo et al., 2019a]. In addition, rejecting the null hypothesis does not always mean that there is evidence in favor of the alternative hypothesis: it just means that the observed statistic (or a more extreme statistic) is very unlikely when the null hypothesis is true.

To show this, we generate 400 samples of the distributions A and B from Case 2 (density functions shown in Figure 2.4) and we apply the Mann-Whitney test, rejecting the null hypothesis when $p < 0.005$. If we repeat this experiment 10^4 times (with different samples each time), the null hypothesis is rejected every

[1] The definition of what *data with a more extreme statistic value* is not the same for every statistical null hypothesis test, and it depends on the test being used.

[2] For paired data, the Wilcoxon signed-rank test [Wilcoxon, 1945] or the sign test [Conover, 1980] should be used. However, in the context of this chapter, the samples observed from the random variables are not paired. In this chapter, we consider the Mann-Withney test as it is probably the most well known non-parametric test for unpaired data, although take into account that more modern alternatives have been proposed [Ledwina and Wyłupek, 2012, Baumgartner et al., 1998, Biswas and Ghosh, 2014].

time[1]. However, $A \not\succ B$ and $B \not\succ A$, implying that the alternative hypothesis is not true. Note that the proposed cumulative difference-plot (shown in Figure 2.12) avoids this problem because it correctly points out that neither random variable dominates the other one, for the same case and with the same number of samples.

2.5.0.3 Bayesian analysis

As an alternative [Benavoli et al., 2017, Calvo et al., 2019a] to the limitations of statistical null hypothesis tests, Bayesian analysis has been proposed. Bayesian analysis [Gelman, 2014, Bernardo and Smith, 2009] estimates the probability that a hypothesis is true, conditioned to the observed data. This estimation requires the prior probabilities of the hypotheses and the data, but usually, they are assumed to follow a distribution that gives equal probability to all hypotheses and data. Recently a Bayesian version of the Wilcoxon signed-rank test [Benavoli et al., 2017, 2014] has been proposed. In this chapter, we will consider the simplex-plot of its posterior distribution. For convenience, in the rest of the chapter, we will call it *simplex-plot*.

Once the posterior distribution is known, the probability that the difference between a sample from A and a sample from B is in the intervals $(-\infty, -r)$, $[-r, r]$ or $(r, +\infty)$ can be computed. These probabilities can be interpreted as the probability that $A > B$, $A = B$ and $B > A$, where two samples a and b are considered equal when $|a - b| \leq r$. Note that the simplex-plot is just a convenient representation of the posterior distribution, where 'rope' or *range of practical equivalence* denotes hypothesis $A = B$ (when the difference is in the interval $[-r, r]$).

We computed the simplex-plot (Figure 2.13) with the 400 samples of A and B from Case 2 obtained in Section 2.4.2. Two samples were considered equal when their difference is lower than $r = 10^{-3}$, and we used the prior proposed by Benavoli et al. [2017]. We can deduce from this figure that the hypothesis $B > A$ is much more likely than $A = B$ or $B > A$.

The simplex plot summarizes the data through the probabilities of $B > A$, $A = B$ or $B > A$, but does not offer any additional information: we cannot deduce from these probabilities in which intervals the values of a random variable are lower than the other. In this sense, the cumulative difference-plot is a more detailed comparative visualization. Specifically, the observation that the 1% lowest values of A are lower than the 1% lowest values of B cannot be deduced from the simplex-plot, while it is easy to see in the cumulative difference-plot. Also, the cumulative difference-plot shows a comparison of the cumulative distributions through the dominance rate, while the simplex-plot does not.

[1] The source code to replicate this experiment is available in the file mann_whitney_counter_example.R in our Github repository.

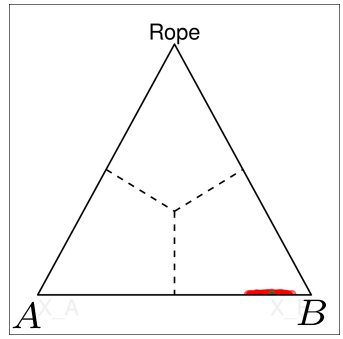


Fig. 2.13: The simplex-plot computed with the package *scmamp* [Calvo and Santafé Rodrigo, 2016a] of the posterior distribution for Case 2.

2.5.0.4 Other plots in the interval [0, 1]

The probability-probability plot is defined as

$$PP : [0,1] \rightarrow [0,1]^2 : p \rightarrow (p, G_A(G_B^{-1}(p))).$$

As proposed by Schmid and Trede [1996], it can be interpreted via the integral of the non-negative part, which represents the amount of violation against the hypothesis that A dominates B.

The quantile-quantile plot [Thas, 2010, Wilk and Gnanadesikan, 1968] is defined as

$$QQ : [0,1] \rightarrow N^2 : p \rightarrow (G_A^{-1}(p), G_B^{-1}(p)),$$

and it is a natural way to visualize the differences in quantiles of A and B in N (the domain of definition of the random variables).

The quantile-quantile plot also allows a comparison between quantiles, just like the *cumulative difference-plot*. However, the cumulative difference-plot proposed in this chapter is distinct from the two plots above in three aspects: i) the proposed cumulative difference-plot is defined directly from the observed samples. Because of its definition, it has a confidence band built-in, which allows the uncertainty associated with the estimation to be directly interpreted within the plot. ii) The proposed cumulative difference-plot contains several statistics simultaneously. Specifically, the estimated $\mathcal{C}_{\mathcal{D}}$, $\mathcal{C}_{\mathcal{P}}$ and the comparison of the quantiles can be visually interpreted. iii) The proposed plot is just the difference of two cumulative distributions (G_{Y_A} and G_{Y_B}), and thus, unlike in the pp-plot and qq-plot mentioned above, it can be defined without the need of the inverse function. The random variables Y_A and Y_B have the same $\mathcal{C}_{\mathcal{D}}, \mathcal{C}_{\mathcal{P}}$ as the kernel density estimations of the original distributions, and therefore, we can think of the cumulative

difference-plot as the difference between the cumulative distribution function of two simpler versions of the original random variables.

2.6 Experimentation with the cumulative difference-plot

To illustrate the applicability of the proposed methodology, in the following, we re-evaluate the experimentation of a recently published work. In a recent paper, [Santucci et al., 2020] introduced a gradient-based optimizer for solving problems defined in the space of permutations (from now on *PL-GS*). In their experimentation, they compared it with an estimation of distribution algorithm [Larrañaga and Lozano, 2001a] (from now on *PL-EDA*). These two algorithms were tested in a set of 50 problem instances of the linear ordering problem [Schiavinotto and Stützle, 2004]. The performance of each algorithm in each instance was estimated with the median relative deviation from the best-known objective value, with $n = 20$ repetitions. From now on, we call *score* to the relative deviation from the best-known objective value and note that a low score is better than a high score, as it means that the objective value found is closer to the best-known.

In the work by Santucci et al. [2020], when the score of one of the algorithms was at least 10^{-4} higher than the other, it was considered that one of the algorithms performed better than the other in that instance. [Santucci et al., 2020] concluded that both algorithms performed equally in the instance *N-t70n11xx*, as the median scores were exactly the same for both algorithms in this instance.

In the following, we take a closer look at the performance of *PL-EDA* and *PL-GS* in this problem instance by comparing $n = 10^3$ measurements of the score from each algorithm. We increase the sample size from $n = 20$ to $n = 10^3$ because the difference between the performance of the algorithms is small. With a sample size of $n = 20$, the uncertainty is too high to come to any meaningful conclusion (regardless of the statistical methodology considered). With this increased sample size, we obtained more accurate estimates of the median scores—*PL-GS* $= 0.00407$, *EDA* $= 0.00433$, lower is better—and *PL-GS* obtains a better value by a difference higher than 10^{-4}.

2.6.1 Step 1: Visualization

Figure 2.14 shows the histogram of the scores. It can be deduced from the figure that neither algorithm clearly produces better scores. In particular, neither algorithm dominates the other: *PL-EDA* has a longer tail both to the right and to the left. Also, notice that the score of the algorithms is not normally distributed:

PL-GS has a bimodal shape, and *PL-EDA* has a very long tail to the right (while the tail to the left is shorter).

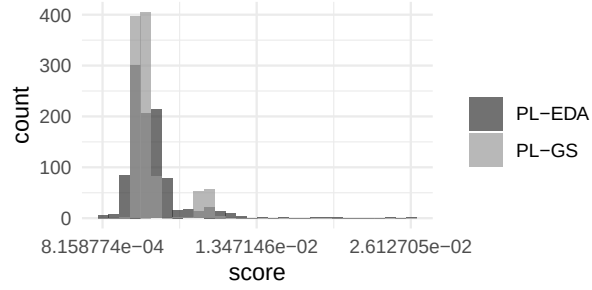


Fig. 2.14: Histogram of the scores obtained in the instance *N-t70n11xx*. Lower is better.

Figure 2.15 shows the box-plot and the violin-plot of the data. Both algorithms have a similar median, but due to the high number of outliers [Carreño et al., 2020], it is difficult to compare the scores of the algorithms with the box-plot. The same happens with the violin-plot.

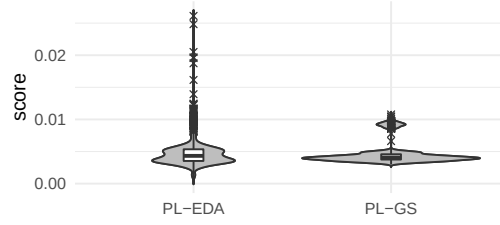


Fig. 2.15: Box-plot and violin-plot of the scores obtained in the instance *N-t70n11xx*. Lower is better.

2.6.2 Step 2: Comparing PL-GS with PL-EDA

Sometimes, visualization is enough to compare the performance of two algorithms: if one of the algorithms always performs better than the other, there is no need for further analysis. However, in this case, the three visualization methods considered

(histogram, box-plot, and violin-plot) have not been able to summarize the scores obtained with the algorithms in a way that enables an easy comparison. In the following, we further study the scores of the algorithms with statistical tests, the simplex-plot, and the cumulative difference-plot.

2.6.2.1 Mann-Whitney test

Applying the Mann-Whitney test we obtain a p-value of $p = 0.035$, lower than the usually used 0.05 threshold. With $p < 0.05$, we reject the null hypothesis and accept the alternate hypothesis: the random variable associated with the score of *PL-GS* dominates *PL-EDA*. Note that neither rejecting the null hypothesis nor a small p-value reflect the magnitude of the difference in score of the algorithms. In addition, as stated when we studied the histogram, we known that it is unlikely that *PL-GS* dominates *PL-EDA*.

2.6.2.2 Simplex-plot

We show the simplex-plot [Benavoli et al., 2017] of the scores in Figure 2.16. Following the criterion by Santucci et al. [2020], we considered that two scores are equal when they differ by less than $r = 10^{-4}$. Unlike in the statistical test, one can deduce the probability that one of the algorithms has a better score than the other from simplex-plot: it is more likely that *PL-GS* takes a lower value than *PL-EDA*. A closer position in the plot to *PL-EDA* indicates a higher probability of measuring a higher score in *PL-EDA* than in *PL-GS*. Specifically, from the simplex-plot shown in Figure 2.16, we can deduce that given two samples b_{gs} and a_{eda} of the scores of *PL-GS* and *PL-EDA* respectively,

$$\mathcal{P}(a_{eda} < b_{gs}) < \mathcal{P}(b_{gs} < a_{eda}).$$

However, the difference in these probabilities is small. Also, the probability that $\mathcal{P}(b_{gs} = a_{eda})$ is low (no data points near 'rope').

2.6.2.3 Cumulative difference-plot

We show the 95% confidence cumulative difference-plot in Figure 2.17. From this plot, we can deduce the following:

1. $\mathcal{P}(a_{eda} < b_{gs})$ and $\mathcal{P}(b_{gs} < a_{eda})$ have similar probabilities, as $\mathcal{C}_{\mathcal{P}}(\textit{PL-EDA}, \textit{PL-GS}) \approx 0.5$. However, The area under $\text{diff}(x) = 0$ is a little larger than the area over $\text{diff}(x) = 0$, hence $\mathcal{P}(a_{eda} < b_{gs})$ is a little smaller than $\mathcal{P}(b_{gs} < a_{eda})$.

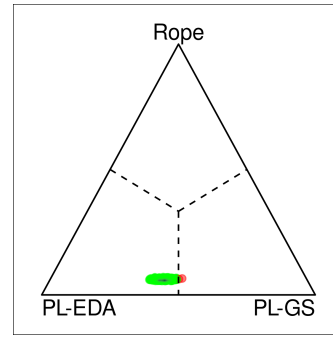


Fig. 2.16: Simplex-plot of *PL-GS* and *PL-EDA* in the instance *N-t70n11xx*. A closer position in the plot to *PL-EDA* indicates a higher probability of measuring a higher score in *PL-EDA* than in *PL-GS*. A low score is preferred to a high score.

2. Neither algorithm dominates the other one, and what is more, $\mathcal{C}_{\mathcal{D}}(\textit{PL-EDA}, \textit{PL-GS}) \approx 0.5$.
3. The difference is positive when $x < 0.3$, and therefore, if we only consider the best 30% values of both algorithms, *PL-EDA* dominates *PL-GS*.
4. The difference is negative when $x > 0.98$. In this case, we conclude that if we only consider the worst 2% values of *PL-EDA* and *PL-GS*, then *PL-GS* dominates *PL-EDA*.
5. These "worst" 2% values are much less likely than the "best" 30% values mentioned in 3), as the estimated probability of these "best" and "worst" values is 0.3 and 0.02 respectively.
6. The difference is negative at $x = 0.5$ and at $x = 0.75$. This can be interpreted as PL-GS having a better median and a better 75% quantile.

Summarizing the above points, we conclude that the performance of the algorithms is quite similar, and *PL-EDA* takes both better and worse scores than *PL-GS*. The probability that *PL-EDA* takes these better values is much higher than the probability that it takes worse values. Therefore, if we are in a setting in which repeating the execution of the algorithms is reasonable, *PL-EDA* is a much better algorithm. On the other hand, if it is critical to avoid really bad values, then *PL-GS* would be preferred. With an increased number of samples, it might be possible to better compare the algorithms (it would reduce the uncertainty associated with the size of the confidence band).

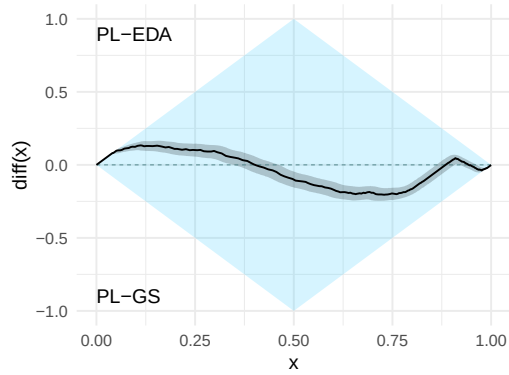


Fig. 2.17: The cumulative difference-plot of 95% confidence of the objective values obtained by *PL-EDA* and *PL-GS* in the instance *N-t70n11xx*.

2.7 Assumptions and limitations

In the following, we briefly summarize the assumptions that the cumulative difference-plot requires and comment on a few caveats.

2.7.1 Assumptions

Correctly using the proposed cumulative difference-plot requires that the following three assumptions are satisfied. The first assumption is that all samples of both A and B are i.i.d, consequently, it should not be used with paired data. This is also an assumption made by the Mann-Whitney test.

The second assumption is that the values of the random variables represent a minimization setting: lower values are preferred to higher values. To apply the proposed method in a maximization setting, it is enough to redefine the objective function by multiplying it by -1.

The third assumption is that A and B are continuous random variables defined in a connected subset of $\mathbb{R}$. This also implies that the cumulative distribution functions of A and B are continuous and that their probability density functions are bounded. Although having a bounded density means that there should never be two identical samples—the probability of observing two independent equal samples is 0 with a bounded density—, in reality, the proposed cumulative difference-plot can deal with repeated samples. To do so, when defining the kernel density estimations of Y_A and Y_B in Section 2.4, repeated samples were assigned the same

rank. Then, the size of the uniform distributions was adjusted (with the γ function) ensuring that the sum of the estimated densities of Y_A and Y_B remains constant even in the case of repeated observations.

2.7.2 Limitations and future work

Just like with other methods, the number of samples determines in part the stability of the results. With a small sample size, the confidence band of the cumulative difference-plot will be larger. There are three reasons why a larger sample size increases the stability of the plot: i) we are doing a kernel density estimation, and a higher sample size [Danica and power, 2009] implies that the estimation is closer to the real distribution, ii) the bootstrap method also requires several samples to be meaningful [Chernick, 2011, Hall, 2013] and iii) the sample size needs to be reasonable with respect to the quantiles being estimated. For example, it would not make sense to use 10 samples to estimate a 1% quantile. In all of these cases, however, determining what is a *too small* sample size is a highly debated question, and is beyond the scope of this chapter. To be on the safe side, we recommend using a sample size of at least $n = 100$. It is worth noting that this was arbitrarily chosen, and a suitable sample size should be chosen depending on the desired conclusions (for example, comparing small and big quantiles requires more data). With $n = 100$ we ensure that the comparison of 1% quantiles in the cumulative difference-plot is meaningful.

The most obvious limitation of the proposed approach is in its applicability: it should only be used in case of doubt between two random variables, and when none of the random variables dominates the other one. Otherwise, there are more suitable alternatives such as Bayesian analysis [Calvo et al., 2019a, Benavoli et al., 2014], or directly comparing box-plots. For instance, if we take 10^3 samples of A and B and all samples of A are lower than all samples of B, then there is no need for further statistical comparison, as the results speak for themselves.

The proposed approach assumes A and B are continuous random variables and that all samples of both A and B are i.i.d, and consequently, it cannot be used with paired data. As future work, the proposed methodology could be extended for paired data and ordinal random variables. Also, the bootstrap method is the slowest part of the *cumulative difference-plot*, especially as the number of samples increases. To increase the computation speed, this slow part was written in C++ (the rest of the package was written in *R* [R Core Team, 2020]). However, its speed can probably be further improved with a better implementation.

2.8 Conclusion

In this chapter, we approached the problem of comparing the performance of two optimization algorithms, in terms of which of them takes lower values. We proposed eight desirable properties for dominance measures: functions that compare random variables in the context of quantifying the dominance. We showed that, at most, a dominance measure can only satisfy seven out of the eight. Among the measures in the literature, we found out that *the probability that one of the random variables takes lower values than the other* satisfies 7 of those properties. However, it fails to satisfy Property 1, hence it cannot be used to determine when one of the random variables stochastically dominates the other. To to complement this, we introduced a new dominance measure: the dominance rate, which quantifies how much higher one of the cumulative distribution function is than the other and satisfies all properties except Property 7.

Based on the above, we proposed the *cumulative difference-plot*, a graphical visualization that allows two random variables to be compared in terms of which of them takes lower values. This cumulative difference-plot contains a comparison of the quantiles, in addition to allowing a graphical estimation of the dominance rate and the probability that one of the random variables takes lower values than the others. It also models the uncertainty associated with the estimate through a confidence band. Finally, in Section 2.6 we showed that the proposed methodology is suitable to compare two random variables, especially when they take similar values and other methods fail to give detailed and clear answers.

Supplementary Material

Code to Reproduce the Results

Alongside the chapter, we provide the code to generate the figures in this chapter and replicate the experimentation. For instructions on how to install the dependencies and replicate the results, refer to the README.md file in the GitHub repo https://github.com/EtorArza/SupplementaryPaperRVCompare.

R Package RVCompare

Furthermore, we created the R *RVCompare* for a convenient application of the methodology proposed in the chapter. Users can compute the $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ of two distributions, given their probability density functions. Furthermore, it can be used to produce the proposed cumulative difference-plot, given the observed data. The package can be directly installed from CRAN and is also available in the GitHub repo https://github.com/EtorArza/RVCompare.

Interactive Example

An interactive example is available at https://etorarza.github.io/pages/2021-interactive-comparing-RV.html that demonstrates how Property 1 and Property 7 are mutually exclusive.

3

Kernels of Mallows Models under the Hamming Distance for solving the Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is a well-known permutation-based combinatorial optimization problem with real applications in industrial and logistical environments. Motivated by the challenge that this NP-hard problem represents, it has captured the attention of the optimization community for decades. As a result, a large number of algorithms have been proposed to tackle this problem. Among these, exact methods are only able to solve instances of size $n < 40$. To overcome this limitation, many heuristic methods have been applied to the QAP.

In this chapter, we follow this direction by approaching the QAP through Estimation of Distribution Algorithms (EDAs). Particularly, a non-parametric distance-based exponential probabilistic model is used. We compare four distances for permutations and show that the Hamming distance is the most suitable for the QAP. We study the properties of the distances when applying the probabilistic model to the objective function of the QAP. Particularly, we focus on the smoothness of the objective function and the number of components changed. Based on this analysis, we argue that the Hamming distance is a suitable distance for assignment type problems in general.

3.1 Introduction

The Quadratic Assignment Problem (QAP) [Koopmans and Beckmann, 1955] is a well-known combinatorial optimization problem. Along with other problems, such as the traveling salesman problem, the linear ordering problem and the flow-shop scheduling problem, it belongs to the family of permutation-based problems [Ceberio, 2014]. The QAP has been applied in many different environments over the years, to name but a few notable examples, selecting optimal hospital layouts [Hahn and Krarup, 2000], optimally placing components on circuit boards [Rabak and Sichman, 2003], assigning gates at airports [Haghani and Chen, 1998], and optimizing data transmission [Mittelmann and Salvagnin, 2015].

Algorithm 2: Estimation of Distribution Algorithm [Ceberio, 2014]

Parameters :
P_s: The size of the population used by the EDA.
M: The size of the set of selected solutions.
S: The number of new solutions generated at each iteration.

1 $D_0 \leftarrow$ initialize population of size P_s and evaluate the population
2 **for** *t=1,2,... until stopping criterion is met* **do**
3 $\quad D_{t-1}^{sel}$ select $M \leq N$ from D_{t-1} according to a selection criterion
4 $\quad p_t(x) = p(x|D_{t-1}^{sel}) \leftarrow$ estimate a probability distribution from D_{t-1}^{sel}
5 $\quad D_t^S \leftarrow$ sample S individuals from $p_t(x)$ and evaluate the new individuals
6 $\quad D_t \leftarrow$ create a new population of size P_s from D_{t-1} and D_t^S

Sahni and Gonzalez [Sahni and Gonzalez, 1976] proved that the QAP is an NP-hard optimization problem, and as such, no polynomial-time exact algorithm can solve this problem unless P=NP. In this sense, until recently, only a few instances of size up to 36 were solved using exact solution algorithms. In fact, and exceptionally, only some instances of size 64 and 128 have been solved by using a combination of three strategies: reformulation to a suitable Mixed-Integer Linear Programming, exploiting the sparsity and symmetry of some particular instances, and a Branch and Bound algorithm (B&B) [Fischetti et al., 2012]. These strategies, however, require the instance to be highly symmetric, and in general, this cannot be guaranteed. Most of the exact methods for the QAP are based on the B&B algorithm [Brixius and Anstreicher, 2004]. In order to overcome the high computation cost required by these algorithms, Astreicher et al. [Anstreicher et al., 2002] proposed a grid computing implementation of B&B. Under this technique, this algorithm can be distributed over the internet, forming a computational grid,

with the advantage of bringing down the costs and increasing the availability of parallel computation power.

Unfortunately, despite the previous improvements, in the general case, it is still computationally unfeasible to use exact algorithms for medium and large size instances ($n > 40$). In response to this drawback, the community of combinatorial optimization has proposed a variety of metaheuristic algorithms to tackle the QAP. A few of these proposed methods include Genetic Algorithms [Drezner, 2003, Tate and Smith, 1995], Tabu Search [Skorin-Kapov, 1990, James et al., 2009a], Simulated Annealing [Mickey R. Wilhelm Ph.D. and Thomas L. Ward Ph.D., 1987], Ant Colony Optimization [Gambardella et al., 1999], Memetic Algorithms [Merz and Freisleben, 2000] and Particle Swarm Optimization Algorithms [Liu et al., 2007].

Currently, three of the best performing approaches for the QAP are Cooperative Parallel Tabu Search (CPTS) [James et al., 2009b], a Memetic Search algorithm (MS) [Benlic and Hao, 2015b] and the Parallel Hyper-Heuristic on the Grid (PHHG) [Dokeroglu and Cosar, 2016]. In particular, CPTS is based on the successful robust tabu search implementation [Taillard, 1991]. This tabu search implementation is simpler and is executed in parallel with a shared pool of solutions, aiming to promote both the quality and the diversity of the solutions available to each tabu search execution. Memetic search algorithms combine population based algorithms with local search procedures. The MS implementation [Benlic and Hao, 2015b] uses a uniform crossover operator, a breakout local search procedure (a local search procedure with perturbation mechanics to overcome local optima) [Benlic and Hao, 2013], a fitness based replacement strategy, and an adaptive mutation strategy. Lastly, PHHG executes different metaheuristic algorithms in parallel, and based on the performance of those algorithms, repeatedly executes the most successful metaheuristics. Even though these three methods are very different from each other, they all share a property: they are highly complex hybrid procedures.

Estimation of Distribution Algorithms (EDAs) [Larrañaga and Lozano, 2001b] have also been used to solve the QAP. Zhang et al. presented an hybrid approach of guided local search and a first marginal based EDA for the QAP [Zhang et al., 2003]. Pradeepmon et al. [Pradeepmon et al., 2018] applied a hybrid EDA to the keyboard layout problem, which is modeled as a QAP problem. Previous EDA references use hybridization techniques to improve the performance of the algorithms, nevertheless, the probability models used are simplistic and are not suited to the characteristics of the QAP. In this chapter, we investigate probability models specially suited to deal with the QAP, and introduce a new EDA proposal based on these models.

An Estimation of Distribution Algorithm (EDA) [Larrañaga and Lozano, 2001b] is a population-based evolutionary algorithm (see Algorithm 2 for the general pseudo-code of EDAs). Starting with an initial population (line 1), a subset of the best solutions is selected (line 2). Subsequently, a probability model is learned based on these selected permutations (line 4). Next, new solutions are sampled from the probability model, and their objective value is computed (line 5). Finally, the new solutions are combined with the selected solutions to create a new population (line 6). This process is repeated until a stopping criterion is met, such as exceeding a certain time constraint or a maximum number of iterations.

As reported frequently in the literature, the behavior of an EDA depends highly on the probability model used in the learn-sample cycle, as this is the core component of the algorithm. When considering permutation problems, EDAs can be classified into three categories according to the probability domain of the model used [Ceberio et al., 2012]. In the first group, we have EDAs that were designed to solve problems on the combinatorial domain. Specifically, EDAs that were designed for the set $\{(k_1, ..., k_n) \in \mathbb{N}^n : k_i \in \{1, ..., n\}i\}$ (denoted as $[n]^n$ in this chapter) can be adapted for the permutation space. This adaptation is based on the idea that the solution space of the QAP is a subset of $[n]^n$, defined by adding the constraint $k_i \neq k_j$. Specifically, the solution space of the QAP is the set of every permutation of size n, denoted as $\mathcal{S}^n$ in this thesis. Therefore, given a set of solutions from $\mathcal{S}^n$, a probability model can be learned in $[n]^n$. Consequently, in order to obtain solutions that are in $\mathcal{S}^n$, the sampling procedure must be adapted to guarantee that the new samples are in $\mathcal{S}^n$.

In the second group, we have EDAs that were originally designed to deal with continuous domain problems. Next, in order to deal with permutations, these EDAs use a mapping $\gamma : \mathbb{R}^n \longrightarrow \mathcal{S}^n$, such that, given a real vector $v \in \mathbb{R}^n$, $\gamma(v)$ denotes the order of the items in v. EDAs for continuous problems transform each permutation σ_i in the population into a real vector v_i, making sure $\gamma(v_i) = \sigma_i$ is satisfied for all the individuals in the population. Then, a probability model is learned on $\mathbb{R}^n$ from these real vectors, and new real vectors are sampled from this probability model. Finally, γ is applied to all the new real vectors, to obtain the new solutions. One of the major drawbacks of these models, as stated by Bosman et al. [Bosman and Thierens, 2001], are the overheads introduced by the sorting of the real vectors, which can be costly in certain situations. Another limitation of the models in this group comes from the large redundancy introduced by the codification, since a permutation can be mapped by infinite real vectors [Ceberio et al., 2012].

Finally, in the third group, we have the EDAs that use probability models that define a probability distribution on $\mathcal{S}^n$. Among these, we find probability models based on order statistics, such as the Plackett-Luce [Plackett, 1975] and Bradley-Terry [Hunter, 2004] models, or those that rely on distance-based distributions on

$\mathcal{S}^n$ such as the Mallows Model (MM) [Mallows, 1957] and the generalized Mallows Model (GMM) [Fligner and Verducci, 1986a]. For these EDAs, the solution space of the problem is also the space onto which the probability distribution is defined, making them a more natural choice. The MM is an exponential distribution that can be seen as analogous to the normal distribution over the group $\mathcal{S}^n$. Recently, it has been shown that the distance-metric under which an MM is defined influences the performance of the EDA [Ceberio et al., 2015c]. In this sense, most of the MMs presented in the literature are based on the Cayley, Kendall's-τ and Ulam distances. For example, Ceberio et al. have applied many variants of MM based EDAs to different permutation problems, including Cayley based Kernels of MM [Ceberio et al., 2015d] and Generalized MM [Ceberio et al., 2014a] on the QAP. In addition, a MM variant for the bi-and tri-objective FSP was proposed by Zangari et al. [Zangari et al., 2018]. In fact, Ceberio et al. obtained state-of-the-art results in the FSP by hybridizing a MM EDA with a variable neighborhood search [Ceberio et al., 2014b]. This illustrates the importance of developing efficient and adequate probability models.

In addition to the three metrics mentioned above, there are other distance-metrics on $\mathcal{S}^n$ that have not previously been considered in EDAs, such as the Hamming distance-metric [Irurozki et al., 2016, 2019a]. The Hamming distance between two permutations counts the number of point-wise disagreements and is a natural choice for measuring the distance between assignments or matchings.

In this chapter, we take a step forward in the development of EDAs specific for assignment type permutation problems by using probabilistic models based on the Hamming distance. Specifically, the proposed probabilistic model is a Mallows model-based kernel density that uses the Hamming distance. The goal of this chapter is not to present a state-of-the-art algorithm. Instead, a methodological contribution is made to the design of probabilistic models of EDAs for permutations. Particularly, we show why the probability model used in the introduced EDA is suitable for the QAP, one of the most popular assignment type permutation problems. In addition, by studying the properties of the QAP, and based on the experimentation results, we claim that the Hamming-based kernel density probability model proposed in this chapter is suited to be used in EDAs for solving assignment problems in the solution space of permutations of a certain size[1].

Another relevant feature of the MM is that it is a unimodal model, and is centered at a given central permutation. The unimodality and symmetry properties imposed by the MM can be too restrictive in certain contexts, not allowing multimodal scenarios to be accurately modelled [Lebanon and Mao, 2008]. However, the MM can be suitable as a building block in more complex models. An alter-

[1] In order to take a step forward in the design of EDAs and avoid misinterpretations, in this chapter, we will only consider EDAs in their base form (no hybridization).

native that breaks these strong assumptions is the kernel density estimate using Mallows kernels (KMMs). Instead of having a central permutation, KMMs spread the probability mass by using a non-parametric averaging of MMs centered at each solution. This allows the distribution to model probability distributions more accurately over the space of permutations when the strong assumptions of the MM are not fulfilled by the set of solutions.

Taking advantage of this flexibility, the EDA approach presented in this manuscript implements a KMM under the Hamming distance. For the sake of analyzing the performance of the proposed algorithm, we conduct three experiments. First, we compare the proposed approach to other Hamming-based MM approaches. Then, we see how it compares to other EDAs that use probability models specific to $\mathcal{S}^n$ on the QAP. Finally, we show that Hamming KMM EDA is better than other classical EDAs on the QAP. Specifically, conducted experiments show that the proposed approach is better than other EDAs in the literature in terms of lower Average Relative Deviation Percentage (ARDP). Moreover, the use of both Kernels and Hamming seems to be necessary for the best possible performance.

The rest of the chapter is organized as follows: in the following section, we briefly explain the QAP and the adequacy of the Hamming distance for this problem. Next, in Section 3.3, we introduce the kernels of Mallows Models over the Hamming distance. Then, in Section 3.4, we detail the proposed algorithm. Afterwards, in Section 3.5 we present the experimentation, and Section 3.6 concludes the article.

3.2 The Quadratic Assignment Problem and the Hamming distance

The Quadratic Assignment Problem (QAP) is the problem of optimally allocating n facilities at n locations in order to minimize a cost function related to the flow and the distance between every pair of facilities and pair of locations. In the QAP, an instance is defined by two matrices $D, H \in \underset{n\times n}{\mathcal{M}}(\mathbb{R}^+)$, where $D_{i,j}$ is the distance between locations i and j, and $H_{l,k}$ is the flow between facilities l and k. The search space of the QAP is the set of every permutation σ of size n, $\mathcal{S}^n$. In this sense, in the QAP, the aim is to find the permutation $\sigma \in \mathcal{S}^n$ that describes the optimal assignment of facilities into locations, where $\sigma(i) = j$ denotes that the j^{th} facility is assigned to the i^{th} location.

When Koopmans and Beckmann [1955] introduced the QAP, they presented it as a maximization problem. Given two cost matrices $D, H \in \underset{n\times n}{\mathcal{M}}$ and a profit matrix $A \in \underset{n\times n}{\mathcal{M}}$, the QAP was formulated as shown in Equation (3.1).

$$\max_{\sigma \in \mathcal{S}^n} \sum_{i=1}^{n} A_{\sigma(i),\sigma(j)} - \sum_{i=1}^{n} \sum_{j=1}^{n} D_{i,j} H_{\sigma(i),\sigma(j)} \tag{3.1}$$

Later on, the lineal term $\sum_{i=1}^{n} A_{\sigma(i),\sigma(j)}$ was dropped, since the complexity of the problem lies within the quadratic term $\sum_{j=1}^{n} D_{i,j} H_{\sigma(i),\sigma(j)}$, [Loiola et al., 2007].

$$\min_{\sigma \in \mathcal{S}^n} \sum_{i=1}^{n} \sum_{j=1}^{n} D_{i,j} H_{\sigma(i),\sigma(j)} \tag{3.2}$$

3.2.1 Distance-metrics

The MM relies on the definition of the distance for permutations, and three have been primarily considered in the framework of EDAs: Cayley, Kendall's-τ and Ulam [Ceberio, 2014, Ceberio et al., 2015c, 2014b]. The Cayley distance measures the minimum number of swaps needed to transform a permutation into another one. The Kendall's-τ distance measures the number of differently arranged pairs of items between two permutations. Finally, the Ulam distance between permutations σ and π is equal to the size of the permutations, n, minus the length of the longest increasing subsequence in $\sigma\pi^{-1}$. In addition to the previous distance-metrics, the Hamming metric also been reported for the case of permutations. The Hamming distance between two permutations, σ and π, counts the number of point-wise disagreements they have.

As mentioned in the introduction, the distance employed in the MM critically conditions the performance of the EDA when solving a given problem. Previous work on this topic [Ceberio et al., 2015c, Jones and Forrest, 1995] demonstrated that it is crucial to choose operators (distances, neighborhoods, mutations,...) that better fit the characteristics of the problem. We carried out an experiment in order to analyze the correlation between the distance at which two permutations are and the number of *components* that differ in both permutations, where component refers to each additive term $D_{i,j} H_{\sigma(i),\sigma(j)}$ for $i, j \in [n]$ in Equation (3.2). Intuitively, it is preferable when a distance-metric structures solutions in the way that close solutions differ in few components and far away solutions differ greatly in number of different components. The experiment consists of the following: For each of the considered four metrics, we choose two permutations at distance k from each other. Then, we measured the maximum number of different components they can have on a problem of size $n = 20$, independently of the instance. Finally, we repeated this process several times with different permutations, at the same distance k, in order to obtain the median and the interquartile range of the values. The results are depicted in Figure 3.1.

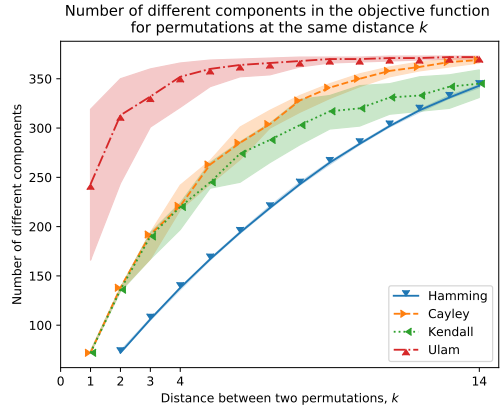


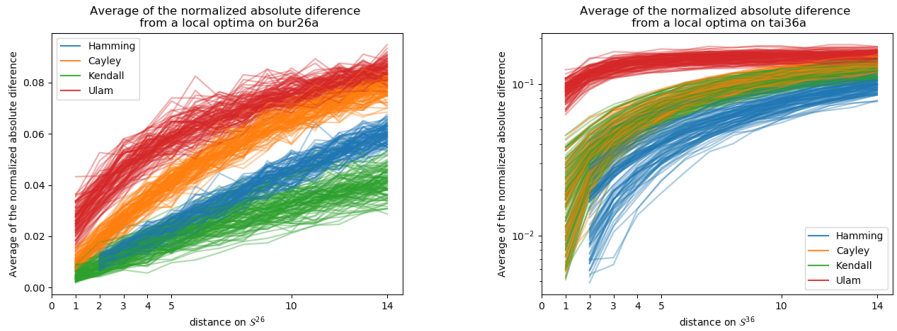
Fig. 3.1: The median, 25% and 75% percentiles of the maximum number of different components that two solutions at a given distance can have in an instance of size $n = 20$. The Hamming distance has the lowest number of different components among the four distance-metrics studied. In addition, the Hamming distance has the most consistent (almost constant) number of different components at a given distance, followed by the Cayley and Kendall distances. The Ulam distance is the worst distance in terms of number of different components.

As can be observed, the Hamming distance-metric shows the best results among the considered distance-metrics. On the one hand, it presents the least number of different components at each distance k. On the other hand, the number of different components is the same for all the permutations at distance k (unlike the rest of the metrics).

The experiment above demonstrated that, taking the definition of the QAP into account, Hamming is the best option. However, considering specific instances of the problem, for many different reasons, the previous conclusion might not hold. For instance, some components could be identical, producing no change; or the change of some components could be compensated by others. For that reason, in a new experiment, we will analyze the objective function transition for each of the metrics on specific instances of the problem. To that end, starting from a random permutation, we run a local search algorithm[1] to find a local optimum. Then, for each of the four distance-metrics, (Hamming, Cayley, Kendall's-τ and Ulam), the average normalized difference in the objective value with respect to the local optimum is computed for $\forall k \in [14]$. Specifically, defining σ_0 as the local optimum, for each of the metrics, we approximate the difference $\psi_k^{-1} \sum_{\sigma \in \mathcal{S}^n \mid d(\sigma_0,\sigma)=k} \frac{\text{abs}(f(\sigma_0)-f(\sigma))}{f(\sigma_0)}$

[1] We used the best-first local search procedure, based on the exchange neighborhood.

with the Monte Carlo sampling method using 50 repetitions, where ψ_k is the number of permutations at distance k. In addition to the average, the variance is also computed. The results are shown[1] in Figure 3.2. The instance *bur26a* (a) has special properties on the distance matrix D, which we believe makes the Kendall distance have a smoother objective value transition. The instance *tai36a* (b), on the other hand, does not have these properties, and thus, the Hamming distance produces a smoother objective value transition in this case.



(a) bur26a (b) tai36a

Fig. 3.2: The results of the objective function value transition experiment for two of the instances studied.

We observed that Hamming shows a smoother objective value transition than Cayley and Ulam, and, most of the times, than Kendall's-τ (in 4 out of 6 instances) also. Considering the results of these two experiments, shown in Figure 3.1 and Figure 3.2, it seems that the Hamming distance is the best choice among the studied metrics for the QAP.

3.3 Distance-Based Probability Models

Probability models for permutations assign a probability value to each of the permutations of n items. For the sake of applicability and computational efficiency, these probability models are defined by a restricted number of parameters. The

[1] For this experiment, 6 instances from the QAPLIB are considered. Figure 3.2 shows the results obtained for two of them. The full results of the experimentation as well as the source code of the proposed approach are available for the interested reader at https://github.com/EtorArza/SupplementaryKMMHamming

Mallows Model (MM) [Mallows, 1957] is an exponential probabilistic model defined over $\mathcal{S}^n$. The MM is described by two parameters: the concentration parameter $\xi \in \mathbb{R}^+$, and the location parameter, $\sigma_0 \in \mathcal{S}^n$ [Irurozki et al., 2019a]. The location parameter, also known as the central permutation, is the mode of the distribution. For the rest of the permutations, their probability decreases exponentially with respect to their distance from the central permutation. The speed of this exponential decay is controlled by ξ. For instance, when $\xi = 0$, the distribution is equivalent to the uniform distribution over $\mathcal{S}^n$. Contrarily, when $\xi \to \infty$, $p(\sigma_0) = 1$. Formally, the probability mass function is given as follows:

$$p(\sigma) = p(\sigma|\sigma_0, \xi) = \frac{e^{-\xi d(\sigma,\sigma_0)}}{\psi(\xi)} \tag{3.3}$$

where $d(\cdot,\cdot)$ is a distance-metric on $\mathcal{S}^n$ and $\psi(\xi)$ stands for the normalization constant.

3.3.1 Factorization and sampling under the Hamming distance

In the following, we extend the presentation of the MM for the case of the Hamming distance describing the factorization of the probability distribution induced by the model and the procedure to sample the solutions [Irurozki et al., 2019a]. Under this factorization, a simple sampling procedure for the Hamming MM can be defined. In addition, this decomposition allows a better understanding of the dynamics of the proposed EDA. Defining $\mathcal{K} \equiv d(\sigma_0, \sigma)$ as the Hamming distance from the consensus to σ, we can think of $\mathcal{K}$ and σ as random variables defined in $\{0\} \cup [n]$ and $\mathcal{S}^n$ respectively. From this point of view, $\mathcal{K}$ is dependent on σ, or in other words, given σ, $\mathcal{K}$ is known. Considering this, we can decompose $p(\sigma)$ as:

$$p(\sigma) = p(\sigma|\mathcal{K})\ p(\mathcal{K}) \tag{3.4}$$

where the first term of the factorization, $p(\sigma|\mathcal{K})$, denotes the probability of σ given the distance at which it is from the consensus, and the second term, $p(\mathcal{K})$, defines the probability of $k = d(\sigma_0, \sigma)$. The conditional probability distribution of the first term of the factorization shown in Equation (3.4), $p(\sigma|\mathcal{K})$, follows a uniform distribution. This is easy to see, since the MM gives the same probability to all permutations that are at the same distance k from the consensus. Conveniently, in $\mathcal{S}^n$, the number of permutations at Hamming distance k from a given permutation, $S(n, k)$, can be easily computed. This sequence is closely related to the number of derangements of size k. A derangement is a permutation, σ, where every item σ_i is different from its corresponding index i, hence, $\sigma = (\sigma_1, ..., \sigma_k)$ is a derangement of size k iff $\sigma_i \neq i$, $\forall i \in [k]$. For example, the permutation, $\tau = (\tau^1, \tau^2, \tau^3) = (3, 2, 1)$

is not a derangement, because $\tau^2 = 2$, while $\gamma = (\gamma^1, \gamma^2, \gamma^3) = (3, 1, 2)$ is a derangement, because $\gamma^i \neq i \ , \ \forall i \in \{1, 2, 3\}$.

In order to compute $S(n, k)$, the following formula can be used:

$$S(n, k) = \binom{n}{k} D(k) \tag{3.5}$$

where $D(k)$ is the number of derangements of size k [Irurozki et al., 2016], which is a known sequence [Sloane]. Specifically, the number of derangements $D(k)$ can be recursively computed in $\mathcal{O}(k)$ as follows:

$$D(k) = \begin{cases} 1 & k = 0 \\ 0 & k = 1 \\ (k-1)(D(k-1) + D(k-2)) & k > 1 \end{cases}$$

Since we are interested in the first $n+1$ elements of the sequence, we must compute $D(k)$ for $0 \leq k \leq n$, and that requires $\mathcal{O}(n)$ time. Therefore, it is easy to compute the conditional probability $p(\sigma|\mathcal{K} = k) = S(n, k)^{-1}$ for any σ at distance k from the consensus.

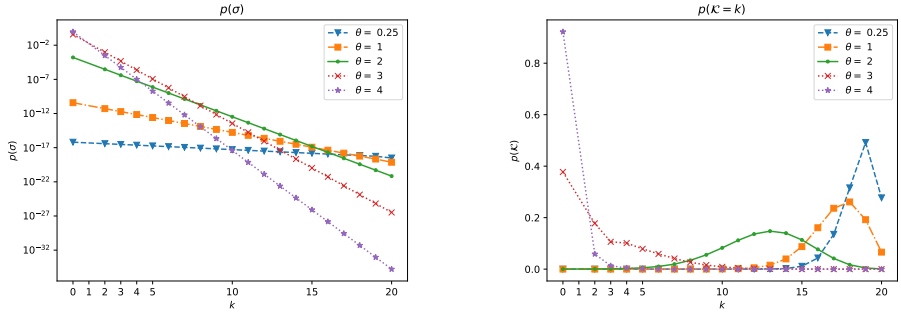
Now, we compute $p(\mathcal{K})$, the second term of the decomposition of $p(\sigma)$ in Equation (3.4). Considering the definition of $p(\mathcal{K} = k) \equiv p(d(\sigma_0, \sigma) = k)$ we obtain:

$$p(\mathcal{K} = k) = \sum_{\sigma | d(\sigma_0, \sigma) = k} p(\sigma) = S(n, k) p(\sigma) = S(n, k) \frac{e^{-\xi k}}{\psi(\xi)} \tag{3.6}$$

The previous equation can be computed in $\mathcal{O}(n)$ time for $k \in \{0\} \cup [n]$, allowing a simple two-step sampling procedure for the Hamming MM to be defined [Irurozki et al., 2019a]. First, considering the probabilities $p(\mathcal{K})$, randomly choose k, the distance at which to sample. Secondly, choose a permutation σ at distance k from the consensus σ_0 uniformly at random. A detailed explanation of the sampling procedure is shown later, in Section 3.4, in Algorithm 3.

The concentration parameter ξ controls where the probability of the permutations is concentrated. For a high value of ξ, the probability mass is concentrated near the consensus. Similarly, for a low value of ξ, the probability mass is concentrated far away from the consensus. This is possible because a low ξ defines an almost uniform distribution on $\mathcal{S}^n$, and the number of permutations at Hamming distance k from the consensus increases exponentially with k. Figure 3.3a shows $p(\sigma)$ described in Equation (3.3). For high values of ξ, $p(\mathcal{K})$ has a higher probability in lower values on k, and vice versa. In Figure 3.3b, we see that by using different values of ξ, the probability mass of $p(\mathcal{K})$ is concentrated at different distances. This is related

with $\mathbb{E}[\mathcal{K}] = \sum_{k \in \{0\} \cup [n]} k \; p(\mathcal{K} = k)$, the expectation of the distance. Given n, the instance size, there exists a bijection that maps the expected distance $\mathbb{E}[\mathcal{K}]$ to the corresponding value of the concentration parameter ξ. When $\mathbb{E}[\mathcal{K}]$ is low, the probability of the solutions near the consensus is high, and, consequently, the concentration parameter of the distribution, ξ, is high. As we will later see in Section 3.4.2, by adjusting $\mathbb{E}[\mathcal{K}]$ (and, consequently, its corresponding ξ) a simple exploration-exploitation scheme can be defined.



(a) $p(\sigma)$ (b) $p(\mathcal{K})$

Fig. 3.3: The representations of $p(\sigma)$ and $p(\mathcal{K})$ on $\mathcal{S}^{20}$ for a Hamming-based MM, considering permutations at Hamming distance $k \in \{0\} \cup [20]$ from the consensus. An instance of size $n = 20$ and different ξ values are considered.

3.3.2 Extending the Mallows Model

The Mallows Model is a unimodal distribution, and as such, it may be too rigid for multimodal problems, limiting the performance of the EDA in certain situations. As a more flexible alternative, the Generalized Mallows Model was proposed [Fligner and Verducci, 1986a] for the Hamming distance [Irurozki et al., 2019a]. Based on the decomposition property of some distances, the GMM has a unique central parameter, just as the MM, but it also has several concentration parameters, giving the model a higher flexibility.

Introduced in [Murphy and Martin, 2003], a multimodal alternative to the MM is the Mixtures of Mallows Model (MMM). In this case, the population is considered to be composed of m differently sized clusters. Given the central and concentration parameters for each cluster, σ_i and ξ_i, the probability mass distribution is expressed as

$$P(\sigma|\boldsymbol{\sigma},\boldsymbol{\xi}) = \sum_{i=1}^{m} w_i \frac{e^{-\xi_i d(\sigma,\sigma_i)}}{\psi(\xi_i)}$$

where $\sum_{i=1}^{m} w_i = 1$, $\boldsymbol{w} = \{w_1, ..., w_m\}$ and $w_i > 0$. Usually, $\boldsymbol{\sigma}$, $\boldsymbol{\xi}$ and $\boldsymbol{w}$ are estimated using the Expectation Maximization algorithm [Awasthi et al., 2014]. An extension of the MMM that considers several concentration parameters per central permutation is the Mixtures of Generalized Mallows Model (MGMM) [Ceberio et al., 2015e].

Taking the idea of mixture models to the limit, and by considering each solution in the set as a cluster of equal weight, another even more flexible model can be defined: Kernels of Mallows Model (KMM). Given a set of m permutations, the KMM is the averaging of m MMs centered on these permutations. Therefore, we say that KMM is a combination of several MM. Contrary to the GMM, the KMM is an MM with the same concentration parameter but different central permutations. This model breaks the strong unimodality assumption of the MM and the GMM. Given the set of central permutations $\boldsymbol{\sigma} = \{\sigma_1, \sigma_2, ..., \sigma_m\}$, the mass probability distribution of KMM can be defined as follows:

$$P(\sigma|\boldsymbol{\sigma},\xi) = \frac{1}{m} \sum_{i=1}^{m} \frac{e^{-\xi d(\sigma,\sigma_i)}}{\psi(\xi)}$$

where $\psi(\xi)$ is the normalization constant.

3.4 Hamming Kernels of Mallows Model EDA

In this chapter, we approach the QAP with an MM-based EDA. Specifically, Kernels of Mallows Model under the Hamming distance are used in the framework of EDAs. To control the convergence of the algorithm, a simple yet effective exploration-exploitation scheme is presented, based on ξ, the concentration parameter.

3.4.1 Learning and sampling

The learning of a model in an EDA usually refers to obtaining the maximum likelihood estimators for the parameters of the selected probabilistic model. Since the proposed EDA is based on Kernels of Mallows Model, a non-parametric model, we do not need to estimate the central permutations. Instead, the selected set of permutations is used as the set of central permutations $\boldsymbol{\sigma} = D_{t-1}^{sel}$. In addition,

the concentration parameter ξ is set by a simple exploration-exploitation, as we extensively explain in the next section.

Once the model is defined, we need to know how to sample solutions from it. In this case, the sampling procedure is based on the distances sampling algorithm [Irurozki et al., 2016], as shown in Algorithm 3. It is a three-step procedure. First, select a central permutation σ_0 from the selected set of permutations $\boldsymbol{\sigma}$ uniformly at random (line 1). Then, based on the probabilities obtained from Equation (3.6), choose a distance k at which to sample (lines 2 and 3). Finally, a permutation at Hamming distance k from σ_0 is chosen uniformly at random (line 4).

Algorithm 3: Sampling algorithm of Hamming KMM

Input:
$\boldsymbol{\sigma} = \{\sigma_1, ..., \sigma_m\}$: The set of central permutations.
ξ: The concentration parameter.
Output:
σ: The sampled permutation.

1 $\sigma_0 \leftarrow$ choose uniformly at random from $\boldsymbol{\sigma}$
2 compute $\left\{p(k) \propto S(n,k)e^{-\xi k} \mid k \in [n] \setminus \{1\}\right\}$
3 $k \leftarrow$ randomly choose k with probability proportional to $p(k)$ (0 is never chosen to avoid sampling the consensus.)
4 $\sigma \leftarrow$ choose k items from σ_0 and derange them (shuffle them uniformly at random, but making sure none of these k items remains in its original place)
5 **return** σ

3.4.2 Exploration-Exploitation scheme: updating ξ

The convergence of the EDA is controlled by a simple exploration-exploitation scheme. The trade-off is balanced by the expectation of the distance, $\mathbb{E}[\mathcal{K}]$, which is transformed into its equivalent ξ at run-time. The advantages of using $\mathbb{E}[\mathcal{K}]$ instead of ξ are threefold. First, we believe $\mathbb{E}[\mathcal{K}]$ is more intuitive than ξ, since its interpretation is much easier. In addition, by using $\mathbb{E}[\mathcal{K}]$, it is easier to take into account the instance size n when increasing $\mathbb{E}[\mathcal{K}]$. Finally, $\mathbb{E}[\mathcal{K}]$ is more correlated with the transition of the objective function value than ξ. Figure 3.4 shows the evolution of the expected difference in the objective function value and $\mathbb{E}[\mathcal{K}]$ with respect to the concentration parameter ξ for an instance of size $n = 125$ (*tai125e01* from the Taixxeyy instances set [Drezner et al., 2005]). It can be seen that the shape of $\mathbb{E}[\mathcal{K}]$ resembles the normalized expected difference of the objective function. In fact, Figure 3.5 shows that the relationship between $\mathbb{E}[\mathcal{K}]$ and

the normalized objective function difference is almost lineal. This means that the transition of the objective value can be more accurately controlled by $\mathbb{E}[\mathcal{K}]$.

The starting and final values of $\mathbb{E}[\mathcal{K}]_t$[1], $\mathbb{E}[\mathcal{K}]_0$ and $\mathbb{E}[\mathcal{K}]_{tmax}$, respectively, are set before the algorithm is executed. In this sense, $\mathbb{E}[\mathcal{K}]_0$ is set to a high value. A high value of $\mathbb{E}[\mathcal{K}]_t$ favors exploration, because the sampled solutions are expected to be far away from the selected solutions. Therefore, the sampled solutions are going to be very different from the selected solutions, forcing them to visit different and unobserved areas of the solution space. At each iteration, the expectation of the distance $\mathbb{E}[\mathcal{K}]$ is decreased ($\mathbb{E}[\mathcal{K}]_{t+1} < \mathbb{E}[\mathcal{K}]_t$). As the number of iterations increases, the algorithm shifts from an exploration state to an exploitation state. In this exploitation stage, the new solutions will be similar (they will be near each other in the Hamming distance sense) to the known solutions.

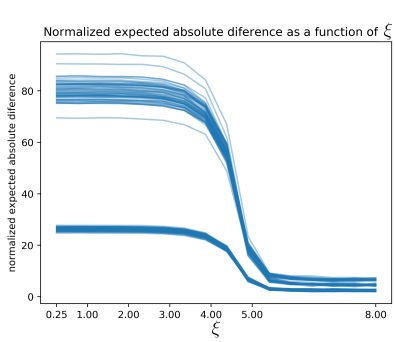
An idea to update $\mathbb{E}[\mathcal{K}]_t$ would be to decrease it at a constant rate. However, we found that decreasing $\mathbb{E}[\mathcal{K}]_t$ at an exponential rate produces better results, as we will later discuss in Section 3.5. The stopping criteria for the algorithm is given in terms of the maximum number of iterations, and the number of solutions evaluated in each iteration is $P_s/2$, where P_s denotes the population size of the EDA. Therefore, at each iteration t, the progress of the algorithm $p \in (0, 1)$ is defined as $p = t/t_{max}$. Then, given the intensity parameter $\gamma \in \mathbb{R}^+$, this progress is transformed into an exponential progress with the function $\delta(p) = \frac{\exp(-\gamma p)-1}{\exp(-\gamma)-1}$. Finally, the expectation at iteration t, $\mathbb{E}[\mathcal{K}]_t$, is set to $\mathbb{E}[\mathcal{K}]_t = \mathbb{E}[\mathcal{K}]_{tmax} + \delta(p)(\mathbb{E}[\mathcal{K}]_0 - \mathbb{E}[\mathcal{K}]_{tmax})$. Figure 3.6 shows $\delta(p)$ for the estimated optimal value of the parameter $\gamma = 5.14$.

3.4.3 Computational complexity and scalability

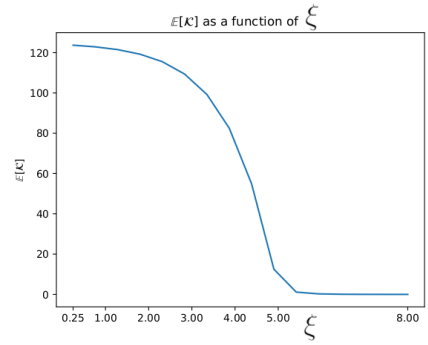
If n is the instance size and m the considered population size, the time complexity of the sampling stage is $\mathcal{O}(mn)$ [Irurozki et al., 2016]. The total cost of the algorithm, without considering the objective function evaluations, is $\mathcal{O}(mn) + \mathcal{O}(m \log(m))$. Considering a constant population size m, the cost of Hamming KMM EDA is dominated by the cost of the evaluations, which is $\mathcal{O}(mn^2)$. The memory complexity of Hamming KMM EDA is $\mathcal{O}(mn)$.

Even though the theoretical computational complexity of Hamming KMM EDA is reasonable (since it is dominated by the cost of the evaluations, $\mathcal{O}(m \cdot n^2)$), a detailed empirical analysis is recommended to study the scalability of the algorithm. To this end, we conducted two experiments on the runtime of the proposed algorithm. In short, we found out that the computational cost of the KMM is low when compared to the cost of evaluating the solutions. What is more, by using a efficient sampling method, the time complexity is in practice lower than $\mathcal{O}(m \cdot n^2)$.

[1] $\mathbb{E}[\mathcal{K}]_t$ denotes the expectation of the distance $\mathbb{E}[\mathcal{K}]$ at iteration t.



(a) Normalized expected absolute difference between local optima and solutions sampled using the correspondent ξ.



(b) The expectation of d, $\mathbb{E}[\mathcal{K}]$ with respect to ξ.

Fig. 3.4: Figure 3.4a shows the normalized expected difference of the objective function value. This difference is measured between 100 random local optima and solutions sampled using an MM centered on these local optima and concentration parameter ξ. Specifically, for each of the considered local optima σ_0, Figure 3.4a shows $\lim_{s\to\infty} s^{-1} \sum_{i=1}^{s} \frac{\text{abs}(f(\sigma_0)-f(\sigma_i))}{f(\sigma_0)}$ where σ_i is obtained by sampling from an MM centered on σ_0 and using the concentration parameter ξ for each $i \in [s]$. The instance tai125e01 was used to obtain this figure. Figure 3.4b shows the expectation of d, $\mathbb{E}[\mathcal{K}]$ as a function of ξ. Observe how the shape of $\mathbb{E}[\mathcal{K}]$ resembles the normalized expected difference of the objective function value.

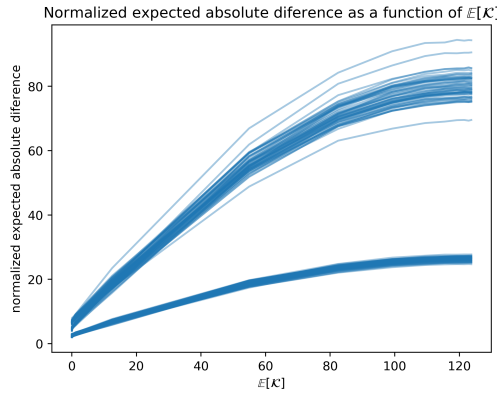


Fig. 3.5: The normalized expected difference of the objective function value as a function of $\mathbb{E}[\mathcal{K}]$.

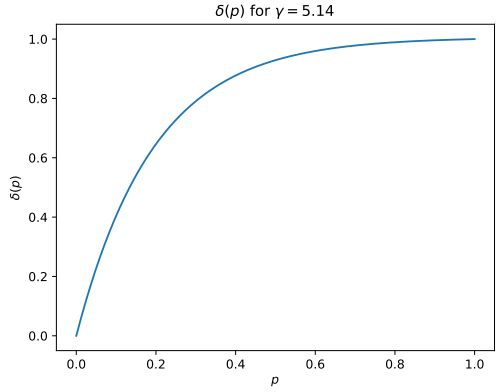


Fig. 3.6: The progress after the exponential function δ is applied.

A. Experiment 1

In the first experiment, we computed the percentage of time spent evaluating the objective function when using Hamming KMM to optimize *Taixxa* instances of size n, with n ranging from 10 to 100. The stopping criterion was set to $1000 \cdot n^2$ evaluations, and the rest of the parameters remain equal to those considered in the experimental section. Results are depicted in Figure 3.7. As can be observed, as the instance size increases, the percentage of time spent evaluating the objective function also increases, up to 80% in instances of size 100. This means that the runtime of Hamming KMM is indeed dominated by the evaluation of the objective function, which is a positive feature, since it means that the runtime overhead of the EDA is small in comparison to the cost of evaluating the solutions.

B. Experiment 2

In this second experiment, we empirically show that the cost of the Hamming KMM EDA algorithm is at most $\mathcal{O}(n^2)$ for a fixed population size of $m = 972$, where n is the problem instance size. With this aim in mind, we run Hamming KMM in instances obtained by cutting *Tai256c*, the largest instance available in the QAPLIB [Burkard et al., 1997]. In this experiment, the stopping criterion was set to 10^6 evaluations, and the rest of the parameters remain equal to those considered in the experimental section. In Figure 3.8, the proportional time spent on each function evaluation divided by n^2 is shown. Specifically, $\frac{runtime}{E \cdot n^2}$ is shown, where E is the number of evaluations used as stopping criterion ($E = 10^6$), and *runtime* is the time it takes to optimize an instance of size n. Results point out that

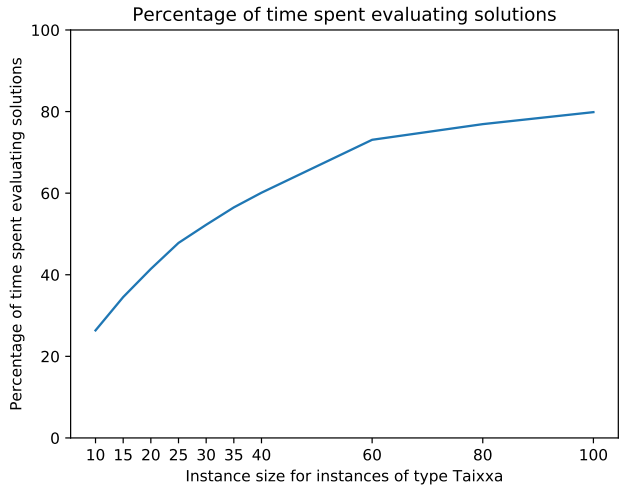


Fig. 3.7: Percentage of time spent on evaluating solutions when optimizing *Taixxa* instances of size n, with $1000n^2$ as stopping criterion.

$\frac{runtime}{E \cdot n^2}$ decreases as n increases. This means that for a fixed stopping criterion in terms of maximum evaluations and a fixed population size, the actual time complexity of Hamming KMM is at most $\mathcal{O}(n^2)$. In the following, we show how this cost can be reduced even further.

Even though the cost of evaluating a candidate solution is $\mathcal{O}(n^2)$, given two different permutations $\sigma_a, \sigma_b \in \mathcal{S}^n$, if $\sigma_b \in \mathcal{N}(\sigma_a)$ and the objective function value of σ_a is known, then the objective function value of σ_b can be updated in $\mathcal{O}(n)$ time [Pardalos et al., 1994]. The proposition below defines the objective function relationship that two solutions at Hamming distance two have.

Property 9 *Suppose* $\exists i_1, i_2 \in [n] | \sigma_a(i) = \sigma_b(i) \;\; \forall i \in [n] \setminus \{i_1, i_2\}$. *Then,*

$$
\begin{aligned}
f(\sigma_b) \;=\; f(\sigma_a) + & \sum_{k \in \{i_1, i_2\}} \sum_{i=0}^{n-1} \left(D_{i,k} H_{\sigma_b(i),\sigma_b(k)} + D_{k,i} H_{\sigma_b(k),\sigma_b(i)} \right) - \\
& \sum_{k \in \{i_1, i_2\}} \sum_{i=0}^{n-1} \left(D_{i,k} H_{\sigma_a(i),\sigma_a(k)} + D_{k,i} H_{\sigma_a(k),\sigma_a(i)} \right) + \\
& \sum_{k \in \{i_1, i_2\}} D_{k,k} H_{\sigma_a(k),\sigma_a(k)} - D_{k,k} H_{\sigma_b(k),\sigma_b(k)}
\end{aligned}
$$

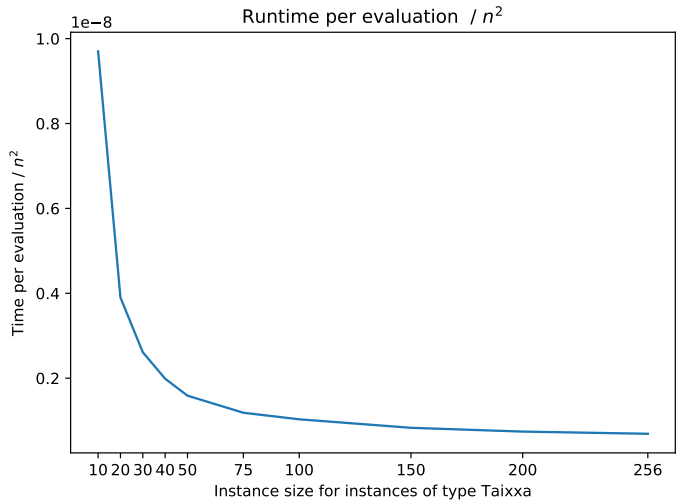


Fig. 3.8: Runtime per evaluation divided by n^2 of Hamming KMM when optimizing *Taixxc* instances of size n, with 10^6 evaluations as stopping criterion.

This process can be repeated over and over again to compute the objective function value of permutations at Hamming distance two or more, and if the permutations are close enough, it is more efficient than directly computing $f(\sigma_b)$. It is worth noting that the proposed approach is based on MM kernels and we used the Distances Sampling Algorithm as the sampling procedure [Irurozki et al., 2016]. Hence, we sample at a given distance of a known permutation. Therefore, this efficient method to compute the objective function yields a considerable speedup in the EDA, especially in the last iterations of the EDA, where the expected distance from the central permutation to the sampled permutation $\mathbb{E}[\mathcal{K}]$ is small.

3.5 Experimental study

In order to prove the validity of the proposed method, in this section, we present an exhaustive analysis of the performance of the algorithm.

3.5.1 General remarks

Some popular instances of the QAP have been employed to evaluate the performance of the proposed approach (Hamming KMM EDA). In particular, 30 instances from the QAPLIB [Burkard et al., 1997] were used.

Before running the experiments, there are a number of parameters that need to be set in the proposed approach. First, we have the starting $\mathbb{E}[\mathcal{K}]_0$ and final $\mathbb{E}[\mathcal{K}]_{max}$ values of $\mathbb{E}[\mathcal{K}]$. $\mathbb{E}[\mathcal{K}]_0$ is set to $n/2$, where n is the instance size, and $\mathbb{E}[\mathcal{K}]_{max}$ is set to 0.25. Setting $\mathbb{E}[\mathcal{K}]_0$ to $n/2$ produces a distribution in which, on average, the sampled permutations have half the items in the same position as the reference permutation σ_0. The chosen $\mathbb{E}[\mathcal{K}]_{max}$ value produces a similar distribution that $\mathbb{E}[\mathcal{K}]_{max} \to 0$ would, but without numerical errors, it is thus the most exploitative state possible. The other two parameters are P_s and γ. The parameter P_s is the population size of the EDA, and γ measures the speed at which $\mathbb{E}[\mathcal{K}]$ is decreased. These two parameters are set using Bayesian optimization [Pedregosa et al., 2011] with the instance *tai31a* (which is not among the benchmark instances considered). The optimal values found for these parameters are 972 and 5.14 respectively. These parameters are used in all the executions of Hamming KMM EDA.

All the algorithms considered in the experimentation are tested in the set of 30 instances. The stopping criterion is the same for all the considered algorithms and instances: $1000n^2$ evaluations. The experimentation was conducted on a single machine with an octa-core AMD Ryzen 7 1800X Processor, with 8Gb of RAM. Hamming KMM was implemented in C++. The rest of the algorithms were implemented in either Java or C++. In any case, since the number of evaluations is used as the stopping criteria, it is not affected by the hardware nor the programming language, and therefore, is easy to reproduce. As a reference, it takes Hamming KMM about 107 seconds to perform $1000n^2$ evaluations in the largest of the studied instances (tai100a) and 0.1 seconds in the smallest one (tai10a).

For each benchmark instance, the results are recorded as the Average Relative Deviation Percentage, ARDP $= |\frac{f_{best} - f_{av}}{f_{best}}|$, where f_{best} is the best known value and f_{av} is the average of the best objective values obtained in each repetition. For further statistical analysis, Bayesian Performance Analysis [Calvo et al., 2019b, Calvo and Santafé Rodrigo, 2016b] (BPA) is carried out to study the uncertainty of the results of each experiment[1] Specifically, Placket-Luce is used as the probability model, defined in $\mathcal{S}^n$, in this case, corresponding to the rankings of the algorithms. BPA considers probability distributions over probability distributions. In our case, assuming a uniform prior distribution, the posterior distribution of the probability

[1] In the file "comparison_between_BPA_and_NHST.pdf", available in the GitHub repository, we justify the use of Bayesian performance analysis instead of other more traditional hypothesis tests.

of each algorithm being the best performing one (winning) is computed. The goal of this analysis is to determine which algorithm performs the best for the set of considered benchmark instances. The inference with the BPA approach is fairly simple [Calvo et al., 2019b]. First the scores of the algorithms are transformed from ARDP to their corresponding rankings on each of the test instances. Then, a sample is produced from the posterior probability distribution of the weights of the Plackett-Luce model (a probability model for rankings). And finally, based on these sampled weights, the credible interval of 90% is computed for each algorithm. This interval means that there is a 90% chance that the actual probability of the algorithm being the highest ranking algorithm (being the winner) lies within the interval.

3.5.2 Experiment 1: Kernels and the exponential $\mathbb{E}[\mathcal{K}]$

For the sake of measuring the contribution of each of the two main parts that extend a Hamming Mallows Model EDA, (i) the use of kernels and (ii) the use of an exponential increase of $\mathbb{E}[\mathcal{K}]$, we compare the performance of the full model with the simplified variants. The simplified models considered are: KMM with linear increase of $\mathbb{E}[\mathcal{K}]$, MM with exponential increase of $\mathbb{E}[\mathcal{K}]$, both with one missing part; MM with linear increase of $\mathbb{E}[\mathcal{K}]$, missing both parts; and finally, a simple Hamming MM in which the concentration parameter ξ is estimated at each iteration. Figure 3.9 shows all the studied simplified models, ordered by their complexity in terms of the number of free parameters. The average ARDP obtained in all the instances for each of the models is also shown in this figure.

The same parameters are considered for all the EDAs, thus, the parameters estimated with Bayesian optimization for the full model are used. The ARDP values are recorded in Table 3.1.

The full model outperforms the rest of the models in 60% of the instances, while the second best model, (KMM with linear increase), only outperforms the rest of the models in 23.3% of the studied instances. It is worth noting that using the kernels part is much more important than the exponential increase of $\mathbb{E}[\mathcal{K}]$, since, as seen in Figure 3.9, both kernel models have an average ARDP lower than 1%, while the rest of the models have an ARDP over 10%. Additionally, the exponential increase is detrimental for the MM, and it is only a positive addition when we consider it alongside the kernels parts. **This experiment indicates that both the kernels and the exponential increase of $\mathbb{E}[\mathcal{K}]$ are key parts of the proposed model.**

Figure 3.10 shows the credible interval of 90% calculated from the samples obtained from the posterior distribution of the probability of each of the algorithms being the best. As can be observed, both kernel-based algorithms (with exponen-

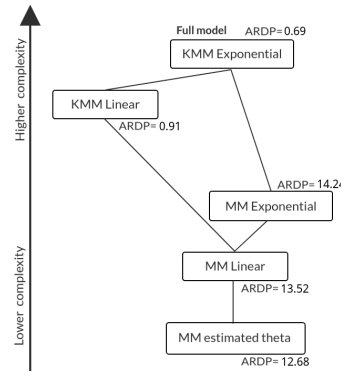


Fig. 3.9: A diagram of the simplified models considered in this chapter. The vertical axis is proportional the complexity level of the model in terms of the number of free parameters. The average ARDP obtained in the studied instances is shown for each model.

tial increase and linear increase) have a higher expected probability of being the best than the rest, around 0.6 and 0.4 respectively.

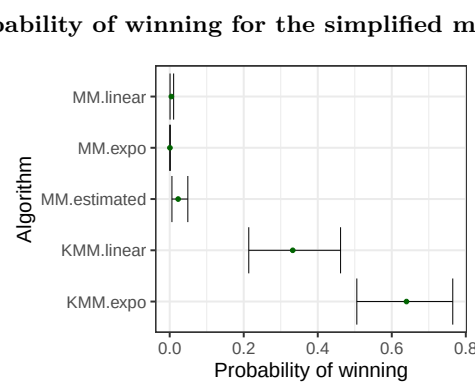


Fig. 3.10: Credible intervals of 90% and expected value of the estimated posterior probability of each algorithm being the winner among those tested.

Table 3.1: The results of the simplified models when compared with the full model. The best performing algorithm is highlighted in bold.

Instance	KMM Exponential	KMM Linear	MM Exponential	MM Linear	MM estimated ξ
bur26a	0.105	**0.100**	1.600	1.469	1.420
bur26b	0.182	**0.165**	1.596	1.534	1.335
bur26c	**0.007**	0.010	2.202	1.970	1.717
bur26d	**0.007**	**0.007**	2.342	2.147	1.930
nug17	0.179	**0.110**	9.919	9.371	8.655
nug18	**0.326**	0.409	10.415	10.332	9.798
nug20	**0.125**	0.175	11.008	10.911	10.381
nug21	0.271	**0.197**	14.475	13.515	12.806
tai10a	**0.000**	**0.000**	3.484	3.207	2.338
tai10b	**0.000**	**0.000**	2.814	2.653	1.524
tai12a	0.140	**0.000**	9.624	8.806	8.280
tai12b	**0.000**	**0.000**	4.924	4.974	4.635
tai15a	**0.179**	0.190	8.252	8.194	7.536
tai15b	**0.007**	**0.007**	0.910	1.020	0.822
tai20a	**0.843**	0.947	12.830	12.379	11.863
tai20b	**0.068**	0.123	9.349	9.089	9.165
tai25a	**1.265**	1.732	13.586	12.941	12.454
tai25b	**0.025**	0.041	30.511	27.680	21.286
tai30a	**1.435**	1.891	12.547	12.258	11.957
tai30b	0.189	**0.075**	34.105	29.592	23.616
tai35a	**1.485**	2.429	13.852	13.121	12.800
tai35b	**0.476**	0.526	30.253	27.398	24.357
tai40a	**1.762**	2.622	13.829	13.451	13.157
tai40b	1.068	**0.299**	37.258	34.402	33.339
tai60a	**2.237**	3.400	13.757	13.524	13.180
tai60b	**0.493**	0.647	33.766	32.900	33.237
tai80a	**2.172**	3.658	12.547	12.361	12.005
tai80b	**2.235**	2.707	33.004	32.617	32.004
tai100a	**2.190**	3.538	11.771	11.619	11.485
tai100b	**1.142**	1.404	30.624	30.156	31.466

3.5.3 Experiment 2: Comparing specific EDAs for permutation problems

In this experiment, we compare Hamming KMM EDA to other specific EDAs for permutation problems[1] considered in the literature. We compare the performance of the proposed approach with respect to other MM-based approaches. For example, the MM has already been applied to permutation problems under the Cayley, Kendall's-τ and Ulam distances [Ceberio et al., 2015c]. Cayley and Kendall-based KMMs [Ceberio et al., 2015f] and GMMs have also been studied. Additionally, *mixtures of GMMs* (MGMMs) have also been applied to permutation-based problems under the Cayley and Kendall distances [Ceberio et al., 2015e]. Specifically, in this last article, the MGMM with two clusters was found to outperform the other MGMM approaches, and therefore, we will only consider MGMMs with two clusters. In addition to MM EDAs, we compare the performance of the proposed algorithm to a Plackett-Luce EDA [Ceberio et al., 2013].

[1] Specific EDAs are those that estimate a probability distribution explicitly on $\mathcal{S}_n$.

The ARDP value for all the instances is recorded in Table 3.2. We kept the parameters proposed by each author in the paper that the algorithm is proposed.

Table 3.2: The average ARPD results of Hamming KMM EDA and other EDA approaches specific to S^n. The best performing algorithm is highlighted in bold.

	Hamming	Ulam	Kendall				Cayley				Plackett-
Instance	KMM	MM	MM	KMM	MGMM	GMM	MM	KMM	MGMM	GMM	Luce
bur26a	**0.105**	3.716	1.937	2.453	1.955	1.746	0.359	0.211	0.690	0.365	1.730
bur26b	**0.182**	4.052	2.008	2.353	1.998	1.461	0.482	0.305	0.603	0.465	1.680
bur26c	**0.007**	4.386	2.121	2.655	2.054	1.827	0.369	0.172	0.634	0.307	1.843
bur26d	**0.007**	4.756	2.352	2.872	2.233	1.882	0.370	0.168	0.695	0.355	1.890
nug17	**0.179**	9.434	9.174	12.535	9.284	7.771	4.706	2.154	6.680	3.147	7.188
nug18	**0.326**	17.601	8.881	12.746	10.306	8.316	4.689	2.948	6.161	3.684	7.850
nug20	**0.125**	11.391	9.603	11.961	8.541	7.759	5.195	1.844	2.977	3.214	10.895
nug21	**0.271**	13.934	12.252	12.613	12.137	10.738	5.722	2.695	7.859	3.380	13.659
tai10a	**0.000**	15.514	6.464	11.306	8.048	8.395	2.354	1.038	2.244	2.348	2.278
tai10b	**0.000**	25.309	7.025	15.845	7.506	6.327	1.032	1.440	1.475	2.324	2.835
tai12a	**0.140**	9.521	11.905	17.115	12.359	11.496	7.226	5.856	7.415	6.415	6.928
tai12b	**0.000**	6.043	11.716	21.591	13.287	11.565	5.046	3.551	8.264	6.414	6.757
tai15a	**0.179**	7.919	9.225	11.174	9.670	9.128	4.648	3.153	6.497	3.586	5.631
tai15b	**0.007**	0.975	1.279	1.536	1.224	0.997	0.528	0.369	0.749	0.411	0.743
tai20a	**0.843**	12.014	12.050	12.921	11.443	10.942	6.971	2.820	5.412	5.355	11.958
tai20b	**0.068**	7.494	13.348	32.965	12.735	12.788	5.112	5.322	2.345	2.646	6.912
tai25a	**1.265**	12.355	11.856	12.439	11.765	11.159	7.572	4.735	8.397	5.537	11.962
tai25b	**0.025**	16.446	24.200	56.513	30.102	22.254	6.071	5.456	10.529	4.692	20.214
tai30a	**1.435**	15.292	11.263	11.314	10.529	10.184	6.628	3.301	4.629	4.947	11.682
tai30b	**0.189**	49.972	29.100	45.465	27.863	21.636	9.282	7.843	10.025	11.077	22.984
tai35a	**1.485**	12.912	11.879	11.820	11.862	11.221	7.316	4.592	7.621	4.880	12.921
tai35b	**0.476**	20.097	24.583	36.410	29.819	21.667	7.083	5.976	9.532	5.346	25.320
tai40a	**1.762**	13.257	11.651	11.546	11.599	11.004	7.162	3.670	4.904	4.862	13.272
tai40b	**1.068**	28.524	30.422	44.129	33.423	25.576	10.729	8.421	6.970	8.703	33.436
tai60a	**2.237**	13.222	11.367	10.996	11.026	10.234	7.354	3.878	4.666	4.574	13.103
tai60b	**0.493**	34.853	33.144	40.119	33.344	24.317	7.112	5.491	5.897	5.238	32.017
tai80a	**2.172**	12.109	9.964	9.740	9.853	9.465	6.525	3.745	4.111	4.358	12.074
tai80b	**2.235**	32.741	31.416	33.977	30.511	26.626	6.674	5.295	6.053	5.792	32.458
tai100a	**2.190**	11.496	9.469	9.062	9.282	8.711	6.237	3.460	3.617	3.913	11.469
tai100b	**1.142**	31.929	25.849	31.703	29.561	22.020	5.469	4.603	4.888	4.982	31.116

Figure 3.11 shows the credible interval of 90% of the posterior distribution of the probability of being the best algorithm for the EDAs specific to $\mathcal{S}^n$. Considering credible intervals of 90%, we can say the probability of Hamming KMM being the highest ranked method is higher than 60%. From this analysis, it is clear that Hamming KMM is the algorithm with the highest probability of being the best, followed by Cayley KMM.

To sum up, we observe that KMM EDA obtains a lower ARDP for all the 30 benchmark instances considered. **Therefore, the experimentation suggests that using the proposed model is the best option among the specific EDAs on $\mathcal{S}_n$ for the 30 instances considered in this chapter.**

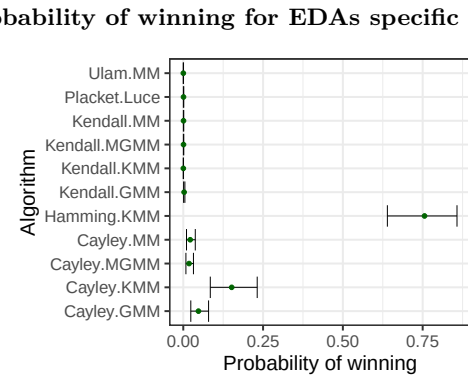


Fig. 3.11: Credible intervals of 90% and expected value of the estimated posterior probability of each algorithm being the winner among those tested.

3.5.4 Experiment 3: Classical EDAs

Finally, we compare Hamming KMM EDA to other classical EDAs for the QAP in the literature. In the review paper on EDAs in permutation problems [Ceberio et al., 2012], the performance of 13 classical EDAs was studied. Using a null-hypothesis statistical testing, the authors found that there were no statistically significant differences among the best performing six methods for the QAP. These six methods are *univariate marginal distribution algorithm* (UMDA) [Larrañaga et al., 2000], *mutual information maximization for input clustering* (MIMIC) [De Bonet et al., 1997], *estimation of Bayesian network algorithm* (EBNA) [Bengoetxea et al., 2002], *edge histogram-based sampling algorithm* (EHBSA) [Tsutsui et al., 2003] and two variants of *node histogram-based sampling algorithm* (NHBSA) [Tsutsui et al., 2006], namely NHBSA$_{WT}$ and NHBSA$_{WO}$. In this experiment, we compare Hamming KMM EDA to these other algorithms. Not limited to the previous algorithms, a recent successful EDA, the Random Key EDA [Ayodele et al., 2016, 2017], was also incorporated to the study.

The parameters proposed by the EDAs review article [Ceberio et al., 2012] are used for the methods compared in that article, and we use the parameters proposed by the authors in the RK-EDA [Ayodele et al., 2016]. The ARDP results are shown in Table 3.3.

Hamming KMM EDA obtains the best results in terms of a lower ARPD value in 70% of the considered instances. The second most competitive approach is NHBSA$_{WT}$, which outperforms the rest of the methods in 20% of the considered instances. In addition to obtaining the lowest ARPD, Hamming KMM EDA is also the most consistent algorithm. For instance, while NHBSA obtains an ARPD over

5% in one instance, the proposed approach obtains lower than 2.5% ARPD in all instances. However, for the four *bur26x* instances considered, the *node histogram-based sampling algorithm* (NHBSA$_{WT}$) is able to outperform Hamming KMM EDA. These instances have special properties in the distance matrix D. Specifically, adjacent rows and columns are similar to each other. Although Hamming KMM EDA is still the second best approach in these instances, we believe that the Hamming distance is not particularly suited for these instances, as argued in Section 3.2.1.

Table 3.3: The average ARPD results of Hamming KMM EDA and other EDA approaches. The best performing algorithm is highlighted in bold.

Instance	Hamming	UMDA	MIMIC	EBNA	EHBSA$_{wt}$	NHBSA$_{wt}$	NHBSA$_{wo}$	RK-EDA
bur26a	0.105	0.323	0.281	0.311	0.442	**0.094**	0.172	0.535
bur26b	0.182	0.327	0.306	0.387	0.304	**0.095**	0.238	0.475
bur26c	0.007	0.064	0.102	0.116	0.208	**0.000**	0.023	0.356
bur26d	0.007	0.063	0.146	0.073	0.021	**0.000**	0.029	0.213
nug17	**0.179**	2.760	2.200	2.673	1.386	0.202	1.247	2.991
nug18	**0.326**	2.979	3.114	2.663	2.073	0.332	1.917	2.684
nug20	**0.125**	3.070	3.459	2.926	2.023	0.479	1.374	2.907
nug21	0.271	2.022	2.806	1.989	3.199	**0.254**	1.214	3.868
tai10a	**0.000**	2.113	3.295	2.833	1.729	0.043	1.944	5.279
tai10b	**0.000**	0.807	2.282	0.837	**0.000**	**0.000**	0.461	6.617
tai12a	0.140	4.980	5.514	4.690	**0.000**	0.208	4.136	7.181
tai12b	**0.000**	4.100	3.706	3.125	**0.000**	0.055	1.184	10.605
tai15a	**0.179**	2.993	3.634	3.415	3.043	0.665	2.151	4.643
tai15b	0.007	0.250	0.406	0.419	0.373	**0.000**	0.163	8.724
tai20a	**0.843**	4.779	5.226	4.224	4.885	2.280	3.360	7.049
tai20b	**0.068**	3.530	4.450	3.840	1.956	0.270	4.220	4.176
tai25a	**1.265**	4.387	4.700	4.297	6.160	3.630	3.325	6.510
tai25b	**0.025**	2.740	3.462	2.728	1.366	0.099	0.824	9.949
tai30a	**1.435**	3.559	4.643	4.091	6.666	3.896	2.640	6.895
tai30b	**0.189**	6.502	10.143	6.621	1.332	0.765	10.801	16.502
tai35a	**1.485**	4.226	4.976	4.025	7.514	4.919	2.606	7.233
tai35b	**0.476**	4.087	6.355	3.453	2.744	1.162	3.723	7.972
tai40a	**1.762**	4.038	5.246	3.771	7.959	5.292	2.748	7.814
tai40b	**1.068**	5.732	8.221	5.932	4.486	2.418	5.334	9.287
tai60a	**2.237**	4.032	5.001	4.009	7.419	4.243	3.346	7.449
tai60b	**0.493**	1.188	5.309	2.558	8.177	0.836	3.379	7.347
tai80a	**2.172**	3.737	4.621	3.595	7.114	4.415	3.032	7.046
tai80b	**2.235**	4.387	5.491	5.276	12.488	2.340	3.290	8.640
tai100a	**2.190**	3.460	4.227	3.321	6.737	4.519	2.792	6.636
tai100b	**1.142**	2.025	4.672	2.353	11.416	1.214	2.469	5.602

Figure 3.12 shows the credible interval of 90% of the posterior distribution of the probability of being the best algorithm for the EDA not specific to $\mathcal{S}^n$. We can say with high confidence that the probability of Hamming KMM being the highest ranked method is above 50%. In contrast, the probability of the next best performing method, $NHBSA_{wt}$, being the best one is lower than 30%. The rest of

the methods have a fairly lower performance, with probability below 0.3 of being the highest ranking methods, considering credible intervals of 90%.

Taking into account this analysis, Hamming KMM has a higher chance of being the highest ranked method than the rest of the EDA methods tested.

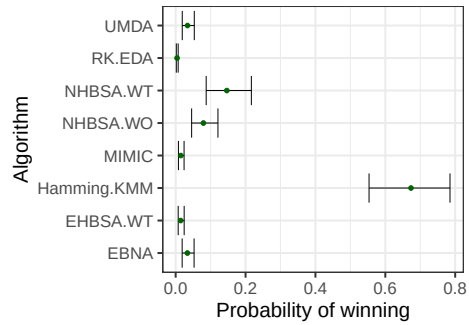


Fig. 3.12: Credible intervals of 90% and expected value of the estimated posterior probability of each algorithm being the winner among those tested.

3.6 Conclusion & future work

In this chapter, we aimed to take a step forward in the development of EDAs for permutation problems. We argued that the Hamming distance is suitable for the QAP, as it produces the smoothest objective function transitions when compared to other distance-metrics. Moreover, unlike for the rest of the distance, we saw that the number of components changed on the objective function of the QAP was constant for the Hamming distance. These two properties suggest that the Hamming distance is suitable for the QAP. After analyzing the adequacy of the Hamming distance for the QAP, we proposed an algorithm that implements a Hamming-based Kernels of Mallows Models (KMMs) EDA. In order to analyze the performance of the proposed approach, we compare it to other non-hybrid EDAs presented in the literature. The conducted experimentation showed that, for the QAP, (i) Hamming KMM EDA performs better than other classical EDAs, (ii) it also performs better than other model based EDAs in the literature, and (iii) the use of Kernels on a Hamming-based MM is the key to the successful performance of the algorithm. Specifically, Hamming KMM EDA is able to outperform the rest

of the methods in 56.7% of the studied instances. Not only that, but Hamming KMM EDA is also more stable than the competitors in terms of maximum Average Relative Deviation Percentage, with an value of less than 2.5% in the most difficult instance for this algorithm.

The incorporation of Hamming-based KMMs to the EDA framework in a competitive manner opens new research directions worth considering. For instance, this method could potentially be applied to other permutation problems, and even in non-permutation based combinatorial problems, if the solution space of the problem can be encoded by vectors. Because the Hamming distance measures the mismatches, regardless of the order, we believe that this method could be especially successful in combinatorial problems where the order of the elements in the vector is not as relevant as the absolute position of the items, such as the graph-partitioning problem [Buluç et al., 2016] or the .

Supplementary Material

Code to Reproduce the Results

The code to reproduce the results in this chapter are available in the repository https://github.com/EtorArza/SupplementaryKMMHamming.

On the Suitability of Bayesian Performance Analysis

We justify why Bayesian Performance Analysis is suitable for the experimentation of this chapter in the document available at https://github.com/EtorArza/SupplementaryKMMHamming/blob/master/comparison_between_BPA_and_NHST.pdf.

4

Multi-domain problem analysis

Problem analysis methods such as Fitness Landscape Analysis [Ochoa and Malan, 2019] and Local Optima Networks [Ochoa et al., 2014] help us better understand optimization problems. One application of these analysis methods is choosing the right algorithm for an optimization problem, which is known as the algorithm selection problem [Rice, 1976]. However, one of the limitations of the problem analysis methods in the literature is that they are particular to a solution space. For example, local optima networks assume a combinatorial optimization problem [Ochoa et al., 2014], or exploratory landscape analysis Mersmann et al. [2011a] assumes a continuous optimization problem.

In this chapter, we propose two multi-domain problem analysis methods, applicable to a wide range of optimization problems in both combinatorial and continuous domains. The proposed methods analyze a set of optimization problems and identify the differences and similarities in the properties of the optimization problems in the set. Given a methodology to adapt an optimization algorithm for one problem, we define *transferability* as the performance of the learned optimization algorithm when applied to a different problem. Similarly, we define the *behaviour* of an optimization algorithm as how it performs optimization in a problem, given that it has previously been "adapted" to solve a different optimization problem.

To measure *transferability* and the *behaviour*, we introduce a multi-domain hyper-heuristic framework based on neural network controllers. The controller guides the optimization process and can be adapted with a training process on an optimization problem. The experimentation carried out on four problem sets demonstrates that the proposed analysis methodology is useful for identifying differences and similarities in the properties of optimization problems in both the continuous and combinatorial domains.

4.1 Introduction

Hyper-Heuristic algorithms automate the generation and selection of heuristic and metaheuristic algorithms [Burke et al., 2013]. Classical proposals approached this task by focusing on a certain optimization problem domain [Keller and Poli, 2008, Grobler et al., 2010, Meignan et al., 2010, Burke et al., 2012, Martin et al., 2016, Stanovov et al., 2022].

Great progress has been made in "cross-domain hyper-heuristics", where the same hyper-heuristic is be applied to different optimization problem domains [Sabar et al., 2013, 2015]. In this context, HyFlex [Ochoa et al., 2012] introduced a popular benchmark platform, proposing six types of combinatorial optimization problems and a set of low-level heuristic algorithms. Nonetheless, the proposal is only applicable to combinatorial type problems [Ochoa et al., 2012]. Revising the literature in the continuous domain, some hyper-heuristics have been proposed [Cruz-Duarte et al., 2020, Caraffini et al., 2019], although to a lesser extent [Pillay and Qu, 2018]. Some approaches in this domain have focused on algorithm selection [Jankovic and Doerr, 2020], improving the parameter control of existing heuristics [Shala et al., 2020], or the discovery of new variants of existing heuristics [Poli et al., 2005].

On the other hand, problem analysis methods such as Fitness Landscape Analysis [Ochoa and Malan, 2019] and Local Optima Networks [Ochoa et al., 2014] help us better understand optimization problems. One application of these analysis methods is choosing the right algorithm for an optimization problem, which is known as the algorithm selection problem [Rice, 1976]. However, one of the limitations of the problem analysis methods in the literature is that they are particular to a solution space. For example, local optima networks assume a combinatorial optimization problem [Ochoa et al., 2014], or exploratory landscape analysis Mersmann et al. [2011a] assumes a continuous optimization problem.

Contribution

In this chapter, we propose a multi-domain problem analysis method based on a hyper-heuristic framework. Domain generality is achieved with two domain specific parts, the encoder and the decoder, that interact with the optimization problem, and need to be defined for the problem domain. This makes the rest of the hyper-heuristic (the training procedure and the problem analysis methodology) domain agnostic. To illustrate the multi-domain applicability of the problem analysis method, we empirically show that the analysis carried out in both the continuous and combinatorial domains is correlated with the properties of the optimization problems themselves. Therefore, the proposed analysis methods are more widely applicable than other approaches that might be domain-dependent, such as fitness landscape analysis [Pitzer and Affenzeller, 2012] or local optima

networks [Ochoa et al., 2014]. In addition, the hyper-heuristic framework can also be used to generate or control optimization algorithms.

Organization

The rest of the chapter is organized as follows: the next section introduces the multi-domain hyper-heuristic framework. Afterward, in Section 4.3 the proposed optimization problem embedding methodology is described. Then, in Section 4.4, we carry out an experimental study. Finally, Section 4.5 concludes the chapter and gives research lines for future work.

4.2 Multi-Domain Hyper-Heuristic

The problem analysis method is based on the "adaptation" process of the hyper-heuristic framework. Hence, before focusing on the analysis method, in the following, we introduce the multi-domain hyper-heuristic framework.

The proposed hyper-heuristic framework is a population-based algorithm with a neural network-based model, called the controller, that decides how the solutions are modified at each iteration. The controller is a trainable model with three interconnected parts, that together, modify solutions according to a previously learned behavior. To achieve domain generality, the controller (see Figure 4.1 for a broader look) has two domain-specific parts that depend on the domain of the problem being solved: an encoder and a decoder. These need to be specified before applying the hyper-heuristic and their purpose is to give the hyperheuristic framework domain generality.

The framework is applicable to any problem domain, given that an encoder/decoder pair can be defined for that domain. To showcase this domain generality, in this chapter we implemented two encoder/decoder pairs, one pair for continuous problems and another for permutation based problems. The proposed hyper-heuristic is not a cross-domain approach [Pillay and Qu, 2018], in the sense that it needs to be trained for a specific encoder/decoder pair. Instead, it is a multi-domain approach in the sense that the training procedure and optimization loop (shown in Figure 4.1) are the same regardless of the domain, as is the problem analysis methodology (explained in detail in Section 4.3).

The hyper-heuristic starts with a set of solutions generated uniformly at random. Then, at each iteration, the controller modifies each solution σ_i in the population in three steps (see Figure4.1). First, the encoder gathers information about the optimization state into a real-valued vector $X \in \mathbb{R}^p$, which we call the *feature vector*. The feature vector is a representation of the current state of the optimization. Then the neural network maps this vector X into another vector $Y \in [-1, 1]^q$

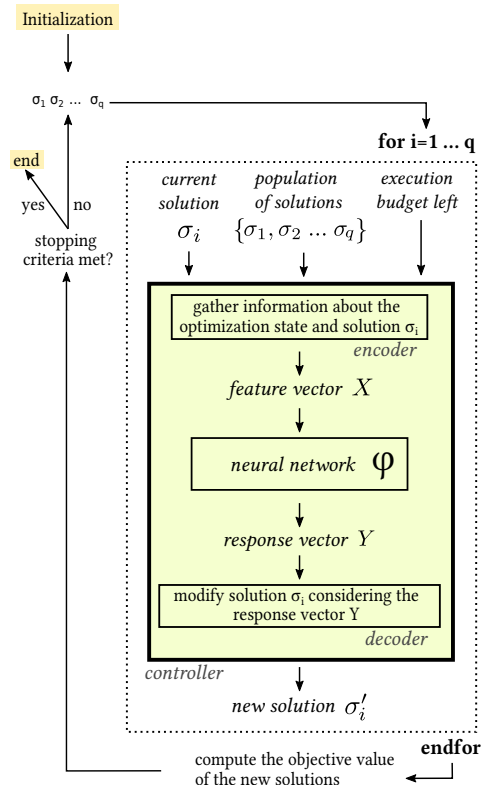


Fig. 4.1: Diagram of the proposed hyper-heuristic framework.

that we call the *response vector*. Finally, based on the response vector, the decoder modifies the solution σ_i. The decoder is an operator that, given a real valued vector Y and a solution σ_i, returns a modified solution.

This process is repeated for each solution σ_i in the population. Then, if the stopping criteria are met, the process is terminated and the best found solution is reported. Otherwise, the process is repeated with the new population of solutions replacing the old ones.

This hyper-heuristic framework is general and can be applied to different types of optimization problems with different search spaces. In this sense, the decoder and the encoder bridge the search space with the space in which the neural network operates (the neural network φ is mapping with n real inputs and m real outputs). The decoder/encoder pair is the only domain-specific component of the hyper-heuristic, and it needs to be compatible with the search space of the optimization

problem. If an encoder/decoder pair can be defined for a problem, then the hyper-heuristic is applicable to that problem.

In order to apply the hyper-heuristic for the optimization of a problem, it needs to be trained first. In this sense, we identify two different stages A) the training stage and B) the application stage. In what follows, we carefully describe each of these stages.

4.2.1 Training stage

The training of the hyper-heuristic involves training the neural network in the controller. The neural network is trained in a reinforcement learning setting with the Neuroevolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002] algorithm. Specifically, given an optimization problem, NEAT tries to find the neural network design that maximizes a score obtained from the application of the hyper-heuristic in the problem, with the neural network in the controller (see Appendix 7.3.1 for additional details).

The training stage (Algorithm 4) starts with an initial random set of m neural networks (a single hidden layer neural networks with random weights) $\bar{\varphi} = \{\varphi_1, ..., \varphi_m\}$ (line 1), and the score of the neural networks is computed as the best objective function value they obtain when being used to optimize the given optimization problem I (line 4). Several executions are averaged to obtain a more certain estimation of the score of the neural network. Next, the most promising neural networks are chosen and combined, creating new neural networks (line 5). Finally, mutations are applied to the new neural networks and their performance is once again tested, concluding the current generation. This process is repeated until a stopping criterion is reached (line 2). The training stage concludes when the best neural network in the last generation is returned (lines 7-8).

4.2.2 Application stage

Once the controller is trained, the trained hyper-heuristic can be applied to optimize other unseen instances of the same problem. In addition, it can also be applied to different problems defined in the same domain.

4.2.3 Interface with the optimization problem

As mentioned before, the hyper-heuristic depends on a three-part neural network-based model (*controller*) that decides how the solutions are to be modified at

Algorithm 4: Training the neural network with NEAT [Stanley and Miikkulainen, 2002]

Input:
I: The training optimization problem.
t_{train}: The maximum training time.

$\bar{\varphi} = (\varphi_1, ..., \varphi_m) \leftarrow$ randomly initialize a set of neural networks
while $t < t_{train}$ **do**
  for $j = 1, ..., m$ **do**
    $g_j \leftarrow$ optimize problem I with the hyper-heuristic, with the neural network φ_j in the controller
  $\bar{\varphi} \leftarrow$ select, crossover and mutate the neural networks $\bar{\varphi}$ based on the computed scores g_j
  t $\leftarrow$ elapsed_time
res $\leftarrow \varphi_j \in \bar{\varphi}$ with the highest g_j
return res

each iteration (the green square in Figure 4.1). Two of these parts are domain-specific (the encoder and the decoder) to achieve domain generality. To validate the proposed approach, in this chapter we considered four problem sets in two different search spaces: the space of permutations of size n and $\mathbb{R}^n$. Therefore, we proposed two encoder/decoder pairs, one pair for each search space and we give a brief overview of how they work in the following.

4.2.3.1 Continuous problems

The controller for continuous problems is based on the well-known particle swarm optimization algorithm [Kennedy and Eberhart, 1995]. This is a population based optimization algorithm for continuous problems, where each solution is updated via a linear combination of three vectors. Specifically, each solution σ_i is moved towards the best solution so far (denoted as σ_{best}) and the best solution visited by σ_i (denoted as $\sigma_{i,\text{best}}$). The move made in the last iteration is also used as a reference movement for the next iteration. These three moves are represented by three vectors, which are then added to σ_i.

Specifically, the modified solution σ_i' is obtained as

$$\sigma_i' = \sigma_i + \Delta_{t,i} \tag{4.1}$$

and

$$\Delta_{t,i} = y_1 \cdot r_1 \cdot (\sigma_i - \sigma_{\text{best}}) + y_2 \cdot r_2 \cdot (\sigma_i - \sigma_{i,\text{best}}) + y_3 \cdot \Delta_{t-1,i}$$

where $\Delta_{t-1,i}$ is the $\Delta_{t,i}$ of the previous iteration for solution σ_i and r_1, r_2 are two random numbers sampled uniformly at random from the interval $(0, 1)$.

Choosing the weights (y_1, y_2, y_3) to use in this linear combination is still an open problem [Engelbrecht and Cleghorn, 2020]. The proposed hyper-heuristic learns how to choose these weights for each solution σ_i to be modified. This is done in three steps, as shown in the green box in Figure 4.1.

In the first step (encoder), a real valued feature vector $X = (x_1, x_2, ..., x_{10})$ is produced with values in the interval $(0, 1)$. These values represent the solution σ_i and the state of the optimization procedure:

- x_1, x_2 indicate the absolute L_1 distance from the current solution σ_i to the best solution in the population and the average solution respectively.
- x_3, x_4, x_5 describe the relative distance from the current solution to the closest, best, or average solution respectively. First, the absolute distances are computed for every solution in the population and then the relative distances are set in the interval $(0, 1)$ as a relative ranking of their absolute counterparts.
- x_6 is the proportion of the computation budget used so far.
- x_7 is proportional to the relative ranking of solutions in the population σ_i with respect to the rest of the solutions. Since the solutions in the population are sorted according to their objective value before X is computed, a value of $\frac{i}{q}$ is assigned to x_4, where q is the population size and i is the relative ranking of the objective value of σ_i with respect to the rest of the solutions in the population.
- x_8 is 1 if σ_i improved its best found solution and 0 otherwise.
- x_9 is 1 if σ_i is the best-found solution so far and 0 otherwise.
- x_{10} is a random number in the interval $(0, 1)$.

The second step involves feeding X into the neural network, and we get the response vector $Y = (y_1, y_2, y_3)$. In the third step (decoding), σ_i is modified with the update rule in Equation (4.1).

4.2.3.2 Permutation problems

The controller for permutation problems considered in this chapter is based on local search and related techniques such as simulated annealing [van Laarhoven and Aarts, 1987]. As with the continuous controller, the controller for permutation problems modifies a solution σ_i in three steps. First, the feature vector $X = (x_1, x_2, ..., x_8)$ is produced (encoder), containing information about the optimization state.

- x_1, x_2 and x_3 indicate whether σ_i is a local optimum (1) or not (0), for each of the three operators considered in this chapter: adjacent swap, exchange and insert [Schiavinotto and Stützle, 2007].
- x_4 is proportional to the relative ranking of solutions in the population σ_i with respect to the rest of the solutions. Since the solutions in the population are sorted according to their objective value before X is computed, a value of $\frac{i}{q}$ is assigned to x_4, where q is the population size and i is the relative ranking of the objective value of σ_i with respect to the rest of the solutions in the population.
- x_5 is set according to the computational budget spent so far.
- x_6 is proportional to the relative ranking of the differences in the objective value with respect to the previous solution in the population. In other words, a value proportional to the ranking of $f(\sigma_{i-1}) - f(\sigma_i)$, where σ_i is the solution to be modified, and σ_{i-1} is the solution that is next to σ_i in terms of relative ranking.
- x_7 and x_8 are proportional to the Hamming and Spearman's footrule [Diaconis and Graham, 1977] distances from σ to a median permutation σ_0 that satisfies $\sigma_0 = \text{argmin}_\sigma \sum_{i=1}^{q} d(\sigma, \sigma_i)$, where d is the Hamming and the Kendall distance [Fligner and Verducci, 1986b] respectively, and σ_i is the permutation to be modified. The Hamming median permutation can be obtained by solving a linear assignment problem [Irurozki et al., 2019b]. In the case of the Kendall distance, the median permutation is approximated using the Borda algorithm [Cook and Seiford, 1982, de Borda, 1781].

Then, in the second step, we feed X into the neural network, and we get the response vector $Y = (y_1, y_2, ..., y_{11})$. Finally, in the third step, σ_i is modified according to the response vector. The decoder for permutation problems considered in this chapter combines and parametrizes heuristic components commonly used in the field of evolutionary computation such as simulated annealing [Glover, 1986], local search, and variable neighborhood search [Mladenović and Hansen, 1997], among others. In the following, we give a detailed explanation of this process.

Decoder for permutation problems

Algorithm 5 shows how the decoder combines and controls heuristic components based on the output of the neural network to modify permutation σ_i. First, y_1 determines if solution σ_i should be randomly reinitialized or not (lines 1-3). Next, the operator to be used is specified by the argmax of (y_2, y_3, y_4), each value representing swap, exchange, and insert operators, respectively (line 4). Then, y_5 chooses between three heuristic options. If $|y_5| < 0.25$ is satisfied, then the solution remains unaltered (lines 5-6). If $y_5 < -0.25$, then a local search iteration is applied with

the chosen operator (lines 7-8). Finally, if $y_5 > 0.25$ then σ_i is modified without local search (lines 10-14).

In this case, a perturbation is applied to σ_i such that its distance to a reference permutation is increased or decreased. If the modification decreases the objective value, it is only accepted with a certain probability (line 10). The reference permutation is specified by the last five values of the response $(y_7, y_8, y_9, y_{10}, y_{11})$, each value associated with a different reference permutation as shown in Table 4.1. The reference permutation is chosen randomly, considering probabilities proportional to the absolute value of the reference coefficients (line 11-12). The direction of the modification is chosen according to the sign of the corresponding reference coefficient, with a positive value representing a modification towards the reference, and a negative value a modification away from the reference (line 13). Finally, the modification is applied to the solution (line 14). We refer the interested reader to Appendix 7.3.2 with additional details on exactly how the modification is applied (line 14)

Table 4.1: Reference permutations available to the decoder

Reference index	**Reference permutation (σ_{ref})**
$ref = 7$	Hamming median permutation. Already computed when generating the feature vector element x_7.
$ref = 8$	Kendall median permutation. Already computed when generating the feature vector element x_8.
$ref = 9$	Best solution that σ_i has visited in past iterations.
$ref = 10$	Best known solution.
$ref = 11$	σ_{i-1}, which is the permutation that is closest in objective value to the solution being modified.

4.3 The hyper-heuristic framework as a tool to analyze optimization problems

In this section, we will introduce two methods for analyzing a set of optimization problems. The purpose of these methods is to identify similarities and differences among optimization problems in a problem set. We will start with a brief overview of the motivation, related works, and potential applications of these analysis methods.

Algorithm 5: decode(σ_i,Y)

Input:
$Y = (y_1, ..., y_{10})$: Response for solution σ_i.
σ_i: The permutation to be modified.

if $y_1 > 0.25$ **then**
  return random permutation
end
operator $\leftarrow$ argmax(y_2, y_3, y_4) `// Corresponding to the swap, exchange and insert respectively`
if $|y_5| < 0.25$ **then**
  return σ_i `// No changes to solution` σ_i
else if $y_5 < -0.25$ **then**
  return local_search_iteration(σ, *operator*)
else
  prob_accept_worse $\leftarrow \frac{y_6+1}{2}$
  $ref \leftarrow$ choose $j \in \{7, ..., 11\}$ with prob. proportional to $|y_j|$
  $\sigma_{ref} \leftarrow$ select reference permutation as shown in Table 4.1
  direction $\leftarrow$ sign(y_{ref})
  return modify(σ_i, σ_{ref}, *direction*, *operator*, *prob_accept_worse*)
end

4.3.1 Motivation and Applications

Analyzing a set of optimization problems has many applications, such as anomaly detection for a sequence of optimization problems. Let's assume that a local post office needs to solve multiple combinatorial optimization problems (such as the orienteering problem [Kobeaga et al., 2018] or the traveling salesman problem [Goldberg and Lingle, 1985]) each day to plan the routes for deliveries. A factory job planner might also have to apply the flowshop scheduling problem [Gupta and Stafford, 2006] on a regular or even dynamic basis if circumstances change [Li et al., 2023]. In these examples, it would be useful to detect when a new problem instance is very different from the rest. This might indicate that the optimization algorithm currently being used is not appropriate for the new problem instance, and new algorithms may need to be developed to solve the problem more effectively. More importantly, it might also indicate a problem with the problem instance data or another type of anomaly [Carreño et al., 2020] that requires further investigation.

Analyzing the properties of optimization problems is also applicable to algorithm selection [Janković and Doerr, 2019] and benchmark design. For instance, when choosing a set of optimization problems for a benchmark, it might be desirable that the properties of the optimization problems are balanced [Dreo et al., 2019].

For example, if the benchmark has 20 problem instances, it would not make sense to have 19 instances with similar properties and another completely different instance. Real-world examples, including benchmarks for optimizing energy usage in buildings [Wölfle et al., 2020] or car structural design [Kohira et al., 2018], show that benchmark design is relevant beyond the optimization community.

Related works

There are many problem analysis methods in the literature, although, in general, they are more domain-specific than the proposed approach. We argue that the main advantage of the proposed problem analysis methods is that the analysis procedures do not depend on the problem domain. This means that they are more widely applicable than other analysis approaches that might be domain-dependent, such as fitness landscape analysis [Pitzer and Affenzeller, 2012], local optima networks [Ochoa et al., 2014], or exploratory landscape analysis [Mersmann et al., 2011b]. For instance, exploratory landscape analysis was designed for combinatorial optimization problems [Smith-Miles, 2009] and has been adapted to continuous problems with different approaches [Munoz et al., 2015]. However, the same exploratory landscape analysis method is not applicable to both combinatorial and continuous optimization problems.

The analysis of a set of optimization problems via transferability has already been studied. Hong et al. [2018] proposed measuring transferability as the average objective value of the hyper-heuristic when training and testing on two continuous optimization problems. Our experimental setup on transferability improves the methodology of Hong et al. [2018] in four key aspects.

Firstly, by defining transferability with ranks instead of objective values, we can compare across different problems (the magnitude of transferability is the same for every problem). Secondly, by repeating several measurements of transferability and computing the average ranks, we can distinguish between noise and the actual difference in performance (this is not possible with average objective values). Thirdly, we experimentally show that the analysis carried out with our approach is correlated with the properties of the optimization problems, which suggests that the proposed technique is useful for finding similarities and differences in the properties of optimization problems. Finally, we show that our approach works in both combinatorial and continuous domains, while the approach by Hong et al. [2018] was only shown to work in the continuous domain. See Appendix 7.3.5.1 for additional details on exactly how our approach overcomes the limitations of the analysis proposed by Hong et al. [2018].

4.3.2 Via the performance of the hyper-heuristic

The first method involves measuring the performance of the hyper-heuristic in the problem set, and how it varies when applied to different problems in the set. In this context, we define the *transferability* as the relative loss of performance of the hyper-heuristic when it is trained on a problem (the *train* problem) and applied in another problem (the *test* problem). By computing the transferability between every pair of problems in a problem set, we can embed these problems into a $k \times k$ real matrix, which we named *transferability matrix*.

4.3.2.1 Transferability matrix

Given a set of optimization problems $\mathfrak{P}_1, ..., \mathfrak{P}_k$, the transferability from problem $\mathfrak{P}_i$ to problem $\mathfrak{P}_j$, denoted as $T_{i,j}$, is a real value in the interval $[0, 1]$ that represents the relative performance of the hyper-heuristic in the test problem $\mathfrak{P}_j$, when it has been trained in problem $\mathfrak{P}_i$ (a lower transferability indicates better performance). This performance is relative to the performance observed when the hyper-heuristic has been trained in another different problem $\mathfrak{P}_{i'}$, and tested in the same problem $\mathfrak{P}_j$.

More precisely, given a test problem $\mathfrak{P}_j$, we train the hyper-heuristic in all the problems $\mathfrak{P}_1, ..., \mathfrak{P}_k$, and set $T_{i,j} = 0$ for the training problem $\mathfrak{P}_i$ that produced the best performance in the test problem $\mathfrak{P}_j$, and $T_{i,j} = 1$ for the training problem $\mathfrak{P}_i$ that produced the worst performance (the rest of the problems get values in between, proportionally to their ranking in performance). We repeat this process for each test problem $\mathfrak{P}_j$, thus obtaining the transferability $T_{i,j}$ between every possible pair of problems. The *transferability matrix* is defined with the element $T_{i,j}$ in row i and column j. A more formal definition of the transferability is available in Appendix 7.3.3.

In a second stage, we reorder the rows and columns in the transferability matrix. The problems $\mathfrak{P}_1, ..., \mathfrak{P}_k$ in a given set are not ordered: the index i of a problem $\mathfrak{P}_i$ has no meaning. Consequently, we can reorder the problems such that problems with a similar transferability have a similar index. To do so, we reorder the optimization problems such that the loss

$$\sum_{j=1}^{k} \sqrt{\sum_{i=1}^{k-1} (T_{i,j} - T_{i+1,j})^2} + \sum_{i=1}^{k} \sqrt{\sum_{j=1}^{k-1} (T_{i,j} - T_{i,j+1})^2} \tag{4.2}$$

is minimized. Minimizing this loss implies that adjacent columns and rows in the transferability matrix are as similar as possible to each other.

4.3.2.2 Interpretation

Figure 4.2 shows an example transferability matrix. If the transferability $T_{i,j}$ is low, the interpretation is "the performance of the hyper-heuristic on problem $\mathfrak{P}_j$ was relatively high when trained on problem $\mathfrak{P}_i$". Now we can go a step further by comparing several $T_{i,j}$ to each other.

Let us assume that problems $\mathfrak{P}_1$ and $\mathfrak{P}_2$ are the same optimization problem. Then, it is likely that $T_{1,j} \approx T_{2,j}$ and $T_{i,1} \approx T_{i,2}$. Consequently, $\mathfrak{P}_1$ and $\mathfrak{P}_2$ will be placed close to each other in the transferability matrix. This suggests that problems that are similar to each other from the point of view of the performance of the hyper-heuristic will be placed next to each other in the matrix.

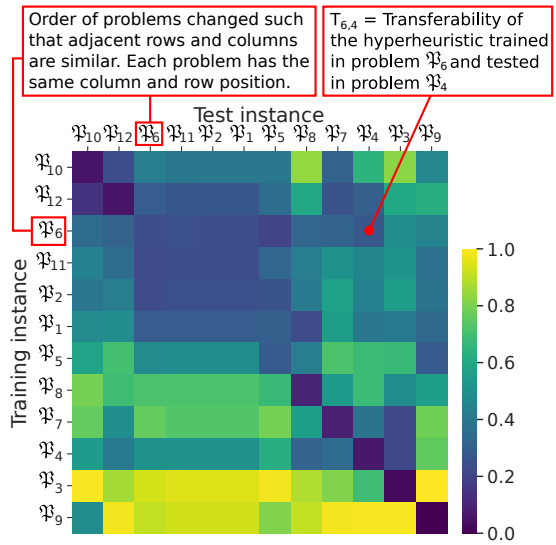


Fig. 4.2: An example transferability matrix. A darker color implies a lower (better) transferability.

4.3.3 Via the behavior of the hyper-heuristic

The second analysis method takes into account how the hyper-heuristic carries out the optimization process. In this sense, the behavior of the trained hyper-heuristic (how the lower-level heuristics are applied to each solution) is determined by the outputs of the neural network (denoted as response vectors in this chapter). In fact,

the set of all the response vectors generated when applying the hyper-heuristic in an optimization problem allows us to replicate the optimization process carried out. Therefore, it is possible to summarize the behavior of the hyper-heuristic by averaging all the response vectors generated when applying the hyper-heuristic in an optimization problem.

Given a set of optimization problems $\mathfrak{P}_1, ..., \mathfrak{P}_k$, we want to identify the similarities/differences of the problems by looking at the behavior of the hyper-heuristic when trained and tested in these problems. To do so, first, we train the hyper-heuristic in problem $\mathfrak{P}_i$ and compute the average response $R(\mathfrak{P}_i \rightarrow \mathfrak{P}_j)$ generated when applying the hyper-heuristic in problem $\mathfrak{P}_j$. We repeat this process for every possible pair of optimization problems $\mathfrak{P}_i, \mathfrak{P}_j \in \{\mathfrak{P}_1, ..., \mathfrak{P}_k\}$, and we obtain 10 samples for each possible pair.

Next, we fit a Linear Discriminant Analysis [Singh, 2020-08-18, 2020, Rao, 1948] (LDA). In Appendix 7.3.6, we justify why fitting a LDA is suitable for this purpose. The class to fit the LDA is the problem $\mathfrak{P}_i$ used to train the hyper-heuristic. Once the LDA is fitted, we compute the average response in each train problem $\mathfrak{P}i$: $\frac{\sum_{\mathfrak{P} \in \{\mathfrak{P}_1, ..., \mathfrak{P}_k\}} R(\mathfrak{P}_i \rightarrow \mathfrak{P})}{k}$. Then, we project the average response in each train problem into a 2D space. This generates an embedding of the optimization problems in $\mathbb{R}^2$ (one point for each train problem $\mathfrak{P}_i$). We denote this embedding as *the LDA of a problem set*.

Once the embedding has been computed, we can interpret it. The interpretation is simple: the behavior of the hyper-heuristic is similar in two problems if they are close to each other in the projected space.

4.4 Experimental study

The purpose of this experimental section is to validate the utility of the proposed hyper-heuristic framework. The first part of the experimentation is devoted to, as a baseline, show that the hyper-heuristic framework can improve existing heuristics beyond a parameter search approach. Then, the second part focuses on the application of the two optimization problem analysis methods described in the previous section. We apply the two analysis methods and then we take a look at how these analyses correlate with the properties of the optimization problems. The code to reproduce the experiments is available in our GitHub repository at https://github.com/EtorArza/TransfHH.

4.4.1 Problem sets

The experimentation was carried out in four problem sets. In the first problem set i), we considered twelve classical continuous optimization problems chosen from the virtual library of simulation experiments [Surjanovic and Bingham]. Furthermore, in the second problem set ii), we used the continuous problem instance generator from Rönkkönen et al. [2008] to experiment on optimization problems with a different number of local optima. Regarding the combinatorial domain, in the third problem set iii) four permutation problems were considered: the Traveling Salesman Problem [Goldberg and Lingle, 1985], the Quadratic Assignment Problem [Koopmans and Beckmann, 1957], the Linear Ordering Problem [Schiavinotto and Stützle, 2004] and the Permutation Flowshop Scheduling Problem [Gupta and Stafford, 2006] and four instances of different sizes and benchmarks were chosen for each problem (in total 16 instances). In the final problem set iv), we considered different instance benchmarks from the Quadratic Assignment Problem. In this section, we provide additional details on the four i) - iv) problem sets.

i) Twelve continuous optimization problems

These are the twelve continuous optimization problems that were considered in this chapter, six ($\mathfrak{P}_1$–$\mathfrak{P}_6$) bowl-shaped and six with many local optima ($\mathfrak{P}_7$–$\mathfrak{P}_{12}$).

$\mathfrak{P}_1$ Sphere, $\sum_{i=1}^{d} x_i^2$

$\mathfrak{P}_2$ Rotated hyper-ellipsoid, $\sum_{i=1}^{d} \sum_{j=1}^{i} x_j^2$

$\mathfrak{P}_3$ Trid function, $\sum_{i=1}^{d} (x_i - 1)^2 + \sum_{i=1}^{d} x_i \cdot x_{i-1}$

$\mathfrak{P}_4$ Log sphere, $\sum_{i=1}^{d} \log(x_i)^2$

$\mathfrak{P}_5$ Sum of powers, $\sum_{i=1}^{d} |x_i|^i$

$\mathfrak{P}_6$ Sum squares, $\sum_{i=1}^{d} i \cdot x_i^2$

$\mathfrak{P}_7$ Langermann, $\sum_{i=1}^{5} G_i \exp(-\frac{1}{\pi} F_{i,j}) \cos(\pi F_{i,j})$ where $G = \{1, 2, 5, 2, 3\}$, $F_{i,j} = \sum_{j=1}^{i} (x_j - D_{i,j|2})^2$, $j|2$ is the reminder of j divided by 2 and

$$D = \begin{bmatrix} 3 & 5 \\ 5 & 2 \\ 2 & 1 \\ 1 & 4 \\ 7 & 9 \end{bmatrix}$$

$\mathfrak{P}_8$ Schwefel, $\sum_{i=1}^{d} x_i \cdot \sin(\sqrt{|x_i|})$

$\mathfrak{P}_9$ Rastrigin, $10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i)\right]$

$\mathfrak{P}_{10}$ Levy,
$\sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] +$
$(w_d - 1)^2[1 + \sin^2(2\pi w_d)]$, where $w_i = 1 + \frac{x_i - 1}{4}$

$\mathfrak{P}_{11}$ Griewank, $\sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1$

$\mathfrak{P}_{12}$ Ackley,
$-20\exp\left(\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right)$

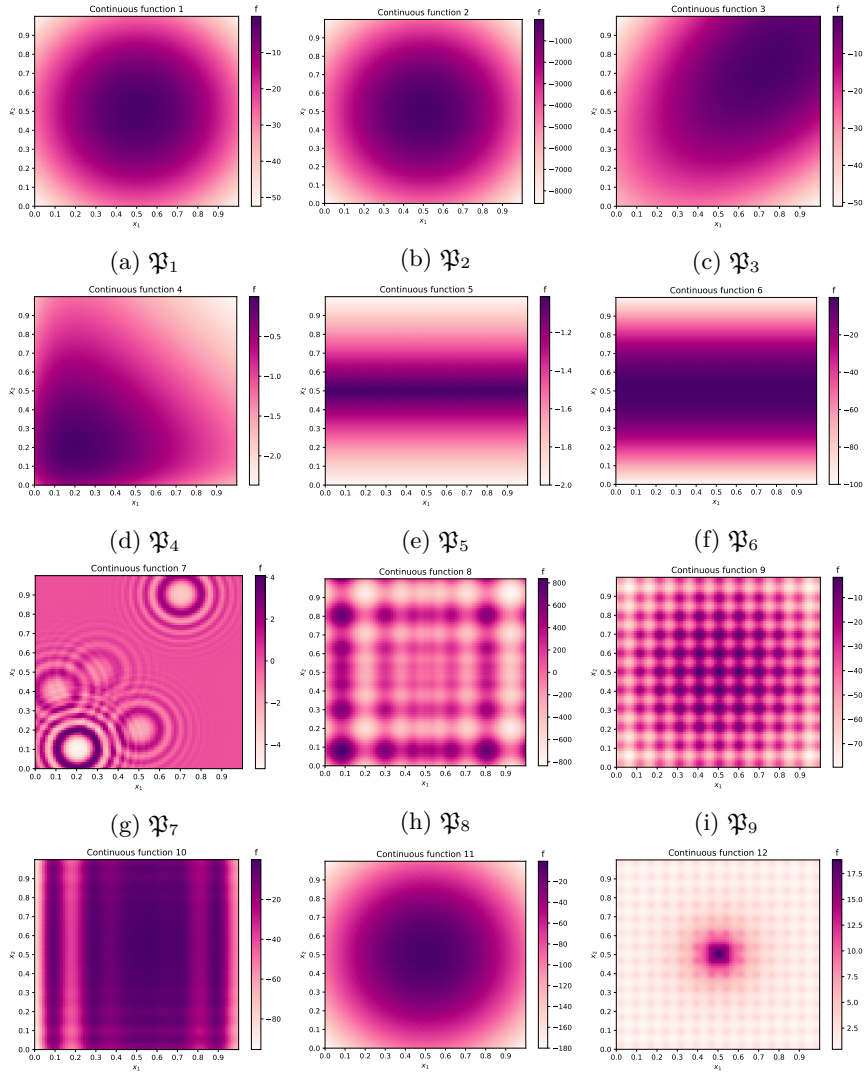
All of these functions (except for the log sphere function) were taken from the Virtual Library of Simulation Experiments [Surjanovic and Bingham]. Also, note that the version of the Langermann function considered in this chapter has been modified to be suitable for any positive dimension d. We considered a dimension size of 20 for every problem.

The search space for each problem was set according to the Virtual Library of Simulation Experiments [Surjanovic and Bingham]. In addition, each time a problem was loaded, the search space was reduced by a random percentage between 0% and 10%. So for example, the search space of $\mathfrak{P}_1$ is $[-5.12, 5.12]^d$ as defined in the Virtual Library of Simulation Experiments [Surjanovic and Bingham]. Therefore, the reduced search space would be

$$[-5.12 + (5.12 - (-5.12)) \cdot \delta_1, 5.12 - (5.12 - (-5.12)) \cdot \delta_2] =$$
$$[-5.12 + 10.24 \cdot \delta_1, 5.12 - 10.24 \cdot \delta_2],$$

with δ_1, δ_2 chosen uniformly at random from the interval $[0, 0.1]$. This reduction in the search space slightly moves the position of the optimal solution avoiding any possible bias associated with the movement of the solutions in the direction of the coordinate axes [Janson and Middendorf, 2007] or the center of the search space [Monson and Seppi, 2005].

In Figure 4.3, we show the contour plot of the two dimension versions of these functions, with the search space linearly transformed to $[0, 1]^d$ for easier visualization.



(a) $\mathfrak{P}_1$ (b) $\mathfrak{P}_2$ (c) $\mathfrak{P}_3$

(d) $\mathfrak{P}_4$ (e) $\mathfrak{P}_5$ (f) $\mathfrak{P}_6$

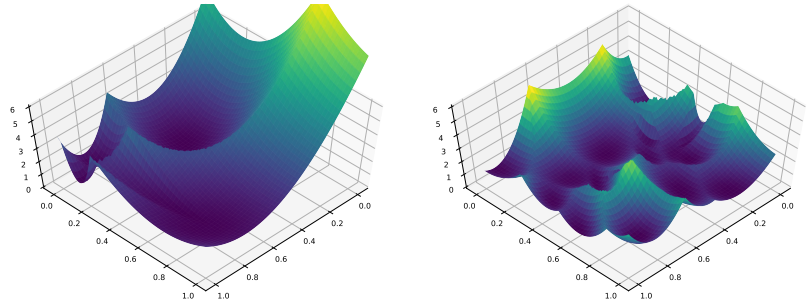
(g) $\mathfrak{P}_7$ (h) $\mathfrak{P}_8$ (i) $\mathfrak{P}_9$

(j) $\mathfrak{P}_{10}$ (k) $\mathfrak{P}_{11}$ (l) $\mathfrak{P}_{12}$

Fig. 4.3: The contour plots for optimization problems $\mathfrak{P}_1$ - $\mathfrak{P}_{12}$.

ii) Continuous problem generator

In addition to classical continuous optimization algorithms, we considered an optimization problem generator that can produce optimization problems with a different number of local optima. Specifically, we considered the "quadratic family" of the optimization problem generator by Rönkkönen et al. [2008] (see Figure 4.4 for two examples). Optimization problems with 1, 2, 4, 8, 16, 32, and 64 local optima were generated.



(a) 4 local optima (b) 16 local optima

Fig. 4.4: Two random minimization problems with a different number of local optima, obtained with the optimization problem generator by Rönkkönen et al. [2008]. Both of them are two-dimensional problems, with the vertical axis as the objective function.

iii) Four permutation-based optimization problems

For this problem set, we chose four well known optimization problems in the literature: the Traveling Salesman Problem, the Permutation Flowshop Scheduling Problem, the Linear Ordering Problem, and the Quadratic Assignment Problem. In the following, we give additional details on the chosen problems.

Traveling Salesman Problem (TSP): Given a set of n cities, the goal of the TSP [Goldberg and Lingle, 1985] is to find a path that connects all the cities forming a single cycle while minimizing the length of the path. An instance of the problem is defined by the matrix $\mathbf{D} = [d_{i,j}]_{n \times n}$ containing the distances between any two cities. Given a permutation σ, the objective function value is computed as:

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

Permutation Flowshop Scheduling Problem (PFSP): Given a set of m machines and n tasks, the Permutation Flowshop Scheduling Problem (PFSP) [Gupta and Stafford, 2006] is the problem of optimally ordering the tasks so that a certain criterion is minimized. An instance of PFSP is defined by a matrix $\mathbf{P} = [p_{i,j}]_{n\times m}$, containing the processing time of task i in machine j. Each of the n tasks has to go through every machine $j \in [m]$ in order, and each machine can only process one task at a time. To compute the objective value of a solution $f(\sigma) = c_{\sigma(n),m}$, a recursive formula can be used:

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

Linear Ordering Problem (LOP): Given an integer matrix $\mathbf{B} = [b_{i,j}]_{n\times n}$, the goal of the Linear Ordering Problem [Schiavinotto and Stützle, 2004] is to find a simultaneous permutation of both rows and columns so that the sum of the values above the diagonal is maximized. The solutions can be encoded as permutations, so that given a permutation σ, the i-th column and row are assigned $\sigma(i)$-th column and row respectively. Formally, the objective function is defined as:

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

Quadratic Assignment Problem (QAP): In the QAP [Koopmans and Beckmann, 1957], we are given a set of n facilities and n locations, along with the distance between any two locations $\mathbf{D} = [d_{i,j}]_{n\times n}$, as well as the workflow between any two facilities $\mathbf{H} = [h_{i,j}]_{n\times n}$. The goal is to find an assignment σ (codified as a permutation) of each of the facilities to one of the locations, such that the objective function $f(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} h_{\sigma(i),\sigma(j)}$ is minimized.

Problem instances in permutation problems

Each permutation-based optimization problem has many problem instances [Elorza et al., 2019]. A problem instance is a specific set of parameters that defines the objective function of the optimization problem. The list of considered problem instances is shown in Table 4.2.

TSP	QAP	PFSP	LOP
eil76	tai75e01	tai50_5_0	N-be75np_150cut
rat99	tai75e02	tai50_5_1	N-be75oi_150cut
kroA100	tai80a	tai100_5_0	N-stabu3_150cut
kroB100	sko90	tai100_5_1	N-t65d11xx_150cut
eil101	sko100a	tai50_10_0	N-t70f11xx_150cut
pr107	sko100b	tai50_10_1	N-t75n11xx_150cut
ch130	tai100a	tai20_20_0	N-tiw56r54_150cut
pr136	tai100b	tai20_20_1	N-tiw56r72_150cut

Table 4.2: Lists the problem instances in problem set iii). Note that the LOP instances were originally of size 150 and have been reduced to size 75.

iv) Instances of the QAP

In addition to different permutation-based optimization problems, we also considered different problem instance benchmarks within the same problem (the QAP). A problem instance of the QAP is given by two square matrices: the flow matrix **H**, and the distances matrix **D**. The goal of the QAP is to minimize the cost associated with the flow and the distance described in these two matrices. In order to design a diverse experimentation setting, three types of QAP instances were chosen from the QAPLIB (the online encyclopedia of QAP instances [Burkard et al., 1997]), *taixxA*, *sko* and *taixxB*. According to the instance classification of Stützle [2006], each of these belongs to a different type of instance class. In the first class, *taixxA* instances have both matrices **H** and **D** generated uniformly at random with values in the range [1,100] and are symmetric. In the second class, *sko* instances [Skorin-Kapov, 1990] have the distance matrix **D** based on the Manhattan (L1) distance. Finally, *taixxB* instances are asymmetric and randomly generated [Burkard et al., 1997]. According to Stützle [2006], *taixxB* instances try to resemble the structure of real life instances.

Multiple instances of the same type and size are desired to experiment with the transferability and the response. However, it is difficult to find instances of the same type and size for the QAP. To obtain multiple instances of the same size and type, larger instances of the same type were cut, obtaining 7 instances of size 40 of the same type, with a total of 21 instances.[1]

[1] The problem instances in the iv) problem set are available for download at https://github.com/EtorArza/TransfHH/tree/master/src/experiments/permus/instances/transfer_qap_cut_instances.

4.4.2 Hyperparameters

The experimentation hyperparameters are listed below. The purpose of this chapter is not to maximize the objective value in each problem, instead, we want to experiment on the hyper-heuristic as a framework that is applicable in different domains. Following this idea, we thought it was desirable to use the same parameters for all problems.

Hyper-Heuristic parameters

- Population size: 8
- Stopping criterion: 400 evaluations

Training parameters (NEAT)

- Population size: 10^3 (default value [Dougherty, 2014])
- Stopping criteria: 4 days or 2000 generations (stop when either criterion is met)

Testing parameters

- Executions averaged: 10^4

A parameter search was conducted to choose the population size of the Hyper-Heuristic. A detailed justification for each of these parameters is available in the appendix for the interested reader.

4.4.3 Comparison to classical parameter search

The goal of this experiment is to measure the capability of the hyper-heuristic of improving an existing heuristic when compared to parameter tuning. To do so, we compare the hyper-heuristic with the standard particle swarm optimization (PSO) algorithm [Engelbrecht and Cleghorn, 2020] (see Section 4.2.3.1 for details). Specifically, we compare three ways of adjusting the three parameters (y_1,y_2,y_3).

Firstly, we try the *Default* parameter values

$$y_1 = 0.729844,\ y_2 = 1.49618,\ y_3 = 1.49618.$$

Engelbrecht proposed these values as a rule of thumb in a tutorial on PSO [Engelbrecht and Cleghorn, 2020] presented at the Genetic and Evolutionary Computation Conference 2020. Secondly, we adjust these three parameters in a *Grid-Search*

approach for each problem. The parameter search was carried out in the interval $[-1.96, 1.96]$ with the parameter space discretized in 20 slices for each dimension, for a total of 20^3 possible combinations. Finally, the *Hyper-Heuristic* is trained in the problems and learns to adjust these three parameters during the optimization.

The average objective values are shown in Table 4.3. We apply the sign test pairwise 6 times to compare the three approaches to each other in the two problem sets. We use the Holm-Bonferroni correction and we observe that the differences are statistically significant at a familywise $\alpha = 0.05$. This statistical analysis can be reproduced in R with the code below:

```
p_values <- c(binom.test(0,12)$p.value,
binom.test(0,12)$p.value,
binom.test(1,12)$p.value,
binom.test(0,7)$p.value,
binom.test(0,7)$p.value,
binom.test(0,7)$p.value)
p.adjust(p_values, method = "holm")
```

The results point out that *Grid-Search* performs better than *Default*. In addition, *Hyper-Heuristic* performs significantly better than both approaches. Hence, we conclude the proposed hyper-heuristic was able to improve the PSO algorithm beyond the parameter tuning approach.

We hypothesize that the hyper-heuristic has a better performance than the heuristic because the search space of the hyper-heuristic is a superset of the search space of the parameter search algorithm. In other words, if we limit the neural network in the hyper-heuristic to having a constant output during the whole optimization process, then the search space would be the same as in the parameter search approach. In this sense, unlike the parameter search approach, the hyper-heuristic can deal with a wider range of behaviors that take into account the state of the optimization.

4.4.4 Analyzing optimization problems

In Section 4.3 we introduced two different problem set analysis methods. The first method (via the performance) analyzes the transferability of the hyper-heuristic: how well it performs when trained and tested in different optimization problems in the problem set. The second method (via the behaviour) takes into account how the hyper-heuristic does optimization (how it controls the lower-level heuristics).

In this part of the experimentation we aim to showcase these two analysis methods. We show that the results of both analyses are correlated with each other and with the properties of the optimization problems.

problem set i)

Problem index	*Default*	*Grid-Search*	*Hyper-Heuristic*
$\mathfrak{P}_1$	-6.01363	-3.23881	**-2.30488**
$\mathfrak{P}_2$	-985.27378	-530.64685	**-466.17630**
$\mathfrak{P}_3$	117.12492	119.66920	**121.10483**
$\mathfrak{P}_4$	-0.80864	-0.55685	**-0.37820**
$\mathfrak{P}_5$	-1.00574	-1.00143	**-1.00097**
$\mathfrak{P}_6$	-177.81698	-86.25251	**-71.58724**
$\mathfrak{P}_7$	0.01097	0.01600	**0.02385**
$\mathfrak{P}_8$	3978.36547	**4587.93669**	4560.50831
$\mathfrak{P}_9$	-140.67744	-83.16599	**-64.85858**
$\mathfrak{P}_{10}$	-13.32011	-9.11814	**-6.49914**
$\mathfrak{P}_{11}$	-21.64781	-12.11095	**-7.94694**
$\mathfrak{P}_{12}$	10.70759	12.20933	**13.47027**

problem set ii)

Local optima	*Default*	*Grid-Search*	*Hyper-Heuristic*
1	-0.99395	-0.69141	**-0.48948**
2	-1.44586	-1.05357	**-0.81471**
4	-1.29121	-0.92589	**-0.65019**
8	-1.30823	-0.88735	**-0.59927**
16	-2.02794	-1.41315	**-0.99433**
32	-5.44415	-3.90858	**-2.44707**
64	-7.84039	-5.14373	**-3.82432**

Table 4.3: Comparison of the heuristic and the hyper-heuristic in the problem sets i) and ii). Higher is better (best value highlighted in bold).

We start the analysis with first method (via the performance).

4.4.4.1 Via the performance of the hyper-heuristic

In this first part, we apply the methodology proposed in Section 4.3.2 and analyze the four problem sets via the performance of the hyper-heuristic.

Problem set i)

The transferability matrix for this problem set is shown in Figure 4.5. First, observe that the values in the diagonal are very low (lower means better performance). This is the expected result: the hyper-heuristic will obtain the best possible objective value when it is trained and tested in the same problem.

Recall that, when generating the matrix, columns and rows are sorted together such that adjacent columns and rows are as similar as possible to each other

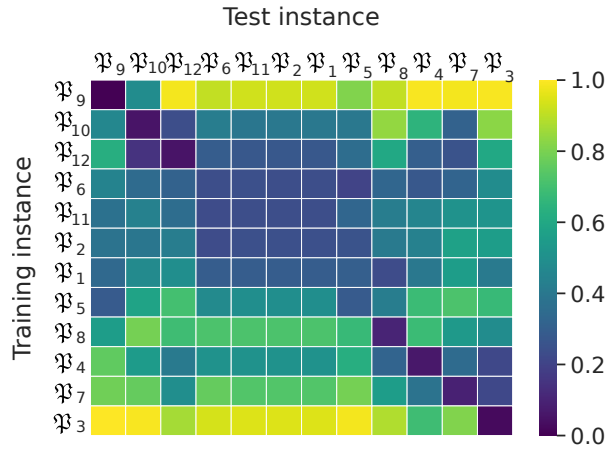


Fig. 4.5: Transferability matrix of problem set i).

(see Section 4.3.2 for a detailed explanation). Taking into account the similarities among adjacent columns and rows in the matrix, we defined three problem clusters (see Figure 4.6). We defined the first cluster as $\{\mathfrak{P}_9, \mathfrak{P}_{10}, \mathfrak{P}_{12}\}$, the second cluster as $\{\mathfrak{P}_6, \mathfrak{P}_{11}, \mathfrak{P}_2, \mathfrak{P}_1, \mathfrak{P}_5\}$ and the third as $\{\mathfrak{P}_8, \mathfrak{P}_4, \mathfrak{P}_7, \mathfrak{P}_3\}$. The hyper-heuristic had a low transferability when trained and tested in the same problem for problems in the first and third cluster (low values in the diagonal). When the hyper-heuristic was trained in any of the the problems in the second cluster, the transferability was decent in all the problems and it was not higher when trained and tested in the same problem (medium values in rows 4-8 in the transferability matrix).

Now, observe that the problems within each cluster have similar properties that are visible in their 2 dimension contour plots (contour plots shown in Figure 4.6). Problems in the first cluster have many local optima that are "comparably good to each other" in a large area of the search space, and have the global optimum in the center of the search space. Problems in the second cluster, on the other hand, have a strong quadratic component, and have the global optimum in the center of the search space. Finally, problems in the third cluster have the global optima in a corner of the search space.

In conclusion, the methodology proposed in Section 4.3.2 generated a transferability matrix for problem set i) that was very correlated with the properties of the optimization problems.

Problem set ii)

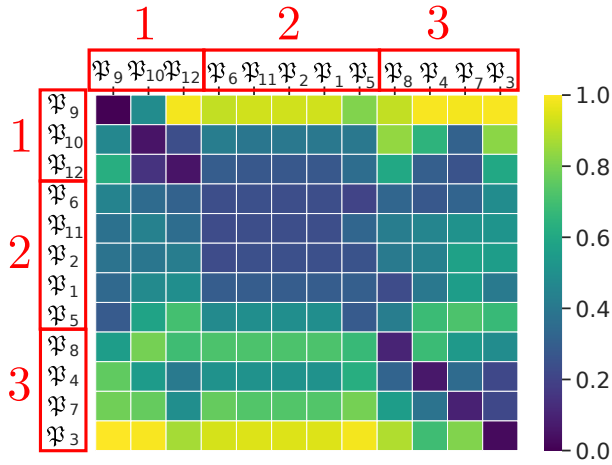


Fig. 4.6: The three problem set clusters for problem set i).

The transferability matrix for the second problem set is shown in Figure 4.7. Unlike in problem set i), in this problem set there seem to be no differences in transferability: it does not matter the number of local optima that the optimization problems—generated with the by Rönkkönen et al. [2008] problem generator—have regarding the performance and learning capability of the hyper-heuristic. Consequently, the transferability is very similar in all problems.

In conclusion, the proposed methodology identified that there was no adaptation to the number of local optima. We can deduce this from the figure as most of the values are similar and are close to 0.5. Measuring the magnitude of the transferability (small in this case) is not trivial. Specifically, we were able to observe this result because of the specific definition of the transferability considered: including the averaging of repeated samples and the use of ranks instead of the objective function directly (explained in detail in Appendix 7.3.5).

Problem set iii)

In Figure 4.8, we show the transferability matrix for the third iii) problem set. In this problem set, we have four types of optimization problems (LOP, PFSP, QAP, and TSP) and eight problems instances for each problem type. From now on, "QAP" will be used to refer to any one of the QAPs in problem set iii) (LOPs, PFSPs, and TSPs will be denoted in the same way). In addition, we will use $\neg$ QAP to refer to any one of the problems in problem set iii) that are not QAPs and PROB to refer to any one of the problems in problem set iii).

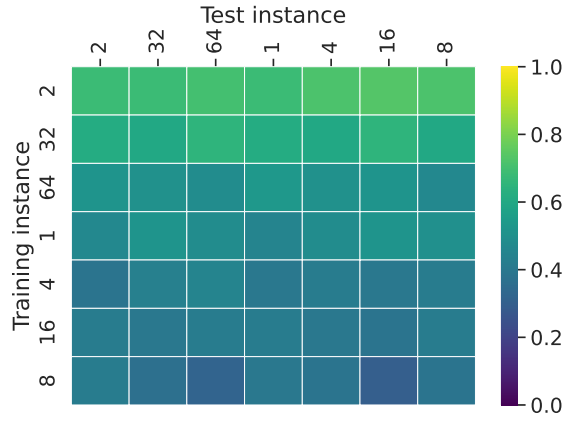


Fig. 4.7: Transferability matrix of problem set ii).

First, observe that

1) $T_{\text{QAP},\text{QAP}} < T_{\neg\,\text{QAP},\text{QAP}}$ and

2) $T_{\text{QAP},\neg\,\text{QAP}} > T_{\text{QAP},\neg\,\text{QAP}}$.

We deduce 1) from the last 8 columns and 2) from the last 8 rows in the transferability matrix (Figure 4.8). The interpretations are 1) the hyper-heuristic performs better in QAPs when it is trained in QAPs and 2) the hyper-heuristic performs worse in non-QAPs when it is trained in QAPs. The rest of the problems do not satisfy an analogous version of 1) and 2). Therefore, the QAP is the problem that is the most *different* from the rest of the problems, from the point of view of the performance of the hyper-heuristic.

Now let us consider the order of the problems in the matrix. Notice that all the QAPs and PFSPs are placed in adjacent rows/columns. However, some of the TSPs and LOPs are mixed. This means that, from the point of view of the performance of the considered hyper-heuristic, TSPs and LOPs are similar to each other (in contrast with QAPs and PFSPs).

Ceberio et al. [2015a] also reached a similar conclusion in their work with multi-start local search. Specifically, they found out that multi-start local search with the insert neighborhood [Schiavinotto and Stützle, 2007] outperforms the other two multi-start algorithms studied in the TSP and LOP, but not in the PFSP or the QAP. Note that the hyper-heuristic considered in this chapter also uses the insert operator and insert local search.

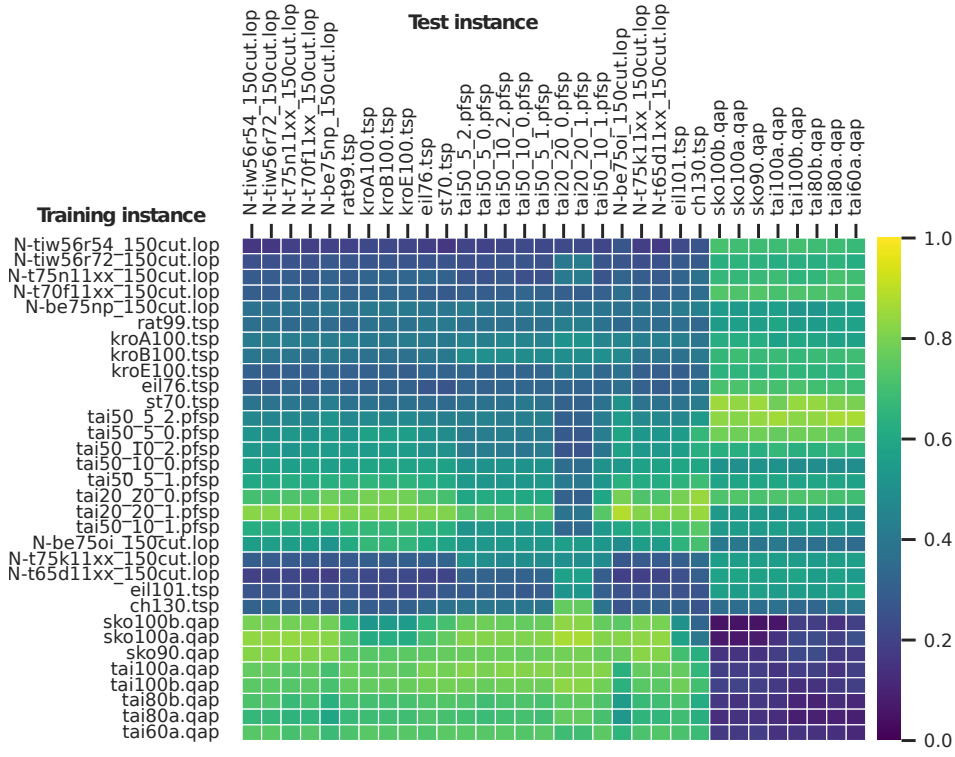


Fig. 4.8: Transferability matrix of problem set iii).

In conclusion, the transferability matrix for problem set iii) was highly correlated with the problem type. The analysis also pointed out that the performance of the hyper-heuristic is more similar in TSPs and LOPs, which is consistent with previous findings [Ceberio et al., 2015a].

Problem set iv)

In Figure 4.9, we show the transferability heatmap for problem set iv) problem set. In this problem set, we study three QAP instance types: *taixxA*, *taixxB* and *sko*. As we can see in the figure, there is a big difference between *sko* instances and the rest (rows/columns 1-7). The difference in transferability between *taixxA*, *taixxB* instances is not appreciable visually.

However, regarding the order, *sko* instances are all next to each other, while *taixxA* and *taixxB* instances are also fairly separated from each other but problems *taixxB_14* and *taixxA_5* are out of place. As described in Section 4.3.2, columns and rows were jointly reordered such that the difference between adjacent rows/-

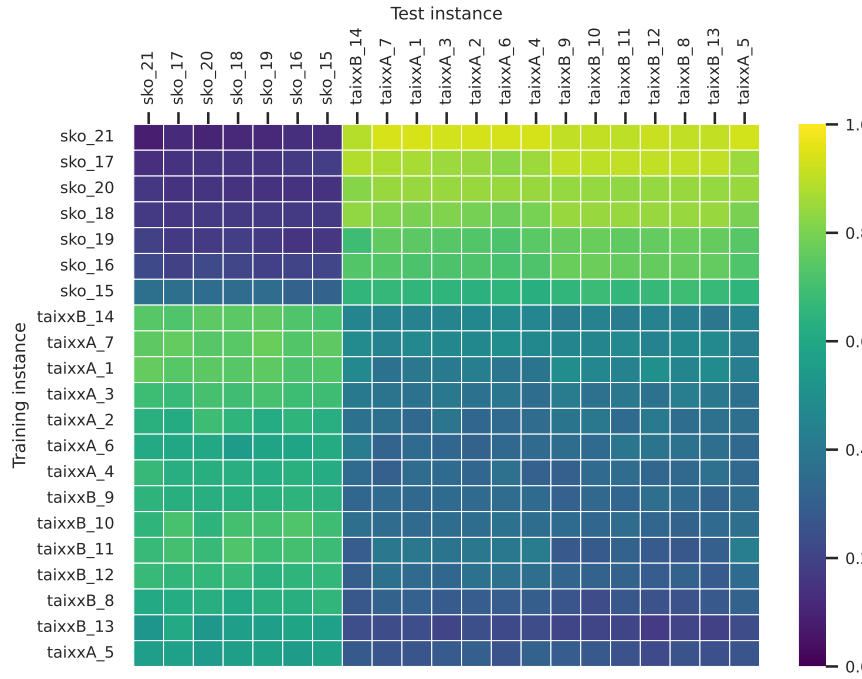


Fig. 4.9: Transferability matrix of problem set iv).

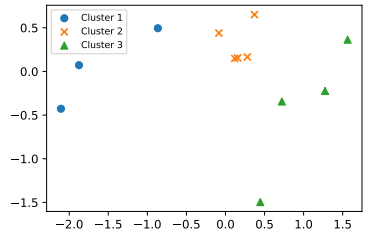
columns is as small as possible (minimizing the loss function in Equation (4.2)). Consequently, adjacent rows/columns are similar to each other are placed together. This suggests that the transferability of *taixxA* and *taixxB* is actually different, even though visually the transferability looks exactly the same.

In conclusion, the proposed analysis clearly indicates a difference between *sko* instances and the rest, and also suggests a smaller difference between *taixxA* and *taixxB* instances.

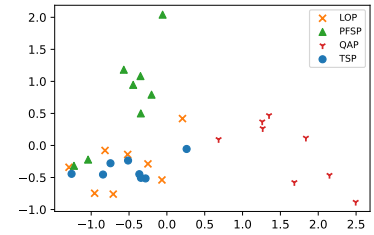
4.4.4.2 Via the behavior of the hyper-heuristic

In this second part, we analyze the optimization problems with the LDA of the problem sets, generated via the behavior of the hyper-heuristic following the methodology introduced in Section 4.4.4.2. The experimentation is carried out in three out of the four problem sets. We skipped problem set ii) because, as we saw in the previous part of the experimentation, the hyper-heuristic performs the

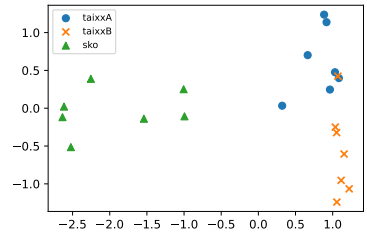
same in these optimization problems, hence looking at the behavior makes no sense in this case.



(a) problem set i)



(b) problem set iii)



(c) problem set iv)

Fig. 4.10: The LDA for problem sets i), iii) and iv). The colors represent the problem types/clusters within the problem sets. These were added after the LDA was computed. If two optimization problems are close to each other in the LDA, the interpretation is that the hyper-heuristic has a similar behavior—*how* it performs optimization—in these two problems.

Problem set i)

Figure 4.10a shows the LDA of problem set i). We colored the data points after generating the LDA based on the three problem clusters that we defined in the previous section. This means that the cluster to which the instances belong where not used to fit the LDA. As seen in the figure, these three clusters are linearly separable. This means that for problem set i) there is a high correlation between the two analysis methods carried out. The results validate the conclusions drawn in the previous part of the experimentation (Section 4.4.4.1) on the same problem set: both embeddings are correlated with the properties of the optimization problems.

Problem set iii)

Figure 4.10b shows the LDA for problem set iii). Firstly, notice that all of the QAPs are linearly separable from the rest. Also, most of the PFSPs are clustered together, except for two of them. Finally, the LOPs and TSPs are mixed together, although it seems that LOPs are somehow surrounding the TSPs.

In the transferability matrix for this problem set, we also observed that the QAP problems were the most different from the rest of the problems and that LOP and TSP problems were similar to each other. Hence, both analysis methodologies point out this similarity between LOP and TSP problems.

Problem set iv)

Figure 4.10c shows the LDA for problem set iv). The three types of instances are quite visibly separated from each other. It could be argued that *taixxA* and *taixxB* are less separated from each other as they are located on the same vertical axis, and *sko* instances are further away in the horizontal axis. Similarly, in the transferability matrix of this problem set, *sko* instances were very dissimilar to the rest, while *taixxA* and *taixxB* were comparatively less dissimilar to each other regarding transferability.

4.5 Conclusion and future work

In this chapter, we proposed a multi-domain problem analysis method. Specifically, we proposed two embeddings into the real space for a set of problems: one based on the performance of the hyper-heuristic and another one based on its behavior. The two methods are domain agnostic: they are not specific to the search space and only require the evaluation of the objective function. In an experimental study on four problem sets, we showed that the two analysis methods are useful to gain new insight on a set of problems. We also showed that the embeddings are correlated with the properties of the optimization problems.

The code to reproduce the experiments and apply the proposed methodology is available in our GitHub repo at https://github.com/EtorArza/TransfHH. In future work, it would be interesting to offer the proposed analysis methods in a python library. This would make it as easier for practitioners to use the proposed framework to analyze other optimization problems.

Supplementary Material

Code to Reproduce the Results and Apply the Methodology

The code to reproduce the results in this chapter are available in the repository https://github.com/EtorArza/TransfHH. This repository also contains a demo get started with the proposed methodology.

5

Generalized Early Stopping in Evolutionary Policy Learning

Evaluating solutions in optimization problems can be time-consuming, for instance, when evaluating robots in the real world. During the evaluation process, it is often possible to predict the poor quality of a solution without waiting for its completion (for example when a two wheeled robot continuously spins on the spot). In such cases, it makes sense to stop the evaluation of a solution early to save computation time, which is known as early stopping. Note that it is not always possible to stop the evaluation early, instead it depends on the objective function. Certain computer simulations can be stopped mid evaluation, which is why early stopping is a popular technique in policy learning.

However, most early stopping approaches in policy learning are problem specific and need to be specifically designed for the task at hand. In this chapter, we propose an early stopping method for policy learning that only looks at the objective value at each time step and requires no problem specific knowledge. The evaluation of the current solution candidate is stopped when it performs worse than the best found solution for a given episode length. We test the introduced stopping criterion in five policy learning environments drawn from games, robotics and classic control domains, and show that it can save up to 75% of the computation time while obtaining solutions of a similar quality. We show that the proposed general solution achieves time savings that are comparable to those attained by problem-specific time-saving approaches.

5.1 Introduction

Evolutionary algorithms (EAs) are increasingly being using in applications such as computer games [De Souza, 2014, Hastings et al., 2009] and robotics [Hoffmann, Sept./2001, Fleming and Purshouse, 2002] to learn control algorithms (policies), as well as being applied to classic control tasks such as the benchmark suites available in OpenAi Gym [Brockman et al., 2016]. Often direct policy search algorithms such as EAs or reinforcement learning based approaches require a large number of evaluations: when these evaluations are costly in terms of time, this can result in extremely long learning times, which can be prohibitive in the worst case. Unfortunately many applications of interest suffer from this problem. For example, the protein folding problem [Dill et al., 2008] requires costly simulations, while applications that involve a *double* optimization process are also considered very computationally costly. This includes for example the joint optimization of robot morphology and control [Le Goff et al., 2021, Hart and Le Goff, 2022] in simulation (which typically use an outer loop to evolve body-plans and a nested inner-loop to evolve control), nested combinatorial optimization problems[Wu et al., 2021, Kobeaga et al., 2021] or hyperparameter optimization [de Souza et al., 2022]. Specifically in robotics, evaluations that need to be conducted directly on a physical robot to avoid any reality-gap tend to be very time-consuming, while repeating lengthy evaluations also places considerable wear and tear on machinery, potentially leading to unreliable objective-function values.

One approach to reducing the computational burden posed by expensive evaluation functions is to use a surrogate model [Hwang and Martins, 2018, Ranftl and von der Linden, 2021]. Surrogate models try to replace the costly objective function with a cheaper alternative, that is usually less accurate but faster to compute [Alizadeh et al., 2020]. This saves computation time because the number of function evaluations of the costly objective functions is reduced. However, selecting a suitable surrogate model can be challenging, typically involving the need to determine an appropriate trade-off between size (i.e. how much information is necessary to compute the surrogate model), the accuracy required, and computational effort (the time required for the surrogate modelling process itself) [Alizadeh et al., 2020] which then influences the choice of surrogate model.

Instead of reducing the number of evaluations, it also possible to save computation time in these types of problems by stopping the evaluation of non promising solutions early. With this approach, given a fixed time budget in which to conduct evaluations which each have a maximum budget of n seconds, it is possible to compute more evaluations than if every potential evaluation is run for exactly n seconds. This is known as early stopping [Hutter et al., 2019, Li et al., 2017] or capping [de Souza et al., 2022, Hutter et al., 2009]. Several early stopping approaches have been proposed for hyperparameter optimization, including *irace* [de Souza

et al., 2022, López-Ibáñez et al., 2016], *sequential halving* [Karnin et al., 2013] and hyperband [Li et al., 2017].

Early stopping has also been considered in the context of policy learning. For example, when learning to control a robot in a simulation, if the robot gets stuck (it does not move) it is useful to stop the evaluation early [Le Goff et al., 2021]. However, there are two limitations associated with these problem specific methods. First, these approaches, in some cases, can fail to stop the evaluation even if it is clear that additional time is not going to improve the objective value. For example, when a two wheeled robot continuously spins on the spot, it would still register as moving, but it is completely useless to continue evaluating. Secondly, they require problem specific knowledge, such as detecting when the robot is not moving, which might not always be trivial (for example when dealing with robots in the real world).

The main contribution of this chapter is to show that generic early stopping is applicable to direct policy search via evolutionary algorithms. Similar to early stopping methods for hyperparameter optimization [Hutter et al., 2019, Li et al., 2017, de Souza et al., 2022, Karnin et al., 2013] and unlike current early stopping criteria for policy learning, the proposed approach only needs the objective value to decide when to stop the evaluation of the robots. We demonstrate both the efficacy and generality of the method in a wide-ranging experimental section in five different direct policy search environments, showing that the proposed approach significantly reduces the optimization time of policy learning algorithms in a wide variety of control tasks.

The chapter is organized as follows. We first discuss some related work to position the proposed method in the literature. In the next section, we provide a formal definition of the problem and introduce the proposed early stopping method. In Section 5.4 we present the five part experimental study on the applicability of the proposed method in direct policy search tasks. Finally, Section 5.5 concludes the article.

5.2 Related work

Many reinforcement learning environments in the literature use problem specific early stopping methods to save computation time. One of the earliest examples is probably the CartPole control problem [Barto et al., 1983]. In this problem, a cart needs to balance a pole by moving left or right (see the animation on OpenAI's website). In the original implementation by Sutton [1984], the evaluation is usually stopped after 10^5 steps (episode length), but is terminated early when the pole is considered not balanced for 100 steps. In the modern version of the same problem

by OpenAI, the episode length is 500 and the evaluation is terminated early when the pole is not considered balanced, or the cart has moved too much to one of the sides.

Another more modern example with early stopping is the Ant control task in the MuJoCo environment in OpenAI gym [Brockman et al., 2016]. In this task, the control of an ant needs to be learned, such that the traveled distance is maximized, while minimizing the energy expenditure and the contact force (see the animation). In this task, the episode length is 1000 steps, but an evaluation is stopped early if the vertical position of the torso is not in the interval $[0.2, 1.0]$, or if there are numerical errors in the simulator.

A limitation of the early stopping criteria of these and other policy learning tasks is that they are highly problem specific. They need to be carefully designed, taking into account the problem at hand. For example, in the previous *ant* example, choosing to stop the evaluation based on the position of the torso is not trivial, and requires understanding of what is a desirable position of the torso.

There has been plenty of work in early stopping based on the objective function alone, but most of it has focused on hyperparameter optimization. Some of the best known early stopping approaches for hyperparameter optimization are the *i-race* algorithm [de Souza et al., 2022, López-Ibáñez et al., 2016], the sequential halving algorithm [Hutter et al., 2019, Karnin et al., 2013] and hyperband [Li et al., 2017, Falkner et al., 2018b]. Early stopping approaches work very well in hyperparameter optimization because the evaluation of solutions can be paused and resumed. Consequently, it is possible to evaluate a set of solutions simultaneously and compare their partial objective values with one other, discarding poorly performing candidates before continuing the evaluations. This is not always possible in direct policy search tasks, especially those that run in the real-world. For example, it would not make sense to pause the evaluation of a controller in a real-world robot, evaluate a different controller in the same robot, and then resume the previous evaluation. The same applies to simulation, where it is not trivial to implement a way to save the state of the simulation and load it later.

More recently, de Souza et al. [2022] proposed an early stopping method that only takes into account the objective function and does not assume that the evaluation of solutions can be resumed. However, the approach was only tested in the context of hyperparameter optimization.

By making certain assumptions on the objective function and considering $1 + \lambda$ *evolution strategies*, Bongard [2011, 2010] proposed an early stopping mechanism for multi-objective evolution of robots, based on the objective function. The approach involves stopping the evaluation of candidates once it is impossible for them to beat the best found candidate in the current generation. This early stopping approach has the very good property of not changing the final outcome while

still saving computation time. However, the applicability of Bongard's method in the general case is very limited, as it depends on both a specific definition of the objective function and the use of the $1 + \lambda$ *evolution strategies* algorithm.[1]

In summary, the early stopping approaches that have been tested in policy learning either require problem specific knowledge, or a specific definition of the objective function and learning algorithm. This motivates our proposal which addresses both these issues.

5.3 Generalized Early Stopping for Episodic Policy Learning (GESP)

We propose a simple early stopping method for direct policy search tasks that overcomes the limitations discussed in the previous section. The proposed method is general and makes no assumptions about the optimization problem or the objective function or the learning algorithm. It uses the output of the objective function without the need of problem specific information. Specifically, the proposed approach is applicable to the optimization problem defined as follows:

Definition 15 *Given a maximum computation time budget t_{max}, f an objective function and S a solution space, we define a optimization problem* $\text{argmax}_{\sigma \in S} f(\sigma)$ *with these four additional properties:*

1. *Computing the objective function of a solution $f(\sigma)$ has a time cost T.*
2. *Instead of computing $f(\sigma)$ exactly, it can be approximated for any lower time cost $t < T$. We denote the approximation in time t as $f[t](\sigma)$.*
3. *A solution σ has to be chosen as the best before the computation budget t_{max} is spent. Only solutions evaluated for time T are eligible to be chosen as the best.*
4. *Approximate evaluations cannot be resumed.*[2]

[1] The objective function to be maximized needs to be a combination of other sub-objective functions in different sub-tasks, assuming that the sub-objective values are always negative. With this objective function, it is not necessary to evaluate the solution in every sub-task, as the objective value can only decrease with further evaluations. Consequently, once the objective value of a solution is lower than the best candidate in the current population, the evaluation can be stopped, as it is guaranteed that it will not outperform the best candidate. When considering this early stopping criterion in combination with the use of the $1 + \lambda$ *evolution strategies* algorithm, the same final solution is obtained while saving computation time.

[2] For example, let σ_1, σ_2 be two solutions and t_1, t_2, t_3 three time costs with $t_3 > t_1$. Then evaluating $f[t_1](\sigma_1)$, $f[t_2](\sigma_2)$, $f[t_3](\sigma_1)$ in that order has time cost $t_1 + t_2 + t_3$,

Assuming a maximization optimization problem, GESP involves stopping the evaluation of a solution at time step $t > t_{grace}$ when certain conditions are met. Specifically, we stop the evaluation of a solution σ at time step t if Equation (5.1) below is satisfied.

$$\max\{f[t](\sigma), f[t - t_{grace}](\sigma)\} < \min\{f[t](\sigma_{best}), f[t - t_{grace}](\sigma_{best})\} \tag{5.1}$$

The grace period parameter t_{grace} is a parameter that establishes a minimum time for which all candidates will be initially evaluated regardless of their objective value. In addition, it determines the bonus evaluation time given to new candidate solutions. The evaluation of the new candidate solution σ is stopped when its objective value at time step t is worse than the objective value of the best solution σ_{best} at time $t - t_{grace}$. This gives new candidates t_{grace} extra time steps to achieve the level of performance of the current best solution

In Algorithm 6 we show the pseudocode of GESP. First we initialize the reference objective values to $-\infty$ (lines 1-2). Then, starting with the first time step $t = 1$, we evaluate the approximate objective function $f[t](\sigma)$ at that time step (line 4). Then, if $t > t_{grace}$ and Equation (5.1) are satisfied (line 5), the objective function of σ is approximated as $f[t](\sigma)$ (line 6), σ is not evaluated in time step $t + 1$ and beyond, and the reference objective values are not updated. Otherwise, we evaluate σ at time step $t + 1$. Finally, if σ makes it to time step T and a new best objective value is found $f[T](\sigma_i) > f[T](\sigma_{best})$, we replace the reference objective values with the new ones (lines 7-8).

Our approach is similar to de Souza et al. [2022]'s early stopping methods for hyperparameter optimization. de Souza et al. [2022]'s approach generates a stopping criterion based on a set of already evaluated solutions, such that the evaluation of new candidate solutions can be stopped early when they perform relatively poorly. In contrast, our approach only requires the best found solution so far to decide when to stop the evaluation of new candidates. Consequently, the proposed approach can be applied once the first solution has been evaluated. In addition, while de Souza et al. [2022]'s work was designed for hyperparameter optimization, we show that our approach is applicable to direct policy search.

5.3.1 Applicability on direct policy search

Early stopping through the objective function alone usually assumes a monotone increasing objective function, which is true for hyperparamter tuning. However,

but evaluating them in the order $f[t_1](\sigma_1)$, $f[t_3](\sigma_1)$, $f[t_2](\sigma_2)$ has a lower time cost of $t_3 + t_2$.

Algorithm 6: Evaluate solution with early stopping

Input:
$f[t](\cdot)$: The approximation of the objective function with time t. $f[T](\cdot)$ is the objective function when no early stopping is considered.
σ: The solution to be evaluated.
t_{grace}: The grace period parameter.
T: The maximum evaluation time.

```
if f[t](σ_best) = φ then
    f[t](σ_best) ← −∞,   ∀t = 0, ..., T
for t = 0, ..., T do
    compute f[t](σ_i)
    if t > t_grace and Equation (5.1) is satisfied then
        return f[t](σ_i)
if f[T](σ) > f[T](σ_best) then
    f[t](σ_best) ← f[t](σ),   ∀t = 0, ..., T
return f[T](σ)
```

this assumption does not hold for direct policy search. Specifically, early stopping without a monotone increasing objective function creates two issues: and we propose two possible modifications of GESP to overcome them.

We motivate and explain these two issues with an example. Let us consider the policy learning pendulum task. The reward in this task is inversely proportional to the speed and the angle of the pendulum (assuming the angle at the upright position is 0). The goal in this task is to maximize the sum of all the rewards in T time steps, where the reward in each step is in the interval $(0, -16.27)$. This means that in each time step, the objective function can only be lower than in the previous time step (the objective function is monotone decreasing). Consequently, if early stopping is applied in this problem, the solutions that are stopped earlier will have a better objective function than if they had been evaluated for the maximum time T.

Issue (1) involves correctly reporting the best found solution. Unless we assume a monotone increasing objective function, the best found solution might be a partially evaluated solution. Consequently, if the issue is not addressed, the reported objective value could potentially be better than the actual objective value. For example, in the *pendulum* task, if a policy applies no torque then the pendulum does not move and the evaluation is stopped early. This is a very poor policy, but since it triggers the early stopping very quickly, it obtains a better objective value than a policy that performs well and was evaluated for the entire episode. Hence, without taking into account Issue (1), the best reported solution could be one that immediately triggers the early stopping criterion. Overcoming this issue is simple

with Modification (1): do not update the best found solution *unless* the new best solution has been evaluated for the maximum time T.

Issue (2) is not as critical as Issue (1), and is related to the credit assignment during the optimization process. In monotone increasing problems (such as hyperparameter optimization), a poor solution that is early stopped might get a worse than deserved objective value, as with additional evaluation time it might have been able to increase its objective value. This is usually not considered a problem, as it is not expected that the learning algorithm is impacted in a negative way. Basically early stopping in this case favors solutions that quickly converge towards good objective values over those that are slow to converge. However, in problems with decreasing objective functions (such as the *pendulum* task), poor solutions might be assigned an objective value (due to early stopping) that is in fact better than the objective value that would have been obtained if the evaluation was continued for the maximum evaluation time. This might pose a challenge for the optimization algorithm. For example, in the *pendulum* task, if a policy applies no torque then the pendulum does not move and the evaluation is stopped early. This is a very poor policy, but since it triggers the early stopping very quickly, it obtains a better objective value than a policy that is able to correctly balance the pole and is evaluated for the entire episode. This is because in the *pendulum* task, the reward is negative in each time step, and the total reward of the policy that optimally balances the pole has many time steps with a negative reward (during which the pole is being moved towards the balancing point). The learning algorithm might therefore optimize the policy to trigger the early stopping as quickly as possible, which is obviously undesirable.

It is possible to overcome Issue (2) by modifying the objective function. It is enough to redefine the objective function such that it is monotone increasing. To achieve this, we can add a constant value k to the objective function in each time step, ensuring that the redefined objective function is monotone increasing. For instance, in the pendulum policy learning task (with a reward in the interval $(0, -16.27)$ in each time step), it is sufficient to redefine the objective function as $f_{new}[t](\sigma) = f[t](\sigma) + t \cdot 16.28$. By adding Modification (2), we also overcome Issue (1).

However, as Issue (2) does not produce an incorrect result (it might only potentially set back the optimization), we chose not to redefine the objective function because we want GESP to be as non-invasive as possible. Thus, we propose GESP as a plug and play method that is compatible with as many problems as possible and requires no modifications in the objective function. Therefore, in the experimentation of this chapter, we ignore Issue (2) and only consider Modification (1). Even with only Modification (1), we show that GESP rarely decreases the performance and significantly speeds up the search process in a wide variety of tasks

(despite Issue (2)), even in tasks with monotone decreasing objective functions such as *pendulum* (Section 5.4.1).

5.4 Experimentation

To validate the proposed approach, we run several experiments in different direct policy search tasks with different evolutionary learning algorithms to demonstrate the benefit of using GESP. We chose different tasks with different learning algorithms to show that GESP is applicable in different scenarios and across a range of evolutionary algorithms. In some of these tasks, the original authors have proposed a problem specific stopping criterion to save computation time, in which case, we also compare GESP to the problem specific approach. The experimentation is carried out in five different environments, each with different properties and learning algorithms. Table 5.1 lists the environments considered.

Name	Environment	Learning algorithm	Task
classic control	Classic control	CMA-ES [Igel et al., 2006]	*cartpole* and *pendulum*.
super mario	Super Mario	NEAT [Stanley and Miikkulainen, 2002]	Move the character "Mario" to the right as much as possible.
mujoco	Mujo-co	CMA-ES [Igel et al., 2006]	*half cheetah*, *inverted double pendulum*, *swimmer*, *ant*, *hopper* and *walker2d*.
NIPES explore	8 x 8 grid with obstacles	NIPES [Le Goff et al., 2020]	Move a robot with four wheels and two sensors and visit as many squares possible in 30 seconds.
L-System	Based on OpenAI gym	$\lambda + \mu$ ES with L-System encoding [Veenstra and Glette, 2020]	Learn the morphology and control of virtual creatures and move to the right as much as possible.

Table 5.1: Environments in the experimentation

Methodology

For each experiment, we record the best objective value found so far with respect to the computation time (known as the *attainment trajectory* [Dreo and López-Ibáñez, 2021]). Each experiment is repeated 30 times, and the median and interquantile ranges are reported. When comparing two attainment trajectories,

we perform a pointwise two sided Mann-Whitney test[Mann and Whitney, 1947, Arza et al., 2022] with $\alpha = 0.01$. The test is performed pointwise with no familywise error correction [Korpela et al., 2014]. Hence, if H_0 is true and the two distributions are the same, *on average*, we expect to find a statistically significantly difference in 1% of the length of the attainment trajectory, when in reality, there is no difference [Härdle et al., 2004].[1]

5.4.1 Classic control tasks (classic control)

OpenAI Gym [Brockman et al., 2016] is a framework to study reinforcement learning. Among the environments available in OpenAI Gym, we have the classic control tasks *cart pole* [Barto et al., 1983] and *pendulum*. In the garage framework, there is an example script (also archived) that uses CMA-ES to learn the policy for the *cart pole* task. The same learning algorithm was considered for *pendulum*. We set the grace period parameter to 20% of the maximum time: $t_{grace} = 0.2 \cdot T$.

The objective function in *cart pole* is the number of timesteps before it is terminated because out of bounds or because the pole is no longer in the upright position (the reward is 1 in every time step). Consequently, any approximation of the objective function that has not yet been terminated is $f_{cartpole}[t](\sigma) = t$. As mentioned before, pendulum has a monotone decreasing objective function with a reward in the interval $(0, -16.27)$ in each time step. The policies are learned with CMA-ES.

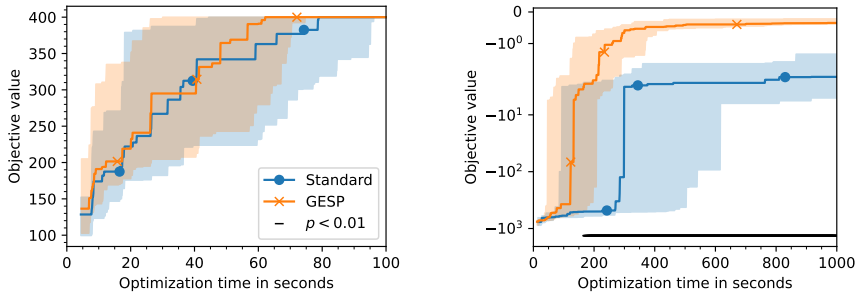
Let us first consider the *cart pole* experiment. Due to the definition of the objective function, applying GESP has no effect in this case: the condition in Equation (5.1) will never be satisfied, and no early stopping will happen. Consequently, we expect that there is no difference experimentally. In fact, since applying GESP has no effect, the null hypothesis is true for this task.

The result for the *cart pole* experiment shown in Figure 5.1a confirms this experimentally. Visually, there is no difference between the two stopping criteria, also suggested by the lack of statistical significant difference of the two sided Mann-Whitney test at $\alpha = 0.01$.

To confirm this hypothesis, we computed the ratio of solutions evaluated with and without GESP. For instance, a ratio is 2 indicates that using GESP, twice as many solutions are evaluated in the same amount of computation time. The ratio of solutions evaluated for the cart-pole is 1, as shown in Figure 5.2, which means that the same amount of solutions are being evaluated with or without GESP.

[1] As with every hypothesis test (frequentist) approach, *on average* in this context means if we were to repeat this same experiment many times [Conover, 1980].

The results for the *pendulum* task are shown in Figure 5.1b. With GESP, a significant amount of time is saved in this task and a final higher objective value is obtained. For instance, with GESP, the median time to reach an objective value of -10 is less than 150 seconds using GESP, compared to more than 300 seconds without.



(a) *cart pole* (b) *pendulum*

Fig. 5.1: The objective value of the agents with and without GESP with respect to computation time (**classic control**).

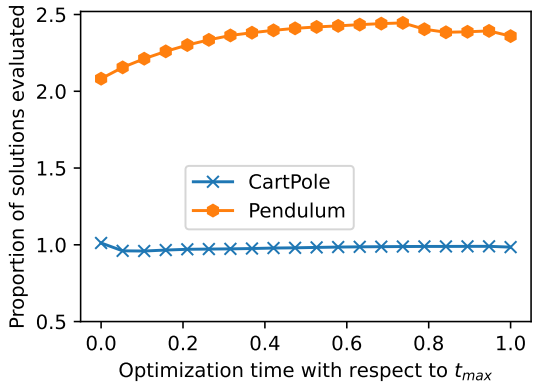


Fig. 5.2: Ratio of solutions evaluated with and without GESP in the same optimization time. A higher value indicates that GESP was able to evaluate more solutions in the same time.

5.4.2 Playing Super Mario (super mario)

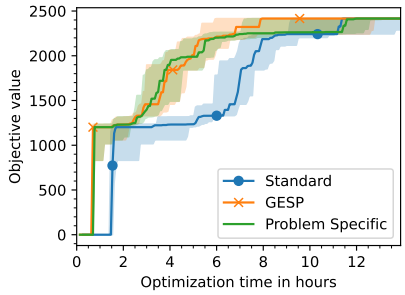
Verma [2020] proposed learning to play the video game "Super Mario" released in 1985 with NEAT [Stanley and Miikkulainen, 2002]. In this popular video game, a character Mario needs to move to the right without dying and reach the end of the level. In Verma [2020]'s implementation, the objective function is the horizontal distance that the character has moved.

A maximum episode length of 1000 steps is considered, although the episode also ends if the character dies. Verma implemented an additional stopping criterion: if the character does not move horizontally in 50 consecutive steps, then the episode also ends. In the following, we compare this problem specific stopping criterion with GESP, and we also consider no stopping criterion as a baseline. We set $t_{grace} = 50$ for a fair comparison with Verma's problem specific termination criterion. We trained the algorithm in the levels 1-4, 2-1, 4-1, 4-2, 5-1, 6-2 and 6-4. We show the results in Figure 5.3.

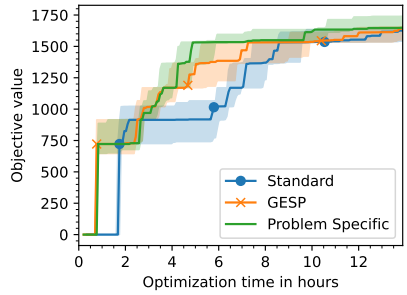
In general, the results demonstrate that there is no big difference in performance between the problem specific method and GESP. In some of the levels, the problem specific approach performs better, but this difference disappears as the computation time increases. Both methods are clearly better than using no stopping criteria.

Curiously enough, in level 5-1, there is no difference between the results obtained using either of the stopping methods and the baseline method that does not use any stopping criterion. The reason is probably that in this level, it is hard to get stuck: there are a large number of enemies and few obstacles. We hypothesize that in this level, it is very easy to die, hence the execution is terminated regardless of the other stopping criteria (and therefore GESP have no effect).

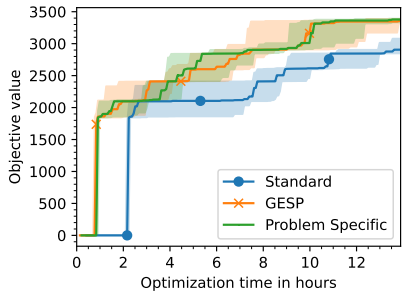
To validate this the hypothesis, we computed the ratio of solutions evaluated in the same optimization time with and without GESP. We compute this ratio for all of the **super mario** levels, and we show the result in Figure 5.4. As can be seen in the figure, in level 5-1 the ratio is almost 1, indicating that GESP rarely stops the evaluation of the agents early in this level. This finding validates the previous hypothesis.



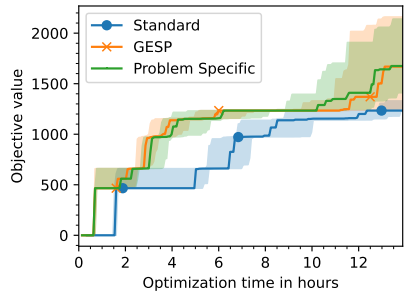
(a) Level 1-4.



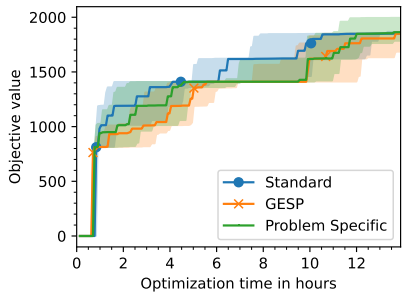
(b) Level 2-1.



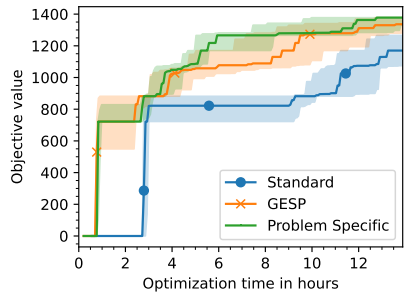
(c) Level 4-1.



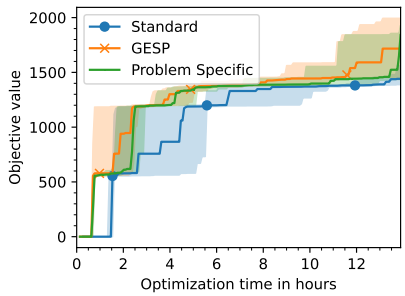
(d) Level 4-2.



(e) Level 5-1.



(f) Level 6-2.



(g) Level 6-4.

Fig. 5.3: The objective value of the agents with respect to computation time in **super mario** with GESP, with the problem specific stopping criterion and without additional stopping criterion.

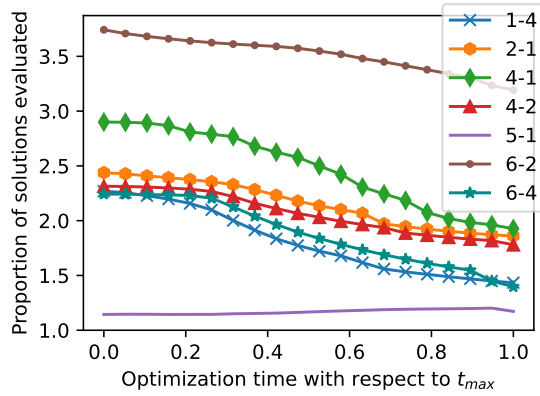


Fig. 5.4: Ratio of solutions evaluated with and without GESP in the same optimization time. A higher value indicates that GESP was able to evaluate more solutions in the same time.

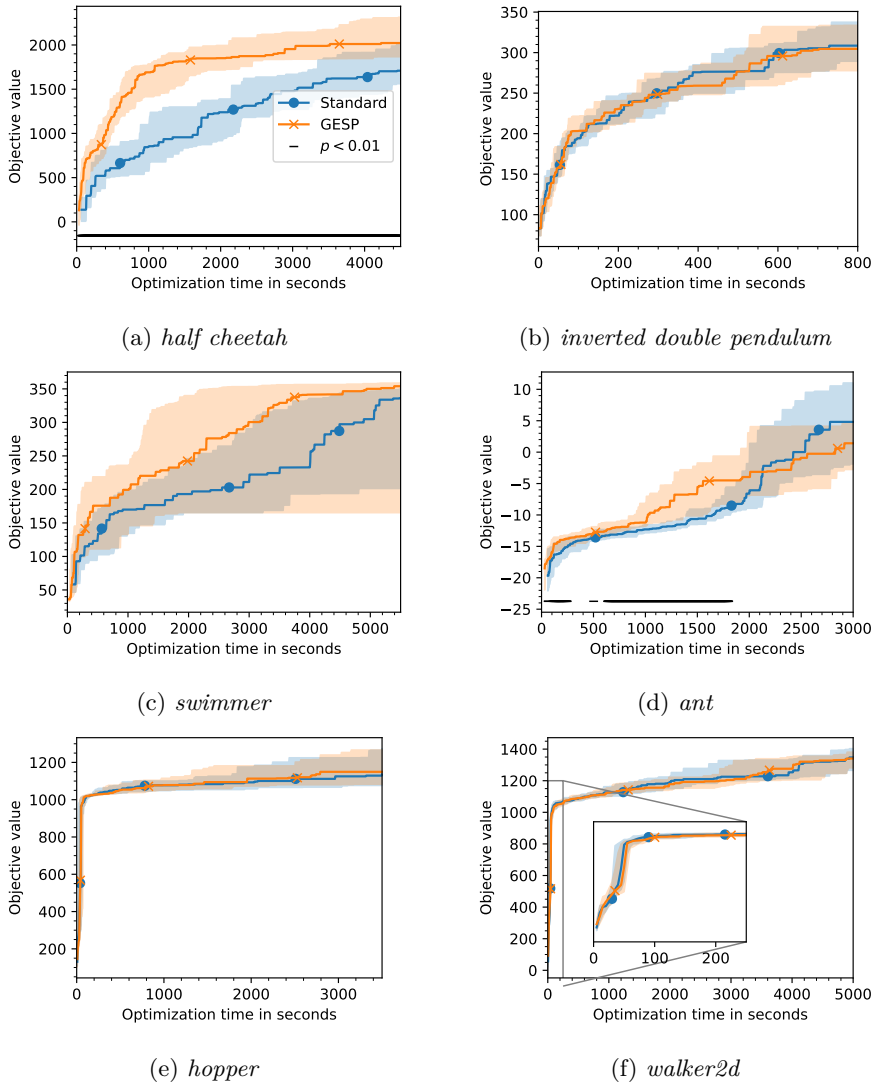
5.4.3 Tasks in the Mujoco environment (mujoco)

Mujoco is a high performance physics simulator. OpenAI Gym has some tasks defined in this environment which have been extensively used in reinforcement learning research. In this section of the experimentation, we experiment with the tasks *half cheetah*, *swimmer*, *ant*, *hopper*, *walker2d*. In all of these tasks, the objective is to move the agent as far as possible from the initial position, while minimizing energy use. We also consider the *inverted double pendulum*, in which the objective is to keep a double pendulum balanced.

In all of these tasks, the policies are learned with CMA-ES [Igel et al., 2006] (with the same algorithm that was considered in the classic control tasks in Section 5.4.1) and we set the grace period parameter to 20% of the maximum time: $t_{grace} = 0.2 \cdot T$.

The results are shown in Figure 5.5. In the *inverted double pendulum*, *hopper* and *walker2d* tasks, applying GESP made no difference in the performance and computation time. In the other three tasks, by using GESP we are able to get a better objective value in the same amount of time, although the difference was only statistically significant in *ant* (until 1600 seconds) and *half cheetah* tasks.

The tasks *ant*, *hopper*, *walker2d* and *inverted double pendulum* have problem specific stopping criterion that stop the evaluation when the state of the agent is 'unhealthy'. The definition of "healthy agent" is different for each problem: for example, in the *ant* task, an agent is considered healthy if all the state spaces



(a) *half cheetah*

(b) *inverted double pendulum*

(c) *swimmer*

(d) *ant*

(e) *hopper*

(f) *walker2d*

Fig. 5.5: The objective value of the agents with respect to computation time in the **mujoco** tasks with and without GESP.

are finite and the distance from the body of the ant to the floor is in the interval $[0.2, 1]$. These additional stopping criteria are essential for the learned policy to be realistic: we do not want a policy that tries to exploit the physics simulator's bugs. However, we hypothesize that these stopping criteria are already very good at avoiding wasting time in undesirable states, and consequently, GESP has little room for further improvement.

To validate this hypothesis, we repeated the experimentation for the tasks *ant*, *hopper* and *walker2d* but this time without the *terminate when unhealthy* stopping criterion enabled. We recorded the performance with respect to the optimization time with GESP enabled and disabled. The results are shown in Figure 5.7. With the *terminate when unhealthy* stopping criteria disabled, GESP is able to save a lot of computation time in these three tasks, indicating that the method would be useful if one did not have the in-depth understanding of the task required to create problem-specific stopping criteria.

We show the ratio of extra evaluations computed with GESP in Figure 5.6. In the case of *hopper* and *walker2d*, the ratio is almost 1, which means that GESP is unable to save computation time in these two tasks. However, when we disable the *terminate when unhealthy* stopping criterion, the ratio is a lot higher in these two tasks, which suggests that GESP is able to early stop under-performing solutions. These results suggest that the hypothesis above is true.

In conclusion, GESP was able to save computation time in some of the tasks in this environment, and it is specially useful when there are no problem specific stopping criteria available. However, it can also save computation time alongside existing stopping criteria, although to a lesser extent (as was the case for the *ant* task, shown in Figure 5.5d).

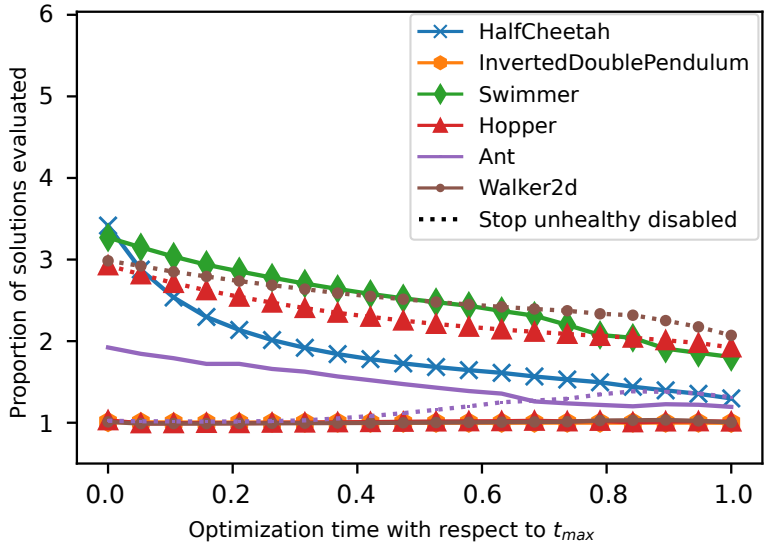
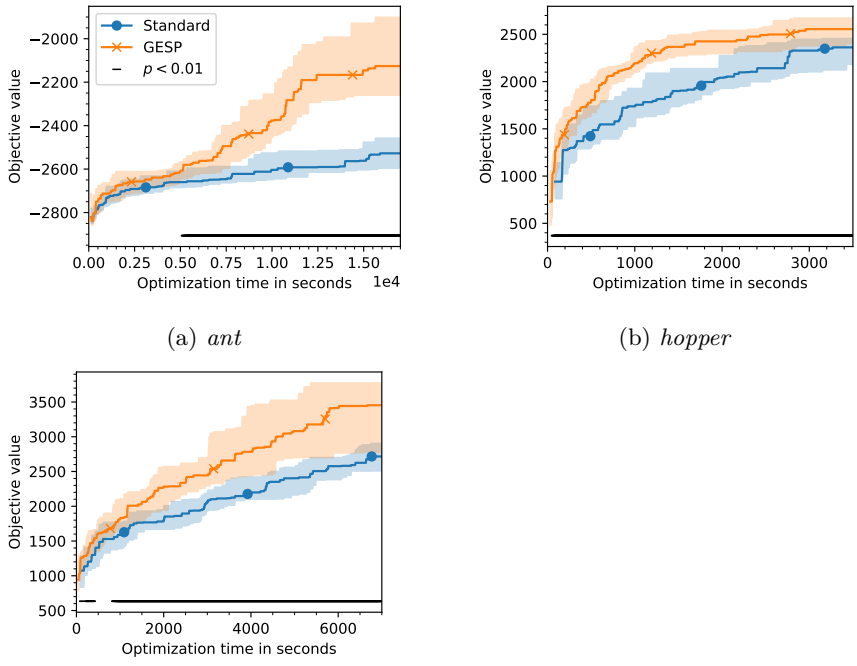


Fig. 5.6: Ratio of solutions evaluated with and without GESP in the same optimization time. A higher value indicates that GESP was able to evaluate more solutions in the same time. A dashed line indicates that the results were obtained with the *terminate when unhealthy* stopping criterion disabled.

5.4.4 NIPES within the ARE framework (NIPES explore)

Recent research in the field of Evolutionary Robotics has attempted to lay a foundation for developing frameworks that enable the autonomous design and evaluation of robots [Eiben et al., 2021]. In contrast to much previous work in Evolutionary Robotics which typically focuses only on control, recent approaches attempt to simultaneously evolve both body and control of a robot. For example, joint optimisation of body and control is accomplished in the framework known as ARE (Autonomous Robot Evolution) [Le Goff et al., 2021] using a nested architecture which uses an EA in an outer loop to evolve a body design and a learning algorithm within an inner loop to optimise its controller. Le Goff et al. [2020] proposed a learning algorithm to learn the control policy of wheeled robots in the inner loop dubbed NIPES for this purpose. The algorithm combines CMA-ES [Igel et al., 2006] and novelty search [Lehman and Stanley, 2011] in order to create a method that is a more sample and time efficient algorithm than CMA-ES alone.

We evaluate GESP using two exploration tasks proposed by Le Goff et al. [2021] in which a wheeled robot needs to explore an arena. The goal is for the robot



(a) *ant*

(b) *hopper*

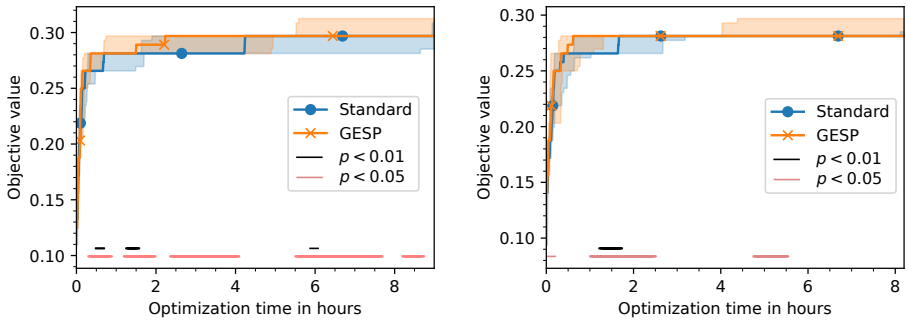
(c) *Walker2d*

Fig. 5.7: The objective value of the agents in *hopper* with and without GESP with respect to computation time, and without stopping the evaluation when the state of the agent is unhealthy.

to explore as much of an arena as possible within 30 seconds. One arena has multiple obstacles hindering exploration, while the other has a maze-like layout that requires the robot to navigate along corridors. Each arena is divided into 64 squares (see Figure 5.8), and the objective function is the proportion of squares visited.

We test NIPES with and without GESP in these two environments. We set the grace period parameter $t_{grace} = 0.2 \cdot T$ to 20% of the maximum time (30 seconds as in the work by Le Goff et al. [2021]). The results are shown in Figure 5.8.

In both environments (*obstacles* and *hard race*), GESP improves the objective value found for the same optimization time, although the difference is not statistically significant at $\alpha = 0.01$. The magnitude of the difference is not observable in the figure, and by looking at the ratio of number of evaluations with or without GESP in Figure 5.9, we can see that GESP is able to evaluate between 40% and



(a) Obstacles (b) Hard race

Fig. 5.8: The objective value of the agents with respect to computation time in the **NIPES explore** experiments, with and without GESP. The black and red horizontal lines represent that the difference is statistically significant at $\alpha = 0.01$ and $\alpha = 0.05$ with a pointwise two sided Mann-Whitney test.

70% more solutions in the same amount of time. This is less of a saving than in scenarios previously described (e.g. Mujoco, Super-Mario and the classic control environments). A higher number of repetitions (we use 30 repetitions in every experiment of this chapter) might reveal a statistically significant difference, which can also be observed at $\alpha = 0.05$ (not shown in the figure).

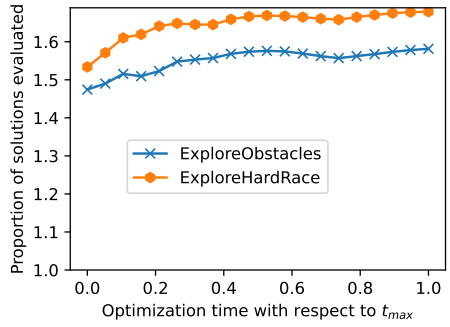
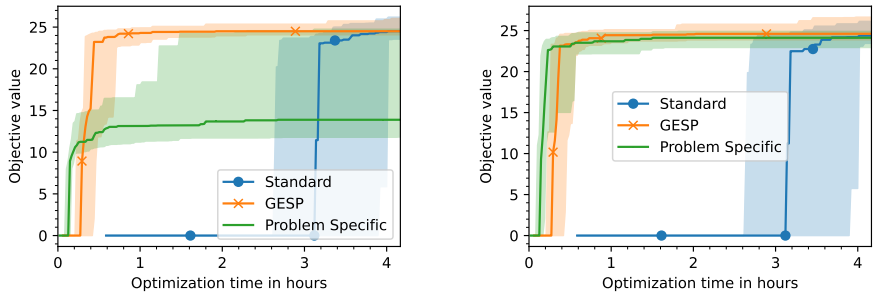


Fig. 5.9: Ratio of solutions evaluated with and without GESP in the same optimization time. A higher value indicates that GESP was able to evaluate more solutions in the same time.

5.4.5 Robotics, Evolution and Modularity (L-System)

Veenstra and Glette proposed a simulation framework to evolve the morphology and control of 2D creatures [Veenstra and Glette, 2020] based on the OpenAI gym *bipedal walker* environment. It is a gym environment for computationally cheap morphology search [Veenstra and Glette, 2020]. Agents start at the horizontal position 4.67 and need to move to the right to increase their horizontal position. They propose a problem specific stopping criterion that terminates the evaluation of the current agent if its position in time t is lower than or equal to $0.04 \cdot t$. We set the time grace parameter of GESP to $t_{grace} = 130$, which is what the amount of frames it takes for the problem specific stopping criterion to terminate randomly generated agents. It is similar to the amount of frames it takes to terminate a non moving agent at 117 frames.

The results are shown in Figure 5.10a. The problem specific approach obtains a better objective value than GESP and using no stopping criterion (from now on "Standard") for the first few hundred seconds. However, later on GESP takes over and is better than the other two approaches until 13000 seconds.



(a) Objective value during training

(b) Objective value when reevaluating

Fig. 5.10: The objective value of the agents in the task proposed Veenstra and Glette [2020] (**L-System**). We compared GESP, the problem specific stopping criterion, and no additional stopping criterion (Standard). The best found objective value is reported with respect to the total computation time. a) shows the best objective value observed during training, and b) shows the objective value of the best candidate in each generation when it is reevaluated with no stopping criteria.

At the end of the training procedure, the problem specific stopping criterion obtains a poorer objective value than the other two approaches. However, we suggest

that this is an artifact of the specific problem-specific stopping criteria suggested for this scenario: if the best agent produced with the problem specific stopping criterion enabled is evaluated without any stopping criteria, it is possible that it might in fact obtain a better performance value. On the other hand, GESP overcomes this limitation, as only solutions evaluated for the entire episode length (for time T) are candidates for the best found solution (thanks to Modification (1) introduced in Section 5.3.1).

To test whether this is the case, we repeated the experiments but reevaluating the best candidate in each generation with all the stopping criteria disabled. The results are shown in Figure 5.10b. While the performance with GESP and Standard do not change with respect to the previous experiments, the same is not true for the problem-specific stopping criterion: in this case the problem specific approach reaches a high objective value (> 20) slightly faster than GESP and considerably faster than with all the stopping criteria disabled.

The problem specific approach terminates some of the high performing agents after a while, which explains why the objective value increases more slowly for the problem specific approach without re-evaluation. Especially at the beginning of the optimization, solutions get terminated very quickly, because early agents move slowly. This makes the problem specific approach advantageous, because we waste less time on agents that can barely move, but it also means that slow moving agents that reach very far will not have a chance to be evaluated with time T. GESP is different in that promising agents will have the chance to be evaluated until time T, because the purpose is to maximize the observed objective value: we are trying to solve the problem introduced in Definition 15.

The task in this framework is to *move to the right as **far** as possible*. However, by using the problem specific stopping criterion, agents that move slower than $0.04 \cdot t$ will eventually be terminated. This means that the task to be solved changes to *move to the right as **fast** as possible*.

For the purpose of the scenarios proposed in the chapter by Veenstra and Glette [2020], we argue that the stopping criterion they proposed is still more suitable than using no stopping criterion or using GESP. The point of their paper is to compare different encoding methods, regardless of the stopping criteria. By adding a problem specific stopping criterion, they significantly sped up the learning process from about three hours of computation time to 30 minutes. With the problem specific approach they are able to evaluate between 10 times and 100 times more solutions in the same amount of time as shown in Figure 5.11. This also changed the objective function from "move as **far** as possible" to "move as **fast** as possible", however the comparison of encoding methods is also applicable to the modified version of the objective function.

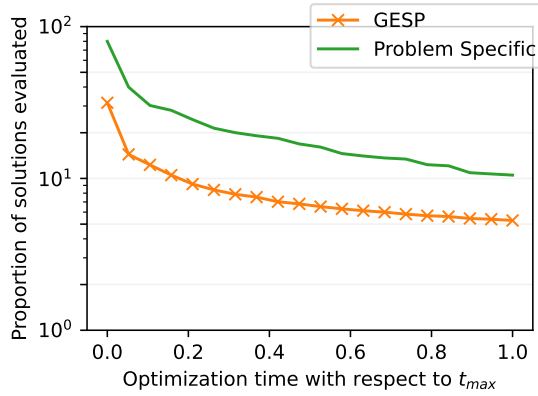


Fig. 5.11: Ratio of solutions evaluated with and without GESP and the problem specific stopping criterion in the same optimization time. A higher value indicates that with GESP (or the problem specific criterion), it was possible to evaluate more solutions in the same amount of time.

GESP could also have been considered as the stopping criterion in their study instead of the problem specific approach. GESP reduces the computation time to around 45 minutes, but unlike the problem specific approach, the objective function does not change (it is still "move as **far** as possible").

5.4.6 Discussion and future work

In the previous section, we experimented with GESP in five different direct policy search environments (see Table 5.2 for a summary of the experimentation). GESP maintained or improved the performance of the candidate solutions trained for the same amount of computation time in all the tasks considered in this chapter. In general, the biggest improvement with GESP was observed when GESP early stopped the evaluation of poorly performing candidates.

However, in some tasks, there was no improvement when applying GESP. When other terminating criteria already stopped the evaluation, GESP produced no further improvement. We observed this for level 5-1 in **super mario** and for *walker2d* and *hopper* in **mujoco**, where GESP produced no effect on the performance.

We have shown that applying GESP to direct policy search is generally beneficial. Firstly, in the experimentation carried out, GESP never made the results worse: in the worst case, it made no difference. In addition, unlike problem specific approaches, it does not require problem specific knowledge and is simpler to

	speedup with GESP*		Additional Conclusions
	first half	last half	
classic control	1 out of 2	1 out of 2	GESP has no effect in *cartpole*. GESP works in *pendulum* even though it has a monotone decreasing objective function.
super mario	6 out of 7	4 out of 7	GESP makes no improvement when the environment itself stops the evaluation early (mario touches an enemy and dies).
mujoco	2 out of 5	1 out of 5	GESP generates additional speedup when the problem specific stopping criteria are disabled.
NIPES explore	1 out of 2	1 out of 2	The speedup with GESP is small in magnitude.
L-System	1 out of 1	1 out of 1	Unlike the problem specific stopping criterion, GESP does not change the definition of the objective function.

*Number of scenarios in which GESP obtained a statistically significantly better score in the first/last half of the optimization process for the same amount of computation time.

Table 5.2: Summary of the experimental results.

implement than other approaches such as surrogate models. Moreover, it does not change the objective function (unlike for example the problem specific stopping criterion in **L-System**). This is useful if the purpose of introducing a early stopping procedure is to save computation time, while solving the same policy learning problem.

GESP has a parameter, t_{grace}, that we recommend setting to $0.2 \cdot T$ (20% of the maximum episode length), based on the experimental results in the previous section. However, we did not tune this parameter, and other values of the parameter might increase/reduce the amount of time saved. Further reduction of this parameter allows more time to be saved, but it also makes the learning algorithm more *greedy*. If we set $t_{grace} = 0$, then any candidate solution will get discarded as soon as it does worse than the best found solution in any time step. Inversely, if we set t_{grace} to 100% of the maximum episode length, then no solutions will be terminated early.

While our method is designed for direct policy search, it could also be interesting to adapt it in the future to work with other policy learning methods in which a reward is returned after each action. This is often found to be more sample efficient than direct policy search. A possible adaptation would be to stop the evaluation of the episode when the objective value is detected to be very low.

Finally, it remains to be seen whether the method can be applicable on tasks in which there is a deceptive reward, i.e. when the reward may deceptively encourage the agent to perform actions that prevent it from discovering the globally optimal behaviour, leading to convergence to a local optimum. In such cases, a policy

search algorithm must be able to select solutions with a low reward to be able to eventually reach the best solutions. A good example of such task is maze-solving (e.g. the hard maze used in the work of Lehman and Stanley [2011]) in which reward is often measured as a Euclidean distance from the end-point. On such tasks, early stopping methods (like GESP) are unlikely to work well as a poor reward can lead to early termination.

Even though GESP is unlikely to work in policy learning with deceptive rewards it might still be interesting to test it to gain more understanding with respect to how the method might be adapted in future to cope with this kind of reward. In addition, there are other potential changes that might improve the applicability of GESP in these settings. For example, novelty search [Lehman and Stanley, 2011] has shown promising results in problems with deceptive rewards, and GESP could be adapted in this context such that candidate solutions that do not show novel behaviour (and also have poor performance) are terminated early.

5.5 Conclusion

In this chapter, we introduced an early stopping method for optimization problems suitable to both increasing and decreasing objective functions, denoted as GESP. The proposed method stops the evaluation of a solution when its performance is unlikely to beat the best found solution so far. GESP is most useful for optimization problems that have costly and lengthy function evaluations. GESP stops the evaluation of the current solution when the objective value of the best found solution is better for a given time step. Unlike problem specific early stopping criteria, GESP is general and applicable to many problems: it does not use domain specific knowledge.

In a wide ranging set of experiments, we showed that adding GESP as an additional stopping criterion usually saves a significant amount of computation time in direct policy search tasks, and allows a better objective value to be found in the same computation time. Moreover, GESP did not decrease the objective value in any of the tested environments. We also compared GESP to problem specific stopping criteria, and concluded that in general, GESP had a similar performance to problem specific approaches while being more generally applicable.

Problem specific approaches can exploit domain knowledge, because the researcher implementing them might have insight into when an agent is wasting time. This often makes them very efficient. GESP, on the other hand, does not require domain knowledge and is applicable 'out of the box' to many problems. We argue that GESP is a useful early stopping mechanism applicable to problems that have no problem specific early stopping approaches. Moreover, it can also be introduced

in addition to problem specific approaches. We have evaluated the method in the context of a number of classic control problems and in a robotics domain, however, there are obvious opportunities to extend the approach to other domains which have an expensive objective function, for example optimisation of production processes [Chen et al., 2021].

Supplementary Material

Code to Reproduce the Results and Apply the Methodology

Code to reproduce all the experiments in the chapter is available in a GitHub repository https://github.com/EtorArza/GESP together with a brief explanation on how to apply the method.

6

General Conclusions and Future Work

6.1 Conclusions

In this dissertation, we studied stochastic heuristic optimization algorithms for non-convex optimization problems. We compared and analyzed optimization problems and optimization algorithms, emphasizing the stochastic nature and the use of computational resources in the latter.

Chapter 1

The performance of an optimization algorithm, with a maximum runtime as stopping criterion, is influenced by computational resources. The number of evaluated solutions increases with the available computational resources. Hence, if a new optimization algorithm is compared to another algorithm in the literature and the algorithms are not executed in the same machine with the same runtime, it is possible that one of the algorithms is given additional computational resources. This leads to an increased probability of falsely concluding that the new algorithm has a better performance, when in fact this difference is only a consequence of the difference in computational resources. For statistical hypothesis testing, this involves falsely rejecting the null hypothesis, when in fact it is true; and the probability of making this mistake is known as the probability of type I error [Conover, 1980].

To address this problem, in Chapter 1, we proposed a procedure to compare two algorithms executed in two different machines. The proposed method has two parts. Firstly, we proposed a method to predict the runtime of an algorithm on a CPU, and we use this model to adjust the runtime on one of the machines such that the runtime of the algorithm executed in the slowest CPU is compensated with extra computation time. Then, we proposed a correction of the one sided sign test [Conover, 1980] that can keep the probability of type I error low even when the optimization algorithms compared are executed in different machines.

Chapter 2

The objective value of the best found solution of a stochastic heuristic optimization algorithm will not be the same in each execution, and can thus be modeled as a random variable, denoted as *performance of an optimization algorithm* in this thesis. Comparing the performance of two optimization algorithms with only summary statistics like the median and the standard deviation might leave out important information.

While determining if (the performance) of one algorithms stochastically dominates [Quirk and Saposnik, 1962, Álvarez-Esteban et al., 2016] another would be ideal, this is often not possible in practice, as often neither algorithm dominates the other one. Existing methods to compare two algorithms such as null hypothesis statistical tests have limitations like being unable to provide information about the magnitude of the difference [Benavoli et al., 2017].

In response to these limitations, in Chapter 2 we proposed a methodology to compare the performance of two algorithms through stochastic dominance. Firstly, we proposed eight desirable properties for measures that compare random variables and a new measure of stochastic dominance that we named *dominance rate*. Then, we proposed a graphical representation with three interesting properties: i) a graphical decomposition of the probability of $A > B$ and the dominance rate, ii) differentiation between high uncertainty and low magnitude in difference, and iii) an estimation of the uncertainty via a 95% confidence band.

Chapter 3

Some optimization algorithms for permutation problems rely on distances/operators for permutations. Distances for permutations [Ceberio et al., 2015a] (e.g. Ulam and Cayley), usually measure the minimum number of times that an operator (e.g. insert and exchange [Schiavinotto and Stützle, 2007]) needs to be applied to transform a permutation into another permutation. The Hamming distance measures the number of different items in two permutations [Irurozki et al., 2016], and is not defined with an operator.

The performance of optimization algorithms that rely on distances for permutation problems is influenced the choice of distance. In Chapter 3, we studied and compared different distances for permutations and their relationship with the objective function of the Quadratic Assignment Problem (QAP) [Koopmans and Beckmann, 1957]. We carried out this comparison by i) measuring the changes in the objective function of the QAP and ii) comparing the performance of Estimation of Distribution Algorithms Larrañaga and Lozano [2001a] with these distances. In general, we argued that the Hamming distance is a suitable distance for assignment type problems.

Chapter 4

Problem analysis methods such as Fitness Landscape Analysis [Ochoa and Malan, 2019] and Local Optima Networks [Ochoa et al., 2014] analyze optimization problems and help us understand them. One of the limitations of the problem analysis methods in the literature is that they are particular to a solution space. For example, local optima networks assume a combinatorial optimization problem [Ochoa et al., 2014], or exploratory landscape analysis [Mersmann et al., 2011a] assumes a continuous optimization problem.

Hyper-heuristic algorithms select/design heuristics to solve a given optimization problem. They operate at a higher level than heuristics, which typically apply heuristics to optimization problems directly Burke et al. [2003]. Among Hyper-heuristic algorithms, we find those that "learn" to adapt to a optimization problem after an initial training phase (offline training).

Taking advantage of this "learning", it is possible to analyze a set of optimization problems, through the study of the transferability: how well does the hyper-heuristic perform, when it is trained in one problem and tested in another. Given a set of optimization problems, it is possible to find the differences/similarities between the problems by measuring the transferability between every possible pair of optimization problems. Hong et al. [2018] proposed this analysis by measuring the transferability as the average objective value of the hyper-heuristic when training and testing on two continuous optimization problems.

In Chapter 4, we proposed two multi-domain problem analysis method based on a multi-domain hyper-heuristic framework. Unlike other problem analysis methods, the proposed methods are applicable to different domains, and we show that they are able to identify similar and different problems within a set of optimization problems, both in the continuous and discrete domains. In addition, the proposed approach improves Hong et al. [2018]'s methodology in four key aspects. Firstly, by defining transferability with ranks instead of objective values, we can compare across different problems (rank based transferability is comparable among problems of different magnitude). Secondly, by repeating several measurements of transferability and computing the average ranks, it is possible to distinguish between noise and the actual difference in performance (this is not possible with average objective values). Thirdly, we experimentally show that the analysis carried out with our approach is correlated with the properties of the optimization problems, which suggests that the proposed technique is useful for finding similarities and differences in the properties of optimization problems. Finally, we show that our approach works in both combinatorial and continuous domains, while Hong et al.'s approach Hong et al. [2018] was only shown to work in the continuous domain.

Chapter 5

In optimization problems, long evaluation times are common in tasks like episodic policy learning. Early stopping is a technique used to terminate evaluations early when further improvements are unlikely. While many early stopping methods are problem-specific on policy learning—for example stopping the evaluation of a robot that got stuck requires determining when the robot is considered stuck—some general approaches (those that do not require problem specific knowledge) have been successful in problems like hyperparameter tuning. Bongard [2011] proposed a general early stopping approach for policy learning, but his approach relies on certain assumptions about the objective function and optimization algorithm.

In Chapter 5, we proposed an early stopping criterion for episodic policy learning that overcomes these limitations. We adapted early stopping approaches from hyperparameter optimization [de Souza et al., 2022] to episodic policy learning, by carefully considering the properties of the objective functions that policy learning tasks have. In an experimental section on five different environments, we showed that the proposed early stopping approach rarely decreases performance and can produce a speedup comparable to problem specific approaches, while being more generally applicable.

6.2 Future Work

The co-optimization of design and control in robotics is an interesting problem, where two stochastic optimization problems are nested one inside another. To optimize the design of the robot, we need to evaluate the designs proposed by an heuristic optimization algorithm, and in turn, evaluating a design requires its controller to be optimized. An efficient alternative [Jelisavcic et al., 2017, Liao et al., 2019, Sims, 1994] is to optimize the design and control together, although this has certain limitations, such as the convergence of the design before the controller [Lipson et al., 2016] and the requirement for controllers to be compatible with every possible designs simultaneously [Le Goff et al., 2021]. Given a fixed amount of time as stopping criterion, in the nested approach, the more we train a controller of each design, the less designs we will be able to evaluate, as evaluating a design involves fully training its controller.

It would be interesting to study how to allocate a fixed amount of computation budget to maximize performance, considering the trade-off between number of designs evaluated and number of controllers per design evaluated mentioned above. It might be possible, for example, to evaluate additional controllers for the most promising design found. In addition, it might be interesting to study the properties of the designs when the amount of controllers evaluated per design is decreased.

6.3 Main Achievements

A. ***Journal Papers***

Accepted:

- **Arza, E.**, Pérez, A., Irurozki, E., & Ceberio, J. (2020). Kernels of Mallows Models under the Hamming Distance for solving the Quadratic Assignment Problem. Swarm and Evolutionary Computation. **(Q1 in Computer Science)**
- **Arza, E.**, Ceberio, J., Irurozki, E., & Pérez, A. (2022). Comparing Two Samples Through Stochastic Dominance: A Graphical Approach. Journal of Computational and Graphical Statistics. **(Q1 in Statistics and Probability)**
- **Arza, E.**, Ceberio, J., Irurozki, E., & Pérez, A. (2023). On the Fair Comparison of Optimization Algorithms in Different Machines. Annals of Applied Statistics. **(Q1 in Statistics and Probability)**

Submitted:

- **Arza, E.**, Pérez, A., Irurozki, E., & Ceberio, J. Transferability in Optimization Via Multi-Domain Hyper-Heuristics. *On revision. To be resubmitted to* Transactions on Evolutionary Computation.
- **Arza, E.**, Le Goff, L., & Hart, E. Generalized Early Stopping in Evolutionary Direct Policy Search. *On revision. To be resubmitted to* ACM Transactions on Evolutionary Learning and Optimization.
- Echevarrieta, J., **Arza, E.** and Pérez, A. Optimal Evaluation Cost to Maximize Optimization Problem Solution Quality in a Given Runtime. *On revision. To be resubmitted to* Transactions on Evolutionary Computation.

B. ***Conference Posters***

- **Arza, E.**, Ceberio, J., Pérez, A., & Irurozki, E. (2019). Approaching the Quadratic Assignment Problem with Kernels of Mallows Models Under the Hamming Distance. Proceedings of the Genetic and Evolutionary Computation Conference Companion.
- **Arza, E.**, Ceberio, J., Pérez, A., & Irurozki, E. (2020). An Adaptive Neuroevolution-based Hyper-Heuristic. Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.

- **Arza, E.**, Ceberio, J., Irurozki, E., & Pérez, A. (2022). Implementing the cumulative difference plot in the IOHanalyzer. Proceedings of the Genetic and Evolutionary Computation Conference Companion.

C. ***Prizes***

- Optimization Competition 2022 for Better Benchmarking of Sampling-Based Optimization Algorithms.

D. ***Research Stays***

- 24 January - 24 April 2022. At the Edinburgh Napier University, Edinburgh, Scotland. Supervision: Emma Hart and Léni Le Goff.
- 24 January - 23 April 2023. At Oslo Universität, Oslo, Norway. Supervision: Kyrre Glette, Tønnes Nygaard and Frank Veenstra.

7

Appendices

7.1 Chapter 1

7.1.1 The importance of using the same resources in algorithm comparison

It is essential to run the algorithms with the same computational resources to carry out a fair comparison. To better illustrate this point, in the following lines, a small experiment is presented. This experiment illustrates the increase in the probability of type I error (the probability of erroneously concluding a difference in performance, when in reality, there is none) with respect to the difference in execution time. Specifically, we run a random search algorithm twice in each problem instance[1] and perform the one-sided sign test [Conover, 1980][2] (see Section 1.3.1 for an explanation of the sign test), on the set of results obtained. The significance level is set to $\alpha = 0.05$. Even though the random search algorithm is being compared with itself, we increase the runtime of one of the executions by 8, 16, 32, or 64 percent. We repeat the steps above 1000 times to estimate the probability of type I error (estimated as the probability of rejecting H_0).

Figure 7.1 shows the estimated probability of type I error. Notice that the type I error starts at 0.05, which is the expected result for a significance level of $\alpha = 0.05$. However, the error shoots up dramatically when the difference in runtime increases, more than doubling when the percentage of extra runtime reaches 32%.

[1] A set of 16 permutation problem instances is considered, 4 instances of 4 problems. The four permutation problems considered are the traveling salesman problem, the permutation flowshop scheduling problem, the linear ordering problem, and the quadratic assignment problem.

[2] In 7.1.4, we explain why we limit the statistical analysis to the sign test in this chapter.

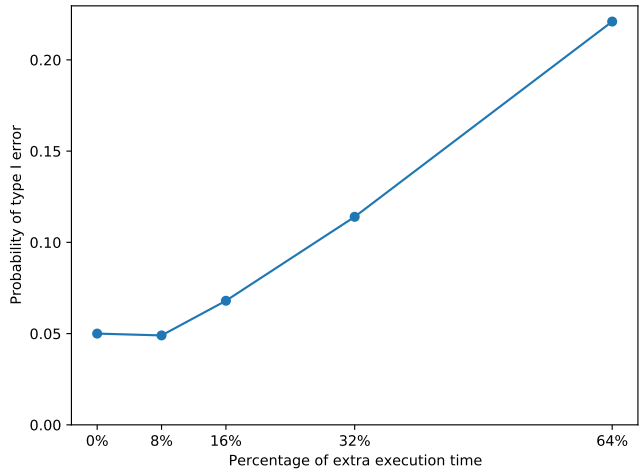


Fig. 7.1: Increase in the probability type I error with respect to the difference in runtime. Specifically, the probability of type I error in the one-sided sign test when comparing two identical random search algorithms. One of the algorithms is given extra runtime, according to the x-axis. The test is applied to a set of 16 problem instances.

Therefore, a discrepancy in the runtime of the algorithms being compared, if high enough, can lead to falsely concluding that the performance of the algorithms is not the same. A fair comparison requires the same computational resources to be assigned in the execution of each algorithm.

7.1.2 Justification of Assumption 1

The runtime of an optimization process (a sequence of computational instructions) is different in each machine. However, even though it is different, there might be a proportional relationship between the runtime of the same optimization process in two different machines. This hypothesis is the basis of Assumption 1.

To experimentally study this assumption, we compute the correlation of the runtime that several optimization processes have on two machines. Specifically, we computed the correlation of 64 different optimization processes (see Appendix 7.1.3 for additional details on the optimization processes) for every possible pair of machines from the 8 different machines used in the experimentation. The average Pearson's correlation coefficient of the runtimes is 0.989987, which shows a strong

linear [Zou et al., 2003] (not necessarily proportional) relationship between the runtime of the same optimization process in two different machines.

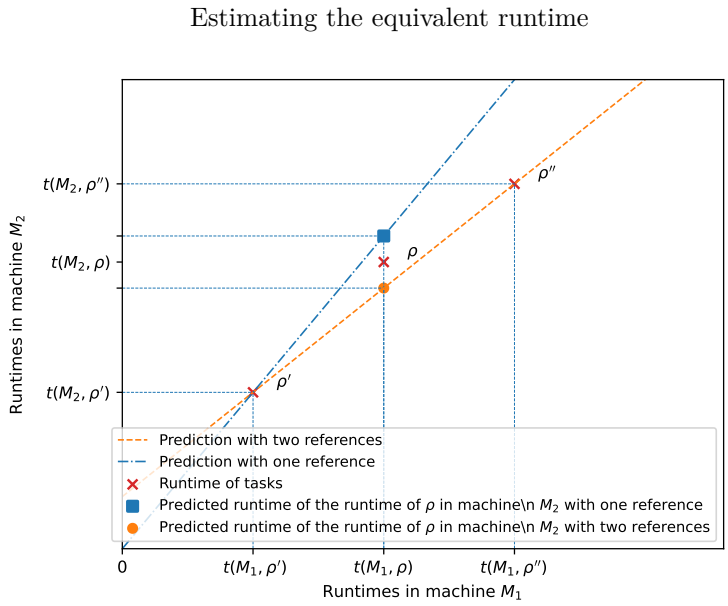


Fig. 7.2: Estimation of the runtime of an optimization process ρ with one ρ' or two ρ', ρ'' reference optimization processes. The x-axis represents the runtime in machine M_1, while the y-axis is the runtime in machine M_2. The runtime of the optimization process ρ is estimated for machine M_2.

Given two machines M_1 and M_2, the runtime of an optimization process can be considered as a two-dimensional vector, where each of the dimensions represents the runtime of the optimization process in each of the machines. Thus, knowing the runtime $t(s, M_1)$ of an optimization process ρ in a machine M_1, it is reasonable to estimate the equivalent runtime of ρ in another machine M_2, when the runtime of two other optimization processes ρ' and ρ'' is known for both machines. In fact, with such a high Pearson's correlation coefficient, the runtimes of these optimization processes (red crosses in Figure 7.2) will almost be aligned in a line [Zou et al., 2003]. Therefore, the estimated runtime of ρ in machine M_2 is defined as the value that makes the runtime of the three optimization processes aligned. This is shown by the orange line in Figure 7.2.

Observe that this procedure requires the runtime of two optimization processes ρ' and ρ'' to be known in both machines M_1, M_2. However, by considering an additional hypothesis, we can reduce the requirement to only one optimization process ρ'. This additional hypothesis is that the regression line has to cross the origin. Intuitively, if an optimization process (sequence of computational instructions) takes no time in a machine, it makes no sense that it takes a positive amount of time in another machine. In addition, without this condition, it could be possible to estimate a negative runtime, which is not properly defined.

In this setting, the estimated runtime for the optimization process ρ in machine M_2 is set so that the runtime of the optimization processes ρ and ρ' and the origin are in the same line. This is represented by the blue line in Figure 7.2. The estimation of the runtime of the optimization process ρ in machine M_2, shown in the figure as a blue square, is given by the slope-intercept formula for the points $(0, 0)$ and $(t(M_1, \rho'), t(M_2, \rho'))$:

$$t(M_2, \rho) \approx \frac{t(M_2, \rho')}{t(M_1, \rho')} t(M_1, \rho) \tag{7.1}$$

By rewriting Equation (7.1), we obtain that the ratio of two optimization processes is (approximately) constant in different machines

$$\frac{t(M_2, \rho)}{t(M_2, \rho')} \approx \frac{t(M_1, \rho)}{t(M_1, \rho')} \tag{7.2}$$

which is exactly Assumption 1.

7.1.3 Optimization processes

In this chapter, we defined an optimization process as a sequence of computational instructions that can be executed in any machine. Specifically, each of the optimization processes described in this section consists of executing an optimization algorithm in a problem instance for a maximum of $2 \cdot 10^6$ objective function evaluations. In total, we considered 64 optimization processes, executing 4 algorithms in 16 problem instances. The optimization process ρ' is the sequential execution of these 64 optimization processes.

Problem instances: We solved four types of optimization problems (all of them are permutation problems): the traveling salesman problem [Goldberg and Lingle, 1985], the quadratic assignment problem [Koopmans and Beckmann, 1957], the linear ordering problem [Ceberio et al., 2015g] and the permutation flowshop scheduling problem [Gupta and Stafford, 2006]. For each of these four problem types, we chose 4 problem instances, as listed in Table 7.1.

Optimization algorithms: Each of the 16 problem instances was optimized with four optimization algorithms. These optimization algorithms are random search and local search with three different neighborhoods: swap, interchange, and insert [Schiavinotto and Stützle, 2007, Ceberio et al., 2015a]. The local search is a best-first or greedy approach that is randomly reinitialized when a local optimum is found.

We define each of the 64 different optimization processes as running each of these four optimization algorithms in each of the 16 problem instances.

Problem instances

instance name	problem	size
tai75e02	qap	75
sko100a	qap	100
tai100a	qap	100
tai100b	qap	100
eil101	tsp	101
pr136	tsp	136
kroA200	tsp	200
kroB200	tsp	200
tai100_20_0	pfsp	(100,20)
tai100_20_1	pfsp	(100,20)
tai200_20_1	pfsp	(200,20)
tai200_20_1	pfsp	(200,20)
N-be75np_150	lop	150
N-stabu3_150	lop	150
N-t65d11xx_150	lop	150
N-t70f11xx_150	lop	150

Table 7.1: The list of 16 problem instances and their size.

Machines: The experimentation was carried out in a set of 8 different machines. Table 7.2 lists the CPU models of these machines, as well as their single thread PassMark CPU scores.

7.1.4 The sign test for algorithm performance comparison

When statistically assessing the comparison of the performance of optimization algorithms, a classical way is to use non-parametric tests as the distribution of the performance is usually unknown. In the literature, the Wilcoxon signed-rank test, the Mann-Whitney test and the sign test [Conover, 1980] are often used to

Machines

CPU model name	PassMark score
Intel i5 470U	539
Intel Celeron N4100	1012
AMD A9 9420 with Radeon R5	1344
AMD FX 6300	1486
Intel i7 2760QM	1559
Intel i7 6700HQ (2.60GHz)	1921
Intel i7 7500U	1955
AMD Ryzen7 1800X	2185

Table 7.2: The list of 8 machines used in the experimentation and their speed score, measured in terms of PassMark single thread score.

assess a statistically significant difference in the performance of two algorithms. We argue that, in the context of optimization algorithm performance comparison, it may be more suitable to use the sign test than the Wilcoxon signed-rank or the Mann-Whitney test.

It turns out that the result of the Wilcoxon and the Mann-Whitney tests might change when the objective function value of some of the problems is scaled (multiplied or divided by a positive constant). The reason is that they both take into account the magnitude of the differences between the observations, and these differences change with scaling. A usual solution is to consider the average relative deviation percentage with respect to the optimum (or any other reference solution) instead of the objective value, but this only changes the problem: now the results of these tests change when the objective function value of some of the problems is shifted (add or subtract a constant). In our opinion, the performance comparison of two optimization algorithms should be invariant to these two alterations, otherwise, problems that are on a higher scale (for example, when the dimension of the problem is high), will have a larger impact on the result of the statistical test. In addition, we believe that it is reasonable that all problem instances have the same weight in the conclusion of the statistical test, which both the Wilcoxon signed-rank and the Mann-Whitney test are unable to accomplish due to their dependence on the magnitude of the differences.

An alternative is the sign test [Conover, 1980], which is invariant to the shifting and scaling of the problems. In fact, the result of the sign test does not change even if some of the problems are modified by composing the objective function with any strictly increasing function. For this reason, and even though the sign test is a less powerful alternative (higher probability of type II error), we believe it is the most suitable hypothesis test for algorithm performance comparison when the objective functions of all the problems are not directly comparable.

7.1.5 Proof of Equation (1.5).

When performing the statistical analysis, a set of n problem instances is used to compute the statistic and the p-value. The goal of the analysis is to draw conclusions on a larger set of problem instances based on the observed sample of size n. Given a problem instance, we can define the performance of an algorithm in this instance.

Definition 16 *(The performance of an algorithm in an instance)*
Let M be a machine, t a stopping criterion in terms of maximum runtime, A an optimization algorithm and i a problem instance. The performance of algorithm A in an instance i, denoted $A(M,t,i)$, is defined as a random variable whose outcome is obtained by first sampling a random seed r and then optimizing instance i with optimization algorithm A in machine M for time t. Given this random seed r, the performance of an algorithm in an instance is deterministic.

In Section 1.2, we defined t_1 as the stopping criterion for algorithm A in machine M_1, which is obviously the time it takes to carry out this optimization process in machine M_1. We also defined the equivalent runtime t_2 as the time it takes to replicate the exact same optimization process in machine M_2 in Definition 6. Because of this definition, $A(M_1,t_1,i)$ and $A(M_2,t_2,i)$ are the same random variables. Therefore, it makes sense to denote $A(M_1,t_1,i)$ and $A(M_2,t_2,i)$ or $B(M_1,t_1,i)$ and $B(M_2,t_2,i)$ as A_i or B_i, respectively. To ease the notation, we will also denote $B(M_2,\hat{t}_2,i)$ as $\hat{B}_i$.

Finally, as discussed in Section 1.5.6, we assume that whether $\hat{t}_2 < t_2$ is true or not is independent for each instance i, and that $\mathcal{P}(\hat{t}_2 < t_2) < p_\gamma$. Let us now prove Equation (1.5).

Lemma 1 *Let n be an integer, X and Y two random variables. Let $X_1,...,X_n$ be n independent random variables distributed as X. Let $Y_1,...,Y_n$ be n independent random variables distributed as Y. Let v_x and v_y be two possible outcomes of the random variables X and Y respectively, $l \in \{0,...,n\}$ be an integer and $p \in (0,1)$ be a real number.*

I) If $\mathcal{P}[Y = v_y \mid X = v_x] = 1$, then

$$\mathcal{P}[X = v_x] \leq \mathcal{P}[Y = v_y]$$

and

$$\#\{X_i = v_x\} \leq \#\{Y_i = v_y\}$$

II) If $\mathcal{P}[Y = v_y \mid X = v_x] = 1$ *and* $\mathcal{P}[X = v_x \mid Y = v_y] = 1$ *then*

$$\mathcal{P}[X = v_x] = \mathcal{P}[Y = v_y]$$

III) If $\mathcal{P}[X = v_x] < p$ *then*

$$\mathcal{P}[\#\{X_i = v_x\} \geq l] < \mathcal{P}[Bin(n, p) \geq l]$$

Lemma 2 *Let* $i \in \{1, .., n\}$ *be* n *problem instances and let* A *and* B *be two optimization algorithms. Let* a_i, b_i *and* $\hat{b}_i$ *be the observed values of* A_i, B_i *and* $\hat{B}_i$ *respectively,* $\forall i \in \{1, ..., n\}$. *Let* k *and* $v \in \{0, ..., n\}$ *be two integers. Suppose that* $A_i \neq B_i$ *and* $A_i \neq \hat{B}_i$.

Then,

$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v) \mid \#\{A_i < B_i\} = v] \leq$$
$$\mathcal{P}[\#\{A_i > \hat{B}_i \wedge A_i < B_i\} \geq \max(0, v - k) \mid \#\{A_i < B_i\} = v]$$

Proof.

$$\#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v) \implies$$
$$\#\{A_i < \hat{B}_i \wedge A_i < B_i\} \leq \min(k, v) \implies$$
$$\#\{A_i < B_i\} - \#\{A_i > \hat{B}_i \wedge A_i < B_i\} \leq \min(k, v) \implies$$

Substituting $\#\{A_i < B_i\} = v$,

$$v - \min(k, v) \leq \#\{A_i > \hat{B}_i \wedge A_i < B_i\} \implies$$

Considering $v - \min(k, v) = \max(0, v - k)$,

$$\#\{A_i > \hat{B}_i \wedge A_i < B_i\} \geq \max(0, v - k)$$

We have just shown that

$$\#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v) \implies \#\{A_i > \hat{B}_i \wedge A_i < B_i\} \geq \max(0, v - k)$$

Which means that,

$$\mathcal{P}[\#\{A_i > \hat{B}_i \wedge A_i < B_i\} \geq \max(0, v-k) \mid \#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v)] = 1$$

Finally, we apply Lemma 1 I), obtaining

$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v) \mid \#\{A_i < B_i\} = v] \leq$$
$$\mathcal{P}[\#\{A_i > \hat{B}_i \wedge A_i < B_i\} \geq \max(0, v-k) \mid \#\{A_i < B_i\} = v]$$

Lemma 3 *Let $i \in \{1, .., n\}$ be n problem instances and let A and B be two optimization algorithms. Let a_i, b_i and $\hat{b}_i$ be the observed values of A_i, B_i and $\hat{B}_i$ respectively, $\forall i \in \{1, ..., n\}$. Let k and $v \in \{0, ..., n\}$ be two integers. Suppose that $A_i \neq B_i$ and $A_i \neq \hat{B}_i$.*

Then,

$$\mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \mid \#\{A_i < B_i\} = v] < \mathcal{P}[Bin(n, p_\gamma) \geq \max(0, v-k)]$$

Proof.

$$\mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \mid \#\{A_i < B_i\} = v] \leq$$
$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq k \mid \#\{A_i < B_i\} = v]$$

Now, observe that $\#\{A_i < \min(B_i, \hat{B}_i)\} \leq \#\{A_i < B_i\} = v$, which implies that

$$\#\{A_i < min(\hat{B}_i, B_i)\} \leq k \iff \#\{A_i < min(\hat{B}_i, B_i)\} \leq min(k, v)$$

This means that

$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq k \mid \#\{A_i < min(\hat{B}_i, B_i)\} \leq min(k, v) \wedge \#\{A_i < B_i\} = v] = 1$$

and

$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq min(k, v) \mid \#\{A_i < min(\hat{B}_i, B_i)\} \leq k \wedge \#\{A_i < B_i\} = v] = 1$$

We apply Lemma 1 II), obtaining

$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq k \mid \#\{A_i < B_i\} = v] =$$
$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v) \mid \#\{A_i < B_i\} = v]$$

Applying Lemma 2, we obtain

$$\mathcal{P}[\#\{A_i < min(\hat{B}_i, B_i)\} \leq \min(k, v) \mid \#\{A_i < B_i\} = v] \leq$$
$$\mathcal{P}[\#\{A_i > \hat{B}_i \wedge A_i < B_i\} \geq \max(0, v - k) \mid \#\{A_i < B_i\} = v]$$

Note that b_i is the score obtained with the true equivalent runtime t_2 as the stopping criterion, while in the case of $\hat{b}_i$, the stopping criterion is the estimated equivalent runtime $\hat{t_2}$. In a minimization context, $\hat{b}_i < b_i \implies \hat{t_2} > t_2$, because a better score can only be obtained with a longer runtime (a shorter runtime implies an equal or worse performance). Let us consider the following implications:

$$a_i > \hat{b}_i \wedge a_i < b_i \implies \hat{b}_i < b_i \implies \hat{t_2} > t_2$$

We infer that

$$\mathcal{P}[\hat{t_2} > t_2 \mid A_i > \hat{B}_i \wedge A_i < B_i] = 1$$

Applying Lemma 1 I), we obtain

$$\mathcal{P}[\#\{A_i > \hat{B}_i \wedge A_i < B_i \mid \#\{A_i < B_i\} = v\} \geq \max(0, v - k)] \leq$$
$$\mathcal{P}[\#\{\hat{t_2} > t_2 \mid \#\{A_i < B_i\} = v\} \geq \max(0, v - k)] =$$
$$\mathcal{P}[\#\{\hat{t_2} > t_2\} \geq \max(0, v - k)]$$

The estimated runtime $\hat{t}_2$ was computed with the equation in Definition 8 in Section 1.2, with an estimated probability that $\hat{t}_2 < t_2$ lower than 0.01. With this information, we apply Lemma 1 III) taking into account that $\mathcal{P}[\hat{t}_2 > t_2] < 0.01$:

$$\mathcal{P}[\#\{\hat{t_2} > t_2\} \geq \max(0, v - k)] <$$
$$\mathcal{P}[Bin(n, 0.01) \geq \max(0, v - k)]$$

Theorem 1 *Let $i \in \{1, .., n\}$ be n problem instances and let A and B be two optimization algorithms. Let a_i, b_i and $\hat{b}_i$ be the observed values of A_i, B_i and $\hat{B}_i$ respectively, $\forall i \in \{1, ..., n\}$. Let H_0 be the null hypothesis under which the statistic $\#\{A_i < B_i\}$ follows the null distribution $Bin(n, 0.5)$. Suppose that $A_i \neq B_i$ and $A_i \neq \hat{B}_i$. Then,*

$$\mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \mid H_0] \leq$$
$$\sum_{v=0}^{n} (1 - \mathcal{P}[Bin(n, 0.01) < \max(0, v - k)]) \cdot \mathcal{P}[Bin(n, 0.5) = v]$$

Proof. Let X, C be a two random variables, where S_C and S_X are the sets of all possible outcomes of C and X respectively. Consider the law of total probability [Beyer, 1991]:

$$\forall x \in S_X \ , \ \mathcal{P}[X = x] = \sum_{c \in S_C} \mathcal{P}[C = c] \cdot \mathcal{P}[X = x \ \mid \ C = c]$$

Applying this formula, we obtain

$$\mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \ \mid \ H_0] =$$

$$\sum_{v=0}^{n} \mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \ \mid \ H_0 \wedge \#\{A_i < B_i\} = v] \cdot \mathcal{P}[\#\{A_i < B_i\} = v \ \mid \ H_0]$$

Given that $\#\{A_i < B_i\} = v$, we can say that $\#\{A_i < \hat{B}_i\} \leq k$ is independent of H_0, because $\#\{A_i < \hat{B}_i\}$ is determined by how many times $\hat{t}_2 > t_2$ resulted in $A_i < B_i \wedge A_i > \hat{B}_i$ and $\hat{t}_2 < t_2$ resulted in $A_i > B_i \wedge A_i < \hat{B}_i$. Specifically, H_0 gives the prior probabilities of $A_i > B_i$, which are not relevant when we know that $\#\{A_i > B_i\} = v$. That gives us

$$\sum_{v=0}^{n} \mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \ \mid \ H_0 \wedge \#\{A_i < B_i\} = v] \cdot \mathcal{P}[\#\{A_i < B_i\} = v \ \mid \ H_0] =$$

$$\sum_{v=0}^{n} \mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \ \mid \ \#\{A_i < B_i\} = v] \cdot \mathcal{P}[\#\{A_i < B_i\} = v \ \mid \ H_0]$$

Applying Lemma 3 and considering that H_0 implies the null distribution $Bin(n, 0.5)$ for the statistic $\#\{A_i < B_i\}$,

$$\sum_{v=0}^{n} \mathcal{P}[\#\{A_i < \hat{B}_i\} \leq k \ \mid \ \#\{A_i < B_i\} = v] \cdot \mathcal{P}[\#\{A_i < B_i\} = v \ \mid \ H_0] <$$

$$\sum_{v=0}^{n} \mathcal{P}[Bin(n, 0.01) \geq \max(0, v - k)] \cdot \mathcal{P}[\#\{A_i < B_i\} = v \ \mid \ H_0] =$$

$$\sum_{v=0}^{n} \mathcal{P}[Bin(n, 0.01) \geq \max(0, v - k)] \cdot \mathcal{P}[Bin(n, 0.5) = v] =$$

$$\sum_{v=0}^{n} (1 - \mathcal{P}[Bin(n, 0.01) < \max(0, v - k)]) \cdot \mathcal{P}[Bin(n, 0.5) = v]$$

7.2 Chapter 2

7.2.1 A literature review of measures

7.2.1.1 f-divergences

The *f-divergence* is a family of functions that can be used to measure the difference between two random variables. Given a strictly convex[1] function $f : (0, +\infty) \to \mathbb{R}$ with $f(1) = 0$, and two continuous random variables A and B, the f-divergence [Liese and Vajda, 2006, Rényi et al., 1961] is defined as

$$D_f(A, B) = \int_{\mathbb{R}} g_B(x) f\left(\frac{g_A(x)}{g_B(x)}\right) dx \tag{7.3}$$

where g_A and g_B are the probability density functions of the random variables A and B respectively. Since $g_B(x)$ can be 0, we assume [Polyanskiy and Wu, 2012] that $0 \cdot f(0/0) = 0$ and $0 \cdot f(a/0) = \lim_{x \to 0+} x \cdot f(a/x)$. Notice that if g_A and g_B are the same probability density functions, then $D_f(A, B) = 0$.

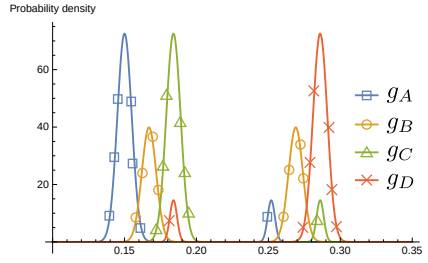
Kullback–Leibler divergence: The Kullback–Leibler divergence [Kullback and Leibler, 1951] is a particular case of the f-divergence, for $f(x) = x \cdot \ln(x)$. Given two random variables A and B, $D_{KL}(A, B)$ can be interpreted [Papadopoulos, 2017] as the amount of entropy increased by using g_B to model data that follows the probability density function g_A.

The Kullback–Leibler divergence is non-negative, and non symmetric $D_{KL}(A, B) \neq D_{KL}(B, A)$, and therefore, it is not actually a distance [Goodfellow et al., 2016]. It will not satisfy Property (2), as it is not antisymmetric either. This also makes the interpretation less intuitive. The Kullback–Leibler divergence is often used to measure the difference between two random variables [Goodfellow et al., 2016], but since $D_{KL}(A, B) \neq D_{KL}(B, A)$, it may be better to interpret the Kullback–Leibler divergence as stated above [Papadopoulos, 2017].

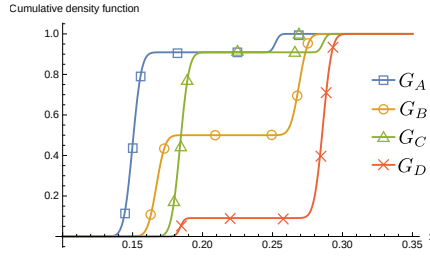
In Figure 7.3, we show the probability density functions and cumulative distribution functions of four random variables A, B, C and D. Looking at their cumulative distributions (Figure 7.3b), one can clearly see that $A \succ B$, $B \lesseqgtr C$ and $B \succ D$. However, as shown in Table 7.3, $D_{KL}(B, A) = D_{KL}(B, C) = D_{KL}(B, D) = 15.4$ and $D_{KL}(A, B) = D_{KL}(C, B) = D_{KL}(C, D) = 6.2$. This means that, given any two random variables A and B, the Kullback-Leibler is not able to distinguish if $A \succ B$, $B \succ A$ or $A \lesseqgtr B$. We can interpret this as the Kullback–Leibler divergence only caring about the difference between two random variables, and not if

[1] A function $f : (0, +\infty) \to \mathbb{R}$ is strictly convex if for all $t \in [0, 1]$, for all $x_1, x_2 \in (0, +\infty)$, $f(tx_1 + (1 - t)x_1) < tf(x_1) + (1 - t)f(x_2)$

this difference is related to one of the random variables taking lower values than the other. Hence, it cannot satisfy Property 1, even if we try to transform it to be defined in the $[0, 1]$ interval. We conclude that the Kullback–Leibler divergence is not suitable to gain information regarding which of the random variables takes lower values.



(a) Probability density.



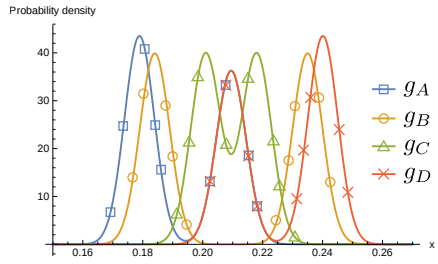
(b) Cumulative distribution function.

Fig. 7.3: The probability density function and cumulative distribution of the four random variables. The distances between these random variables are listed in Table 7.3.

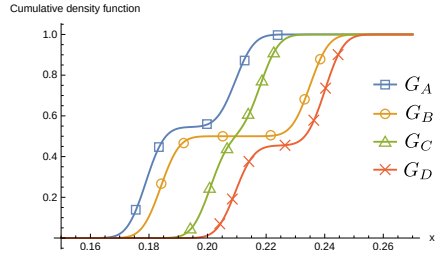
Jensen-Shannon divergence: The Jensen-Shannon divergence [Polyanskiy and Wu, 2012] is very similar to the Kullback-Leibler divergence, and is another the particular case of the f-divergence for $f(x) = x \cdot \ln(\frac{2x}{x+1}) + \ln(\frac{2}{x+1})$. It is also known as the symmetrized version of the Kullback–Leibler divergence [Polyanskiy and Wu, 2012], because

$$D_{JS}(A, B) = D_{KL}(A, X_{\mathcal{M}}) + D_{KL}(B, X_{\mathcal{M}})$$

where the probability density function of $X_{\mathcal{M}}$ is $g_{\mathcal{M}}(x) = 0.5(g_A(x) + g_B(x))$. Thus, we can interpret this divergence as the sum of the Kullback–Leibler divergences of g_A and g_B with respect to the average probability density function $g_{\mathcal{M}}$.



(a) Probability density.



(b) Cumulative distribution function.

Fig. 7.4: The probability density function and cumulative distribution of four other random variables. The Wasserstein distance between B and each of the other random variables is 0.017.

The Jensen-Shannon divergence also fails to identify (see Table 7.3) the dominance relationships between B and the rest of the random variables in Figure 7.3, thus, it cannot satisfy Property 1. In addition, the Jensen-Shannon divergence also fails to satisfy Properties 2 and 3. See Table 2.1 for a detailed list of the properties that each measure satisfies.

Total variation: The total variation [Polyanskiy and Wu, 2012] is also a particular f-divergence, for $f(x) = \frac{1}{2}|x-1|$. Unlike the Kullback–Leibler divergence, the total variation is symmetric. In fact, it is a properly defined distance [Polyanskiy and Wu, 2012, Tsybakov, 2009]. In addition, it is defined between 0 and 1.

Given two random variables A, B, the total variation can also be defined as:

$$TV(A,B) = \sup_{C \subseteq \mathbb{R}} |\mathcal{P}_A(C) - \mathcal{P}_B(C)|,$$

where $\mathcal{P}_A$ and $\mathcal{P}_B$ are the probability distributions[1] of A and B respectively. Since the subset C that takes the supremum is $C = \{x \in \mathbb{R} \mid g_A(x) > g_B(x)\}$[Devroye

[1] Given the random variable A defined in $\mathbb{R}$, its probability distribution, noted as $\mathcal{P}_A$, is a mapping that, for all $U \subseteq \mathbb{R}$ that is measurable, $A(U) = \mathcal{P}(A \in U)$ [Vapnik, 1998].

Kullback–Leibler

RV_1 \ RV_2	A	B	C	D
A	0.0	6.2	28.6	88.8
B	15.4	0.0	15.4	15.4
C	29.4	6.2	0.0	2.6
D	88.8	6.2	2.6	0.0

Jensen-Shannon

RV_1 \ RV_2	A	B	C	D
A	0.0	1.2	1.4	1.4
B	1.2	0.0	1.2	1.2
C	1.4	1.2	0.0	0.8
D	1.4	1.2	0.8	0.0

Total variation

RV_1 \ RV_2	A	B	C	D
A	0.000	0.934	0.999	1.000
B	0.934	0.000	0.934	0.934
C	0.999	0.934	0.000	0.818
D	1.000	0.934	0.818	0.000

Hellinger

RV_1 \ RV_2	A	B	C	D
A	0.00	1.28	1.41	1.41
B	1.28	0.00	1.28	1.28
C	1.41	1.28	0.00	0.99
D	1.41	1.28	0.99	0.00

Wasserstein

RV_1 \ RV_2	A	B	C	D
A	0.000	0.06	0.03	0.12
B	0.06	0.00	0.04	0.06
C	0.03	0.04	0.000	0.083
D	0.12	0.06	0.083	0.000

$\mathcal{C}_{\mathcal{P}}$

RV_1 \ RV_2	A	B	C	D
A	0.50	0.95	0.92	0.99
B	0.05	0.50	0.54	0.95
C	0.08	0.46	0.50	0.91
D	0.01	0.05	0.09	0.50

$\mathcal{C}_{\mathcal{D}}$

RV_1 \ RV_2	A	B	C	D
A	0.50	1.00	1.00	1.00
B	0.00	0.50	0.59	1.00
C	0.00	0.41	0.50	1.00
D	0.00	0.00	0.00	0.50

Table 7.3: $\mathcal{C}(\mathrm{RV}_1, \mathrm{RV}_2)$ for the random variables A, B, C and D shown in Figure 7.3.

et al., 2020], we can interpret the total variation as the "size" of the difference in the density functions in all points where g_A is more likely than g_B. Following this intuition, when $TV(A, B) = 1$, g_A and g_B have disjoint supports [Polyanskiy and Wu, 2012], and thus A and B are at their maximum difference with respect to this metric. On the other hand, when $TV(A, B) = 0$ the random variables are identical.

The Total-Variance also fails to identify (see Table 7.3) the dominance relationships between B and the rest of the random variables in Figure 7.3.

Hellinger distance and the Bhattacharyya distance: The Hellinger distance is the square root of the f-divergence for $f(x) = (1 - \sqrt{x})^2$ [Polyanskiy and Wu,

2012]. It is related to the Bhattacharya coefficient, since $D_H(A,B) = 2(1 - \text{BhattCoef}(A,B))$ [Polyanskiy and Wu, 2012, Xi, 2017], where $\text{BhattCoef}(A,B)$ is the Bhattacharyya coefficient [Kailath, 1967, Bhattacharyya, 1943]. This coefficient is defined as $\text{BhattCoef}(A,B) = \int_{\mathbb{R}} \sqrt{g_A(x)g_B(x)}dx$, and has proven useful on signal processing [Kailath, 1967]. Given two probability density functions g_A and g_B, the Bhattacharyya coefficient can be interpreted as the integral of the geometric mean of the probability density functions. The Bhattacharyya coefficient is also related to the Bhattacharyya distance, as $D_{Bhatt}(A,B) = -\ln(\text{BhattCoef}(A,B))$.

The Hellinger distance and the Bhattacharyya distance also fail to identify (see Table 7.3) the dominance relationships between B and the rest of the random variables in Figure 7.3.

7.2.1.2 Wasserstein distance

The Wasserstein distance is another type of distance between probability random variables. Given two continuous random variables A, B, the Wasserstein distance (of order 1) is defined as [Schuhmacher, 2021, Panaretos and Zemel, 2019]

$$D_W(A,B) = \int_{\mathbb{R}} |G_A(x) - G_B(x)|dx$$

In Figure 7.4, we show a different set of four random variables A, B, C and D. In this case, it is also clear that $A \succ B$, $B \lesseqgtr C$ and $B \succ D$ (Figure 7.4b), but $D_W(B,A) = D_W(B,C) = D_W(B,D) = 0.017$. Therefore, in this case, the Wasserstein distance does not give any insights about the dominance between B and the rest of the random variables, thus, it cannot satisfy Property 1 even with a transformation. It also does not satisfy Properties 2, 3, 6, 7, 8.

However, with a small change, the Wasserstein distance can comply with Properties 2 and 3. This change also improves its correlation with the dominance, even though it still does not comply with Property 1. We remove the absolute value, such that the *signed Wasserstein* distance is defined as

$$D_{SW}(A,B) = \int_{\mathbb{R}} G_A(x) - G_B(x)dx.$$

For the random variables in Figure 7.4, the signed Wasserstein distance has different values: $D_{SW}(B,A) = 0.17$, $D_{SW}(B,C) = 0$ and $D_{SW}(B,D) = -0.017$. Notice that

$$A \succ B \implies D_{SW}(B,A) > 0 \text{ and } B \succ A \implies D_{SW}(B,A) < 0,$$

but unfortunately, when $A \lesseqgtr B$, $D_{SW}(B, A)$ could be positive or negative. This implies that $D_{SW}(B, A)$ still can not determine if $A \succ B$, $B \succ A$, or $A \lesseqgtr B$.

7.2.1.3 Heuristic derivation of the first-order stochastic dominance

A measure similar to the Wasserstein distance has been proposed in the literature [Schmid and Trede, 1996] in the context of comparing random variables. Specifically, this measure is part of the heuristic derivation of a distribution-free statistical test for first-order stochastic dominance [Schmid and Trede, 1996]. Given two random variables A, B, this measure is defined as

$$\mathcal{C}_I(A, B) = \int_{\mathbb{R}} max(0, G_A(x) - G_B(x))dG_B(x).$$

Note that the values of I range between 0 and 0.5. When $\mathcal{C}_I(A, B) = 0.5$, we know that $A \succ B$. Unfortunately, when $\mathcal{C}_I(A, B) \in (0, 0.5)$, it could be that $A \succ B$ or $A \not\succ B$. Consequently, $\mathcal{C}_I(A, B)$ cannot satisfy Property 1.

7.2.2 Quantile random variables

7.2.2.1 Computing the probability density functions of Y_A and Y_B

In Section 2.4.1 we introduced the quantile random variables Y_A and Y_B. We now describe how to compute the probability density functions of g_{Y_A} and g_{Y_B} step by step, with the pseudocode shown in Algorithm 7. We define a function r that returns the position of an observation according to its rank in the sorted list of the observation $A^n \cup B^n$ (lines 1–4). The ranks go from 0 (for the smallest observation) to r_{max} (for the largest), where r_{max} is the number of unique observation in $A^n \cup B^n$ minus 1. Repeated observations are assigned the same rank, and no ranks are skipped: there is at least a value in $A^n \cup B^n$ corresponding to each rank from 0 to r_{max}. For each observation in $\{a_1, ..., a_n\}$, a uniform distribution defined in the interval $(\frac{r(a_i)+\gamma(r(a_i)-1)}{2n}, \frac{r(a_i)+\gamma(r(a_i))}{2n})$ is added to the mixture (lines 10–19), where $\gamma(k)$ (lines 7–9) counts the number of ranks in $A^n \cup B^n$ that are lower than or equal to k (since the lowest rank is 0, $\gamma(-1) = 0$) . The kernel density estimation for Y_B is defined similarly, but with the observations $\{b_1, ..., b_n\}$ instead.

7.2.2.2 The quantile random variables have the same $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ as the kernel density estimates of A and B.

In Section 2.4.1, we claimed that when a "small enough" uniform [scikit-learn developers, 2021] kernel is used in the kernel density estimations of A and B,

Algorithm 7: Kernel density estimation of Y_A and Y_B

Input:
$A^n = \{a_1, ..., a_n\}$: The n observed samples of A.
$B^n = \{b_1, ..., b_n\}$: The n observed samples of B.

Output:
g_{Y_A}: The probability density of Y_A.
g_{Y_B}: The probability density of Y_B.

`/* Compute the ranks of` $A^n \cup B^n$`. The lowest value has rank 0. Assign the same rank to ties without skipping any rank. */`
for *i = 1,...,n* **do**
  $r(a_i) \leftarrow$ rank of a_i in $A^n \cup B^n$
  $r(b_i) \leftarrow$ rank of b_i in $A^n \cup B^n$
$R \leftarrow \{r(a_1), ..., r(a_n), r(b_1), ..., r(b_n)\}$
$r_{max} \leftarrow \max(R)$
for *k = -1,0,1,...,*r_{max} **do**
  $\gamma(k) \leftarrow$ the number of items in R lower than or equal to k
`/* The probability density function of` g_{Y_A} `is represented as a mixture of` n `uniform distributions.` $g_{Y_A}[s]$ `is the probability density of` Y_A `in the interval` $[\frac{s}{2n}, \frac{s+1}{2n})$. */`
$g_{Y_A} \leftarrow$ array of zeros of length $2n$
$g_{Y_B} \leftarrow$ array of zeros of length $2n$
for $c = a_1, ..., a_n, b_1, ..., b_n$ **do**
  $A_{mult} \leftarrow$ number of times that c is in A^n
  $B_{mult} \leftarrow$ number of times that c is in B^n
  for *mult = 1,...,*$(A_{mult} + B_{mult})$ **do**
    $g_{Y_A}[\gamma(r(a_i) - 1) + mult - 1] \leftarrow (n \cdot A_{mult})^{-1}$
    $g_{Y_B}[\gamma(r(b_i) - 1) + mult - 1] \leftarrow (n \cdot B_{mult})^{-1}$
return g_{Y_A}, g_{Y_B}

these estimations will have the same $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ as the quantile random variables Y_A and Y_B. Specifically, the size of the uniform kernels needs to be smaller than $\min_{i,j \in \{1...n\} | a_i \neq b_j} 2|a_i - b_j|$, where $A^n = \{a_1, ..., a_n\}$ and $B^n = \{b_1, ..., b_n\}$ are the n observed samples of A and B respectively. As a result, the $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ of the kernel density estimations will not change when the size of the kernels is reduced below its initial size. This can be deduced from Property 8 in Section 2.2.2, which both $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ satisfy.

The quantile random variables Y_A and Y_B can also be obtained by applying a sequence of transformations to the kernel density estimations (with small uniform kernels) of A and B. Three consecutive transformations are required, none of which modify the $\mathcal{C}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$ due to Property 8. The first transformation involves further

reducing the size of the kernels to $1/(4n)$. Secondly, each kernel k is moved into the position $r(k)/(2n) + (4n)^{-1}$, where $r(k)$ is the rank of the sample in k in $A^n \cup B^n$. In the case of ties, r assigns the same rank to all kernels and this same rank is the average of the previous and the next rank. Since each of the possible positions are at distance $1/(2n)$ from each other, this transformation will not change the $\mathcal{C}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$. Finally, the length of the kernels is increased to $mult/(4n)$, where *mult* is the number of times that the sample defining the kernel is repeated in $A^n \cup B^n$. Note that this increase in the length will in no case cause an overlap of kernels.

7.2.3 $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ in the cumulative difference-plot

In this section, we mathematically prove and experimentally verify that the cumulative difference-plot can be used to deduce $\mathcal{C}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$. First, we describe which estimators are used when these dominance measures are visually estimated from the cumulative difference-plot. Then, we show that these estimators converge to $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ as the number of samples increases.

7.2.3.1 Estimating $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{D}$ from the cumulative difference-plot

Definition 17 *(observations of random variables)*
Let A be a continuous random variable. We define n observations of A as the realizations of the i.i.d random variables $\{A^i\}_{i=1}^n$ that are distributed as A, denoted as $A^n = \{a_i\}_{i=1}^n$.

Definition 18 *(estimation of $\mathcal{C}_\mathcal{P}$)*
Let A and B be two continuous random variables and A^n and B^n their n observations respectively. We define the estimation of the probability that $A < B$ as

$$\widetilde{\mathcal{C}_\mathcal{P}}(A^n, B^n) = \sum_{i,k=1\ldots n} \frac{\text{sign}(b_k - a_i)}{2n^2} + \frac{1}{2}.$$

Definition 19 *(estimation of $\mathcal{C}_\mathcal{D}$)*
Let A and B be two continuous random variables and A^n and B^n their n observations respectively. Let $\{c_j\}_{j=1}^{2n}$ the sorted list of all the observations of A^n and B^n where c_1 is the smallest observation and c_{2n} the largest. Let $\{c_d\}_{d=1}^{d_{max}}$ be the sorted list of unique values in $\{c_j\}_{j=1}^{2n}$. We define the estimation of the dominance rate as

$$\widetilde{\mathcal{C}_\mathcal{D}}(A^n, B^n) = \frac{\sum_{j=1}^{2n} \frac{\psi(c_j)}{2n} + 1}{2} \cdot k_c^{-1}$$

$k_c = \frac{\sum_{j=1}^{2n} \mathcal{I}[\psi(c_j)\neq 0]}{2n}$ *is the normalization constant and* ψ_j *is defined as*

$$\psi(c_d) = \begin{cases} 0 & \text{if } \hat{G}_A(c_{d-1}) = \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) = \hat{G}_B(c_d) \\ 1 & \text{if } \hat{G}_A(c_{d-1}) \geq \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) > \hat{G}_B(c_d) \\ 1 & \text{if } \hat{G}_A(c_{d-1}) > \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) \geq \hat{G}_B(c_d) \\ -1 & \text{if } \hat{G}_B(c_{d-1}) \geq \hat{G}_A(c_{d-1}) \\ & \text{and } \hat{G}_B(c_d) > \hat{G}_A(c_d) \\ -1 & \text{if } \hat{G}_B(c_{d-1}) > \hat{G}_A(c_{d-1}) \\ & \text{and } \hat{G}_B(c_d) \geq \hat{G}_A(c_d) \\ 1 - 2\gamma(c_d) & \text{if } \hat{G}_B(c_{d-1}) > \hat{G}_A(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) > \hat{G}_B(c_d) \\ 2\gamma(c_d) - 1 & \text{if } \hat{G}_A(c_{d-1}) > \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_B(c_d) > \hat{G}_A(c_d) \end{cases}$$

with $\gamma(c_d) = \frac{\hat{G}_B(c_{d-1}) - \hat{G}_A(c_{d-1})}{[B^n = c_d] - [A^n = c_d]}$. *Note that* $[A^n = c_d]$ *counts the number of items in* A^n *equal to* c_d *and* $\hat{G}_A$ *is the empirical distribution [Steck, 1971] estimated from* A^n. *To improve the readability, we abuse the notation and assume that* $\hat{G}_A(c_0) = 0$.

We now show that these estimates can be directly computed from the cumulative difference plot. First, we show that the estimation of $\mathcal{C}_\mathcal{P}$ from the cumulative difference-plot is equivalent to the estimation in Definition 18. As mentioned in Section 2.4.3, the $\mathcal{C}_\mathcal{P}$ estimated from the cumulative difference-plot is $0.5 + \int_0^1 \text{diff}(x)dx$ where diff is the difference function introduced in Equation (2.2). Specifically, the difference function was defined as $\text{diff}(x) = G_{Y_A}(x) - G_{Y_B}(x)$.

Lemma 4 *Let* A *and* B *be two continuous random variables and* A^n *and* B^n *their* n *observations respectively. Then,*

$$\int_0^1 \text{diff}(x)dx = \sum_{j=1}^{2n} \frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n})}{2n}$$

Proof. Considering that the density functions of Y_A and Y_B are constant in each interval $[\frac{j}{2n}, \frac{j+1}{2n})$ for $j = 0, ..., (2n-1)$, we get that

$$\int_{\frac{j}{2n}}^{\frac{j+1}{2n}} \text{diff}(x)dx = \frac{\text{diff}(\frac{j}{2n}) + \text{diff}(\frac{j+1}{2n})}{4n} =$$

$$\frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n}) + G_{Y_A}(\frac{j+1}{2n}) - G_{Y_B}(\frac{j+1}{2n})}{4n}$$

Taking into account that $G_{Y_A}(0) = G_{Y_B}(0) = 0$ and $G_{Y_A}(1) = G_{Y_B}(1) = 1$,

$$\int_0^1 \text{diff}(x)dx = \sum_{j=0}^{2n-1} \int_{\frac{j}{2n}}^{\frac{j+1}{2n}} \text{diff}(x)dx =$$

$$\frac{G_{Y_A}(\frac{0}{2n}) - G_{Y_B}(\frac{0}{2n}) + G_{Y_A}(\frac{2n}{2n}) - G_{Y_B}(\frac{2n}{2n})}{4n} +$$

$$\sum_{j=1}^{2n-1} \frac{2 \cdot G_{Y_A}(\frac{j}{2n}) - 2 \cdot G_{Y_B}(\frac{j}{2n})}{4n} =$$

$$\sum_{j=1}^{2n-1} \frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n})}{2n}$$

Finally, since $G_{Y_A}(1) = G_{Y_B}(1) = 1$, we have that

$$\sum_{j=1}^{2n-1} \frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n})}{2n} =$$

$$\sum_{j=1}^{2n} \frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n})}{2n}$$

Proposition 3 *($\mathcal{C}_\mathcal{P}$ estimated from the cumulative difference-plot)*
s Let A and B be two random variables and A^n and B^n their n observations respectively. Let diff *be the difference function obtained from the samples A^n and B^n as defined in Equation (2.2). Then,*

$$\widetilde{\mathcal{C}_\mathcal{D}}(A^n, B^n) = \int_0^1 \text{diff}(x)dx + \frac{1}{2}$$

Proof. Given the observations A^n and B^n, we need to prove that

$$\sum_{i,k=1...n} \frac{\text{sign}(b_k - a_i)}{2n^2} + \tfrac{1}{2} = \int_0^1 \text{diff}(x)dx + \tfrac{1}{2}$$

With Lemma 4, it is enough to prove that

$$\sum_{i,k=1...n} \frac{\text{sign}(b_k - a_i)}{2n^2} = \sum_{j=1}^{2n} \frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n})}{2n}$$

Let $C_{2n} = \{c_j\}_{j=1}^{2n}$ be the list of all the sorted observations of A^n and B^n where c_1 is the smallest observation and c_{2n} the largest. Then, we have that

$$G_{Y_A}(\frac{j}{2n}) = \frac{[A^n < c_j] + \frac{[A^n = c_j][k \leq j | c_k = c_j]}{[C_{2n} = c_j]}}{n} \text{ and}$$

$$G_{Y_B}(\frac{j}{2n}) = \frac{[B^n < c_j] + \frac{[B^n = c_j][k \leq j | c_k = c_j]}{[C_{2n} = c_j]}}{n}$$

where $[A^n < c_j]$ counts the number of items in A^n lower than c_j, and $[k \leq j | c_k = c_j]$ counts the number of items in C_{2n} equal to c_j but with a lower or equal position in C_{2n}. Therefore, we have that

$$\sum_{j=1}^{2n} \frac{G_{Y_A}(\frac{j}{2n}) - G_{Y_B}(\frac{j}{2n})}{2n} =$$

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] + \frac{[A^n = c_j][k \leq j | c_k = c_j]}{[C_{2n} = c_j]} - [B^n < c_j] - \frac{[B^n = c_j][k \leq j | c_k = c_j]}{[C_{2n} = c_j]}}{2n^2}$$

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j] + \frac{([A^n = c_j] - [B^n = c_j])[k \leq j | c_k = c_j]}{[C_{2n} = c_j]}}{2n^2} \tag{7.4}$$

Now we group the terms in Equation (7.4) into d_{max} groups such that each group contains all the terms with the same c_j, and each group d contains $[C_{2n} = c_d]$ terms, with $c_j = c_d$.

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2} + \sum_{d=1}^{d_{max}} \sum_{c_j} \frac{\frac{([A^n = c_j] - [B^n = c_j])[k \leq j | c_k = c_j]}{[C_{2n} = c_j]}}{2n^2} =$$

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2} + \sum_{d=1}^{d_{max}} \frac{([A^n = c_d] - [B^n = c_d])(([C_{2n} = c_d] + 1) \cdot [C_{2n} = c_d]/2)}{2n^2} =$$

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2} + \sum_{d=1}^{d_{max}} \frac{([A^n = c_d] - [B^n = c_d])([C_{2n} = c_d] + 1)/2}{2n^2} =$$

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2} +$$

$$\sum_{d=1}^{d_{max}} \frac{([A^n = c_d] - [B^n = c_d])([C_{2n} = c_d])/2 + ([A^n = c_d] - [B^n = c_d])/2}{2n^2} =$$

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2} + \sum_{j=1}^{2n} \frac{([A^n = c_j] - [B^n = c_j])/2}{2n^2} +$$

$$\underbrace{\sum_{d=1}^{d_{max}} \frac{([A^n = c_d] - [B^n = c_d])/2}{2n^2}}_{=0} =$$

$$\underbrace{\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2}}_{\text{first summand}} + \underbrace{\sum_{j=1}^{2n} \frac{([A^n = c_j] - [B^n = c_j])/2}{2n^2}}_{\text{second summand}}$$

Focusing on the first summand, we have that

$$\sum_{j=1}^{2n} \frac{[A^n < c_j] - [B^n < c_j]}{2n^2} =$$

$$\frac{\sum_{j=1}^{2n}[A^n < c_j] - \sum_{j=1}^{2n}[B^n < c_j]}{2n^2} =$$

$$\frac{\sum_{j=1}^{2n}\sum_{i=1}^{n}[\{a_i\} < c_j] - \sum_{j=1}^{2n}\sum_{i=1}^{n}[\{b_i\} < c_j]}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}[\{a_i\} < a_k] + \sum_{k=1}^{n}\sum_{i=1}^{n}[\{a_i\} < b_k]}{2n^2} -$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}[\{b_i\} < a_k] + \sum_{k=1}^{n}\sum_{i=1}^{n}[\{b_i\} < b_k]}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}[\{a_i\} < a_k] + [\{a_i\} < b_k] - [\{b_i\} < a_k] - [\{b_i\} < b_k]}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}[\{a_i\} < b_k] - [\{b_i\} < a_k] + [\{a_i\} < a_k] - [\{b_i\} < b_k]}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i) + [\{a_i\} < a_k] - [\{b_i\} < b_k]}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{k=1}^{n}[A^n < a_k] - [B^n < b_k]}{2n^2}$$

From the second summand, we obtain

$$\sum_{j=1}^{2n}\frac{([A^n = c_j] - [B^n = c_j])/2}{2n^2} = \sum_{k=1}^{n}\frac{([A^n = a_k] - [B^n = a_k] + [A^n = b_k] - [B^n = b_k])/2}{2n^2}$$

Combining these summands,

$$\sum_{j=1}^{2n}\frac{[A^n < c_j] - [B^n < c_j]}{2n^2} + \sum_{j=1}^{2n}\frac{([A^n = c_j] - [B^n = c_j])/2}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} +$$

$$\frac{\sum_{k=1}^{n}[A^n < a_k] - [B^n < b_k]}{2n^2} + \frac{\sum_{k=1}^{n}([A^n = a_k] - [B^n = a_k] + [A^n = b_k] - [B^n = b_k])/2}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} +$$

$$\frac{\sum_{k=1}^{n}[A^n \leq a_k] - [B^n \leq b_k]}{2n^2} + \frac{\sum_{k=1}^{n}(-[A^n = a_k] - [B^n = a_k] + [A^n = b_k] + [B^n = b_k])/2}{2n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{k=1}^{n}[A^n \leq a_k] - [B^n \leq b_k]}{2n^2} + \frac{\sum_{k=1}^{n}(-[C_{2n} = a_k] + [C_{2n} = b_k])}{4n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{n(n+1)/2 + \sum_{d=1}^{d_{max}}\frac{[A_n=c_d]^2 - [A_n=c_d]}{2}}{2n^2} -$$

$$\frac{n(n+1)/2 + \sum_{d=1}^{d_{max}}\frac{[B_n=c_d]^2 - [B_n=c_d]}{2}}{2n^2} + \frac{\sum_{k=1}^{n}(-[C_{2n} = a_k] + [C_{2n} = b_k])}{4n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{d=1}^{d_{max}}\frac{[A_n=c_d]^2 - [A_n=c_d]}{2} - \sum_{d=1}^{d_{max}}\frac{[B_n=c_d]^2 - [B_n=c_d]}{2}}{2n^2} +$$

$$\frac{\sum_{k=1}^{n}(-[C_{2n} = a_k] + [C_{2n} = b_k])}{4n^2} =$$

considering that $\sum_{d=1}^{d_{max}}\frac{[B_n=c_d] - [A_n=c_d]}{2} = 0$, we simplify the previous equation to

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{d=1}^{d_{max}}\frac{[A_n=c_d]^2-[B_n=c_d]^2}{2}}{2n^2} + \frac{\sum_{k=1}^{n}(-[C_{2n}=a_k]+[C_{2n}=b_k])}{4n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{d=1}^{d_{max}}[A_n=c_d]^2-[B_n=c_d]^2}{4n^2} + \frac{\sum_{k=1}^{n}(-[C_{2n}=a_k]+[C_{2n}=b_k])}{4n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{d=1}^{d_{max}}[A_n=c_d]^2-[B_n=c_d]^2}{4n^2} + \frac{\sum_{d=1}^{d_{max}}(-[C_{2n}=c_d][A_n=c_d]+[C_{2n}=c_d][B_n=c_d])}{4n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{d=1}^{d_{max}}[A_n=c_d]^2-[B_n=c_d]^2}{4n^2} + \underbrace{\frac{\sum_{d=1}^{d_{max}}[C_{2n}=c_d]([B_n=c_d]-[A_n=c_d])}{4n^2}}_{\text{third summand}} =$$

We expand the third summand,

$$\frac{\sum_{d=1}^{d_{max}}[C_{2n}=c_d]([B_n=c_d]-[A_n=c_d])}{4n^2} =$$

$$\frac{\sum_{d=1}^{d_{max}}([B_n=c_d]+[A_n=c_d])([B_n=c_d]-[A_n=c_d])}{4n^2} = \frac{\sum_{d=1}^{d_{max}}([B_n=c_d]^2-[A_n=c_d]^2)}{4n^2}$$

Finally,

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2} + \frac{\sum_{d=1}^{d_{max}}[A_n=c_d]^2-[B_n=c_d]^2}{4n^2} + \frac{\sum_{d=1}^{d_{max}}([B_n=c_d]^2-[A_n=c_d]^2)}{4n^2} =$$

$$\frac{\sum_{k=1}^{n}\sum_{i=1}^{n}\text{sign}(b_k - a_i)}{2n^2}$$

Proposition 4 *Let A and B be two random variables and A^n and B^n their n observations respectively. The $\mathcal{C}_\mathcal{D}$ estimated from the cumulative difference-plot is $\widehat{\mathcal{C}_\mathcal{D}}$.*

Proof. In Section 2.4.3, we defined the $\mathcal{C}_{\mathcal{D}}$ estimated from the cumulative difference-plot as

$$\mathcal{C}_{\mathcal{D}} = \frac{\frac{\int_0^1 \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx}{2} + \frac{1}{2}}{\int_0^1 \mathcal{I}[\text{diff}(x) \neq 0]dx},$$

where $\mathcal{I}$ is the indicator function. This proposition claims that

$$\frac{\frac{\int_0^1 \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx}{2} + \frac{1}{2}}{\int_0^1 \mathcal{I}[\text{diff}(x) \neq 0]dx} = \frac{\sum_{j=1}^{2n} \frac{\psi(c_j)}{2n} + 1}{2} \cdot k_c^{-1}.$$

To prove it, we show that

i) $\int_0^1 \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx = \sum_{j=1}^{2n} \frac{\psi(c_j)}{2n}$

and

ii) $\int_0^1 \mathcal{I}[\text{diff}(x) \neq 0]dx = k_c$.

Let us focus our attention in *i*). We split the integral into $2n$ parts:

$$\int_0^1 \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx =$$
$$\sum_{j=1}^{2n} \int_{\frac{j-1}{2n}}^{\frac{j}{2n}} \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx \tag{7.5}$$

Let $C_{2n} = \{c_j\}_{j=1}^{2n}$ be the list of all the sorted observations of A^n and B^n where c_1 is the smallest observation and c_{2n} the largest and let $\{c_d\}_{d=1}^{d_{max}}$ be the sorted list of unique values in C_{2n}. We group the terms in the sum of Equation (7.5) into d_{max} groups such that for every j in a group, $c_j = c_d$.

$$\sum_{d=1}^{d_{max}} \sum_{j} \int_{\frac{j-1}{2n}}^{\frac{j}{2n}} \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0]dx$$

Now we join the integrals for every j in each group, such that the j of the integral goes from $j_{d\downarrow} - 1$ to $j_{d\uparrow}$ (if the sample c_d is unique in C_{2n}, then $j_{d\downarrow} = j_{d\uparrow} = j$).

$$\sum_{d=1}^{d_{max}} \int_{\frac{j_{d\downarrow}-1}{2n}}^{\frac{j_{d\uparrow}}{2n}} \mathcal{I}[\text{diff}(x) > 0] - \mathcal{I}[\text{diff}(x) < 0] dx \tag{7.6}$$

In the interval $(\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$, diff evaluates to one of these four possibilities:

1. $\text{diff}(x) = 0$ for all $x \in (\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$
2. $\text{diff}(x) > 0$ for all $x \in (\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$
3. $\text{diff}(x) < 0$ for all $x \in (\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$
4. $\text{diff}(x) = 0$ in one point in the interval $(\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$ and $\text{diff}(x) > 0$ or $\text{diff}(x) < 0$ for every other x in the interval. However, we can safely ignore this point as the value of the integral is invariant to the value of the function in sets of zero measure.

By looking at the empirical distributions $\hat{G}_A(x)$ and $\hat{G}_B(x)$ estimated from A^n and B^n respectively, we can guess which of these possibilities corresponds to each interval.

$$\begin{cases} 1) & \text{if } \hat{G}_A(c_{d-1}) = \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) = \hat{G}_B(c_d) \\ 2) & \text{if } \hat{G}_A(c_{d-1}) \geq \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) > \hat{G}_B(c_d) \\ 2) & \text{if } \hat{G}_A(c_{d-1}) > \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) \geq \hat{G}_B(c_d) \\ 3) & \text{if } \hat{G}_B(c_{d-1}) \geq \hat{G}_A(c_{d-1}) \\ & \text{and } \hat{G}_B(c_d) > \hat{G}_A(c_d) \\ 3) & \text{if } \hat{G}_B(c_{d-1}) > \hat{G}_A(c_{d-1}) \\ & \text{and } \hat{G}_B(c_d) \geq \hat{G}_A(c_d) \\ 4) & \text{if } \hat{G}_B(c_{d-1}) > \hat{G}_A(c_{d-1}) \\ & \text{and } \hat{G}_A(c_d) > \hat{G}_B(c_d) \\ 4) & \text{if } \hat{G}_A(c_{d-1}) > \hat{G}_B(c_{d-1}) \\ & \text{and } \hat{G}_B(c_d) > \hat{G}_A(c_d) \end{cases}$$

The value of the integral in Equation (7.6) corresponding to these possibilities are the following:

1. 0
2. $[C_{2n} = c_d] \cdot \frac{1}{2n}$
3. $-[C_{2n} = c_d] \cdot \frac{1}{2n}$
4. $[C_{2n} = c_d] \cdot (2 \cdot l_d - 1) \cdot \frac{1}{2n}$

where $[C_{2n} = c_d]$ counts the number of items in C_{2n} equal to c_d and l_d is the proportion in which $\text{diff}(x) > 0$ in the interval $(\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$. For example, $l_d = 0.75$ would represent that $\text{diff}(x) > 0$ in 75% of the total length of the interval, and $\text{diff}(x) < 0$ in the other 25%.

With this, we can rewrite Equation (7.6) as

$$\sum_{d=1}^{d_{max}} [C_{2n} = c_d] \cdot \psi(c_d) \cdot \frac{1}{2n} = \sum_{j=1}^{2n} \frac{\psi(c_j)}{2n},$$

where ψ is the function introduced in Definition 19.

Now, we only need to prove *ii*). Specifically, we need to show that

$$\int_0^1 \mathcal{I}[\text{diff}(x) \neq 0]dx = k_c.$$

We have that

$$\int_0^1 \mathcal{I}[\text{diff}(x) \neq 0]dx = \sum_{d=1}^{d_{max}} \int_{\frac{j_{d\downarrow}-1}{2n}}^{\frac{j_{d\uparrow}}{2n}} \mathcal{I}[\text{diff}(x) \neq 0]dx,$$

and

$$k_c = \frac{\sum_{j=1}^{2n} \mathcal{I}[\psi(c_j) \neq 0]}{2n} = \sum_{d=1}^{d_{max}} [C_{2n} = c_d] \frac{\mathcal{I}[\psi(c_d) \neq 0]}{2n}.$$

Finally, it is easy to see that

$$\int_{\frac{j_{d\downarrow}-1}{2n}}^{\frac{j_{d\uparrow}}{2n}} \mathcal{I}[\text{diff}(x) \neq 0]dx = [C_{2n} = c_d] \frac{\mathcal{I}[\psi(c_d) \neq 0]}{2n},$$

because $\text{diff}(x) = 0$ in the interval $(\frac{j_{d\downarrow}-1}{2n}, \frac{j_{d\uparrow}}{2n})$ if and only if $\psi(c_d) = 0$.

7.2.3.2 Convergence of the estimators

Proposition 5 *Let A and B be two continuous random variables and $\{a_i\}_{i\in\mathbb{N}}$ and $\{b_i\}_{i\in\mathbb{N}}$ be two infinite sequences of their observations respectively. Let A^n and B^n be the two finite subsequences that contain the first n elements of $\{a_i\}_{i\in\mathbb{N}}$ and $\{b_i\}_{i\in\mathbb{N}}$ respectively. Then,*

$$\mathcal{C}_{\mathcal{P}}(A,B) = \lim_{n\to\infty} \widetilde{\mathcal{C}_{\mathcal{P}}}(A^n,B^n)$$

Proof. Let $\{P_s\}_{s\in\mathbb{N}}$ be a sequence of estimators with every estimator is determined randomly with the following procedure:

1) generate two random permutations σ_s and τ_s of size n.

2) define each estimation as

$$P_s(A^n,B^n) = \sum_{i=1}^{n} \frac{\text{sign}(b_{\sigma_s(i)} - a_{\tau_s(i)})}{2n} + \frac{1}{2}.$$

It is easy to see that each P_s is an estimator of $\mathcal{P}(A < B)$ (since A, B are continuous, we know that $\mathcal{P}(A = B) = 0$). Now observe that the sequence $\left\{\frac{\sum_{t=1}^{s} P_t(A^n,B^n)}{s}\right\}_{n\in\mathbb{N}}$ converges to $\widetilde{\mathcal{C}_{\mathcal{P}}}(A^n,B^n) = \sum_{i,k=1\ldots n} \frac{\text{sign}(b_k - a_i)}{2n^2} + \frac{1}{2}$, which means that $\widetilde{\mathcal{C}_{\mathcal{P}}}(A^n,B^n)$ is also an estimator of $\mathcal{P}(A < B)$.

Unfortunately, the estimator $\widetilde{\mathcal{C}_{\mathcal{D}}}$ will not always converge: $\mathcal{C}_{\mathcal{D}}$ fails to satisfy Property 7, and this means that a a few points can still have a big impact in the estimation of $\mathcal{C}_{\mathcal{D}}$. Specifically, given the continuous random variables A and B defined in N, $\widetilde{\mathcal{C}_{\mathcal{D}}}$ will converge iff $\int_N \mathcal{I}[G_A(x) = G_B(x)] \cdot (g_A + g_B)dx = 0$.

Luckily, this lack of convergence is not a problem when the estimation of $\mathcal{C}_{\mathcal{D}}$ is carried out visually in the cumulative difference-plot. Since the visual representation of the cumulative difference-plot involves rendering the plot with pixels, there exists an small $\delta > 0$ such that when $|\text{diff}(x)| < \delta$, the difference is displayed as 0.

In practice, we do not even need to account for the case that $\text{diff}(x) = 0$. The cumulative difference-plot models the uncertainty with a confidence band, and when $\text{diff}(x) = 0$ is inside the confidence band, then so are $\text{diff}(x) > 0$ and $\text{diff}(x) < 0$. If we assume that the difference is positive, negative or zero every time that $\text{diff}(x) = 0$ is inside the confidence band, we obtain the estimations $\widetilde{\mathcal{C}_{\mathcal{D}}}^{+}$, $\widetilde{\mathcal{C}_{\mathcal{D}}}^{-}$ and $\widetilde{\mathcal{C}_{\mathcal{D}}}^{0}$ respectively. Now since $\widetilde{\mathcal{C}_{\mathcal{D}}}^{+} > \widetilde{\mathcal{C}_{\mathcal{D}}}^{=} > \widetilde{\mathcal{C}_{\mathcal{D}}}^{-}$, the estimation of $\mathcal{C}_{\mathcal{D}}$ with the highest part of the confidence band is an upper bound of $\mathcal{C}_{\mathcal{D}}$. The

same is true for the estimation with the lowest part of the confidence band: it is a lower bound of $\mathcal{C}_{\mathcal{D}}$.

Although $\widetilde{\mathcal{C}_{\mathcal{D}}}$ does not converge to $\mathcal{C}_{\mathcal{D}}$, for any $\epsilon > 0$ we can find a δ small enough such that the difference between $\widetilde{\mathcal{C}_{\mathcal{D}}}^{\delta}$ and $\mathcal{C}_{\mathcal{D}}$ is smaller than ϵ. We formalize this claim in Conjecture 1, and we leave the proof for future work.

Definition 20 *(δ-estimation of $\mathcal{C}_{\mathcal{D}}$)*
Let A and B be two continuous random variables and A^n and B^n their n observations respectively. Let $\{c_j\}_{j=1}^{2n}$ the sorted list of all the observations of A^n and B^n where c_1 is the smallest observation and c_{2n} the largest. Let $\{c_d\}_{d=1}^{d_{max}}$ be the sorted list of unique values in $\{c_j\}_{j=1}^{2n}$.

We define the δ-estimation of $\mathcal{C}_{\mathcal{D}}$, denoted as $\widetilde{\mathcal{C}_{\mathcal{D}}}^{\delta}$, as the same estimation as $\widetilde{\mathcal{C}_{\mathcal{D}}}$, but assuming that the empirical distributions computed from A^n and B^n are equal when $|\hat{G}_A(x) - \hat{G}_B(x)| < \delta$.

The previous definition can also be based in the δ-difference, defined as $\text{diff}^{\delta}(x) = \text{diff}(x)$ when $\text{diff}(x) \geq \delta$, and $\text{diff}^{\delta}(x) = 0$ otherwise.

Conjecture 1 *Let A and B be two continuous random variables and $\{a_i\}_{i\in\mathbb{N}}$ and $\{b_i\}_{i\in\mathbb{N}}$ be two infinite sequences of their observations respectively. Let A^n and B^n be the two finite subsequences that contain the first n elements of $\{a_i\}_{i\in\mathbb{N}}$ and $\{b_i\}_{i\in\mathbb{N}}$ respectively. Then, for all $\epsilon > 0$, there exists a $\delta > 0$ such that*

$$\left| \mathcal{C}_{\mathcal{D}}(A, B) - \lim_{n\to\infty} \widetilde{\mathcal{C}_{\mathcal{D}}}^{\delta}(A^n, B^n) \right| < \epsilon$$

7.2.3.3 Experimental verification

In the following, we experimentally verify that the cumulative difference-plot can be used to deduce $\mathcal{C}_{\mathcal{D}}$ and $\mathcal{C}_{\mathcal{P}}$. To do so, we define six pairs of example random variables and measure the $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ with three different methods: the definition of $\mathcal{C}_{\mathcal{D}}$ and $\mathcal{C}_{\mathcal{P}}$ (Equation (2.1) and Definition 12), the estimators in Definitions 18 and 19 and from the *cumulative difference-plot*. The *cumulative difference-plot* has a confidence band in addition to the estimation, and this confidence band allows the lower and upper bounds of $\mathcal{C}_{\mathcal{D}}$ and $\mathcal{C}_{\mathcal{P}}$ to be computed.

The probability density functions of the six examples are shown in Figures 7.5 through 7.10. The probability density of these random variables is a mix of normal distributions, the beta distribution, and the log-normal distribution.

The difference plot and the estimations were carried out with 5000 samples from each random variable. The $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{D}}$ values computed are shown in Figures 7.11

and 7.12 respectively. In every case, the estimations with the three methods match, except for $\mathcal{C}_{\mathcal{D}}$ in Example 4 (Figure 7.8). This is a deceptive example because, in most of the probability mass of A and B, the cumulative distribution functions are equal. Consequently, in this example, the estimator of $\mathcal{C}_{\mathcal{D}}$ introduced in Definition 19 is unstable: it is very likely that the estimated empirical distributions are different even though the cumulative distribution functions are identical. Overcoming this limitation involves choosing a small $\delta > 0$, such that when the difference between the empirical distributions is smaller than δ, they are considered equal.

We conclude that, in most cases, the three estimation methods (from densities, using the estimators and with the cumulative difference-plot) yield a similar result, which validates the statements in the previous section.

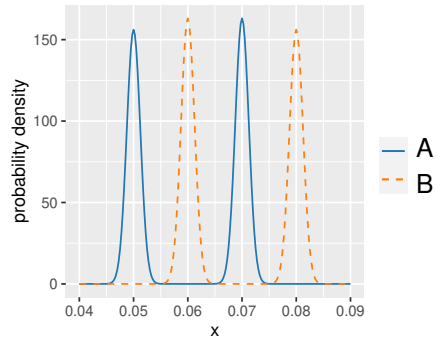


Fig. 7.5: Probability density functions of Example 1

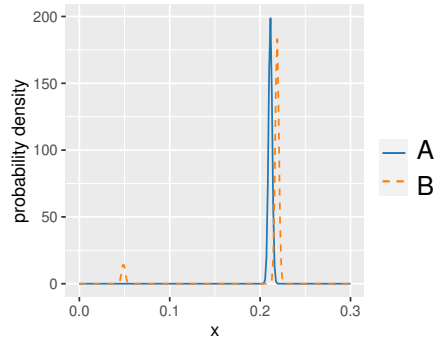


Fig. 7.6: Probability density functions of Example 2

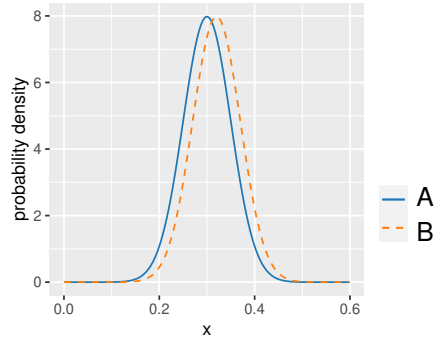


Fig. 7.7: Probability density functions of Example 3

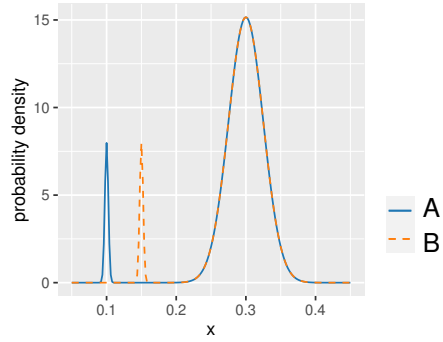


Fig. 7.8: Probability density functions of Example 4

7.3 Chapter 4

7.3.1 Neuroevolution of Augmenting Topologies

In this chapter, the neural networks were trained with a reinforcement learning, neuroevolution algorithm. More specifically, neuroevolution of augmenting topologies (NEAT) has been used[1]. Introduced by Stanley and Miikkulainen [2002],

[1] The source code, along with all the experimentation is available at the GitHub repository https://github.com/EtorArza/TransfHH. The neuroevolution algorithm was taken from the accneat package, with a few minor changes made to it. Accneat is a fork of the implementation by Stanley and Miikkulainen [2002] with some improvements, such as delete mutations and speed improvements, available at

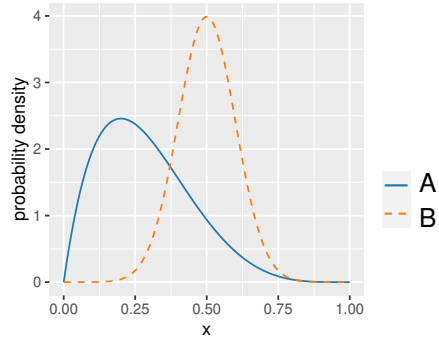


Fig. 7.9: Probability density functions of Example 5

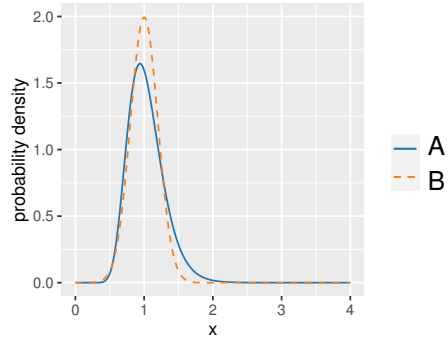


Fig. 7.10: Probability density functions of Example 6

NEAT is a genetic algorithm for neural networks that can evolve not only the parameters (weights) of a neural network but also its structure.

NEAT is one of the most influential approaches in the field of neuroevolution to date. In fact, before the introduction of NEAT, most of the neuroevolution techniques without a fixed topology suffered from the competing conventions problem [Schaffer et al., 1992]. When two networks have similar functionality but are entirely different, and thus, incompatible structures, we say that these two networks contain competing conventions. NEAT is a revolutionary neuroevolution technique because it defined a new way to efficiently handle the competing conventions problem through a novel speciation mechanism. Specifically, each time

https://github.com/sean-dougherty/accneat. The code provided alongside this chapter also uses parts of other software projects, see the LICENCE file in the repository for a comprehensive list.

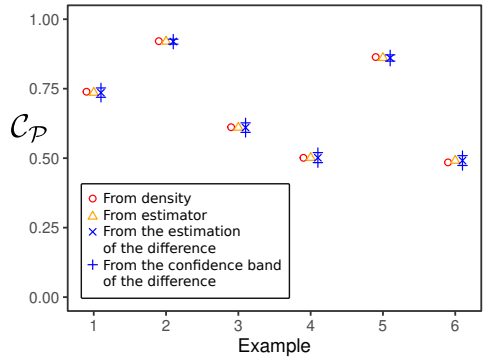


Fig. 7.11: The $\mathcal{C}_{\mathcal{P}}$ values obtained in the six examples with the three methods.

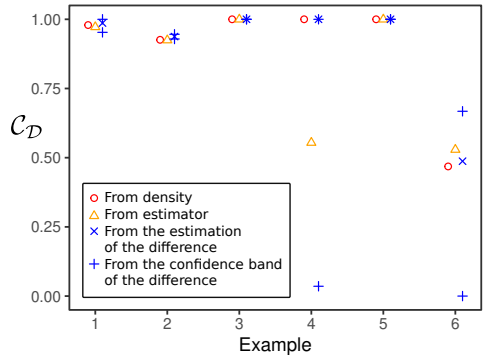


Fig. 7.12: The $\mathcal{C}_{\mathcal{D}}$ values obtained in the six examples with the three methods.

NEAT introduces a new structure due to a mutation, this new structure is assigned a unique innovation number. Then, these innovation numbers are used to 1) select individuals that are sufficiently similar to each other for crossover and 2) only perform crossover in the parts of the structure that have the same innovation numbers.

While early studies on the subject suggest that fixed topology networks perform worse than their counterparts [Stanley and Miikkulainen, 2004], there are new promising alternatives that have been found to outperform NEAT even with a fixed topology [Gomez et al., 2008, Pagliuca et al., 2018]. New trends in neuroevolution include integrating the Covariance Matrix Adaptation metaheuristic to the framework of neuroevolution [Moriguchi and Honiden, 2012], designing the architecture of deep neural networks with evolutionary algorithms [Sun et al., 2019, Fernandes Junior and Yen, 2019] and HyperNeat [Stanley et al., 2009], in which

a two dimension real function is learned that represents the weights of a neural network, achieving a more compact representation that takes into account the position of the neural network links. In this chapter, the NEAT algorithm with node and edge removal is used for two reasons: 1) NEAT is a simple yet well-tested neuroevolution algorithm, and 2) the component removal allows the network to adapt its size to the time consumed in the evaluation of the neural network.

To achieve this, one only needs to set a maximum runtime as the stopping criterion for the hyper-heuristic solver during the training step and make sure the evaluation time of the solutions is negligible with respect to the evaluation time of the network. The preference towards smaller networks is achieved implicitly: smaller networks are processed faster, and so the hyper-heuristic performs more function evaluations in the same amount of time, possibly obtaining better performance.

7.3.2 Detailed explanation of the *modify* process

This appendix includes a complete explanation of how the function *modify* (line 14 in Algorithm 2 in the main manuscript) modifies a solution σ_i. Algorithm 8 shows the pseudo-code of how this modification is applied in detail. The modification towards a reference (lines 2-3) tries to modify the permutation into another permutation that is one unit of distance nearer the reference [Schiavinotto and Stützle, 2007]. The distance is defined for each operator as the minimum number of applications of that operator to transform one permutation into the other permutation [Schiavinotto and Stützle, 2007]. A modification away from the reference is defined similarly (line 5), but choosing the parameters of the operator such that the distance between the reference permutation and the modified permutation is increased by one unit. Finally, after the modification is applied (line 7), the objective value of the solution is measured, and a worse candidate is only accepted with probability $1 - prob\_accept\_worse$ (lines 8-10).

Algorithm 8: modify(σ_i, σ_{ref}, direction, operator, prob_accept_worse)

Input:
σ_i: The permutation to be modified.
σ_{ref}: The reference permutation.
direction: Direction of the modification.
operator: Operator of the modification.
prob_accept_worse: Probability of accepting a worse solution.

```
1  if direction = 1 then
2  |  create a minimum length path from σ_i to σ_ref, defined as the minimum
   |   sequence of applications of operator to solution σ_i that are required to
   |   obtain solution σ_ref (As explained by Schiavinotto and Stützle [2007]).
3  |  operator_params ← parameters of the first operator in the minimum length
   |   path
4  else if direction = −1 then
5  |  operator_params ← parameters of operator that, if applied to σ_i, would
   |   increase the distance between σ_i and σ_ref by one. The distance is defined
   |   as the minimum required applications of operator to transform σ_i into σ_ref.
6  end
7  σ_cand ← create candidate solution by applying operator once with parameters
     operator_params to solution σ_i
8  if objective value σ_i is better than objective value σ_cand then
9  |  with probability (1 - prob_accept_worse)
10 |        σ_cand ← σ_i
11 end
12 return σ_cand
```

7.3.3 Transferability

In this appendix, we formally define the transferability and give additional details on how to generate the embedding based on the performance of the hyper-heuristic. For additional details, we refer the interested reader to the GitHub repository with the source code.

7.3.4 Transferability matrix

We begin by introducing some notation to formally define this concept.

Notation 1 *Controller trained in a problem*
Let $\mathfrak{P}_1$ be an optimization problem and φ a controller. We denote that φ was trained in problem $\mathfrak{P}_1$ as φ_1.

Definition 21 *Performance of a controller*
Let $\mathfrak{P}_1, \mathfrak{P}_2$ be two optimization problems and φ_1 a controller trained in optimization problem $\mathfrak{P}_1$. We define the performance of φ_1 on $\mathfrak{P}_2$, as the result of optimizing $\mathfrak{P}_2$ with the proposed hyper-heuristic algorithm, using φ_1 as the controller and we denote it as $\varphi_1(\mathfrak{P}_2)$.

Definition 22 *Transferability*
Let $\mathfrak{P}_1, ..., \mathfrak{P}_k$ be k maximization problems[1] and φ_i, φ_j two controllers trained in the problems $\mathfrak{P}_i, \mathfrak{P}_j$ respectively. We define the transferability from problem $\mathfrak{P}_i$ to problem $\mathfrak{P}_j$, $T(\mathfrak{P}_i \to \mathfrak{P}_j)$, as the normalized rank in performance that the hyper-heuristic guided by controller φ_1 has in problem $\mathfrak{P}_2$. Formally,

$$T(\mathfrak{P}_i \to \mathfrak{P}_j) = \frac{r_{i,j}}{k-1}$$

where $r_{i,j}$ is the ranking (best to worst) of $\varphi_i(\mathfrak{P}_j)$ in $\{\varphi_t(\mathfrak{P}_j)\}_{t=1,...,k}$. In other words, $T(\mathfrak{P}_i \to \mathfrak{P}_j)$ is assigned a rank proportional to how well controller φ_i does on problem $\mathfrak{P}_j$ with respect to the performance of the rest of the controllers in the same problem. The transferability $T(\mathfrak{P}_i \to \mathfrak{P}_j)$ is 0 if φ_i is the controller with the best performance in $\mathfrak{P}_j$ and 1 if it is the worst performing.

Notation 2 *Transferability Matrix*
Let $\mathfrak{P}_1, ..., \mathfrak{P}_k$ be k maximization problems. We denote the transferability matrix as T where each element $T_{i,j}$ is $T(\mathfrak{P}_i \to \mathfrak{P}_j)$.

The transferability matrix can be further improved by reordering the problems in problem set. Specifically, we reorder the optimization problems such that the loss

$$L = \sum_{j=1}^{k} \sqrt{\sum_{i=1}^{k-1} (T_{i,j} - T_{i+1,j})^2} + \sum_{i=1}^{k} \sqrt{\sum_{j=1}^{k-1} (T_{i,j} - T_{i+1,j})^2}$$

is minimized. Minimizing this loss implies that adjacent columns and rows in the transferability matrix are as similar as possible to each other.

7.3.5 Repeated measurements and noise

It is important to measure the transferability several times and average the results. The reason is that it is very likely that stochastic optimization algorithms will

[1] We assume that all problems considered are maximization problems. Specifically, minimization problems have been transformed into maximization problems by redefining the problem as a maximization of the objective function when multiplied by -1.

get different scores each time they are tested. Without this repetition step, it is possible to observe transferability, when in reality, is just due to the random nature of the optimization algorithms. This is better understood with the following example.

Let us assume that the transferability of a random search algorithm is measured in problem set i). Since the random search algorithm has no learning phase, the score will be random each time. When we measure the transferability only once, we get what is shown in Figure 7.13a. Notice that $T_{3,2} = 0$. From that, it would be erroneous to conclude that the problem $\mathfrak{P}_3$ is good at training the random search algorithm for problem $\mathfrak{P}_2$. In fact, $T_{3,2} = 0$ has the same probability as $T_{i,2} = 0$, for all $i = 1, ..., 12$.

On the other hand, if we average 10 repeated measures of the transferability, we obtain the results shown in Figure 7.13b. In this figure, we see that most of the values are near 0.5. The interpretation is that there was no transferability: it does not matter which instance is used during the training step. In short, several repeated measurements overcome the limitation of observing transferability in stochastic algorithms when there should be none. For this reason, in the rest of the chapter, we averaged 10 measurements of the transferability.

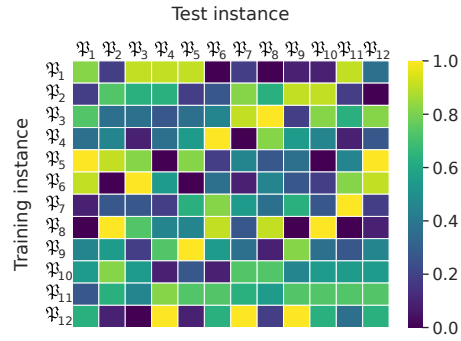
7.3.5.1 Related work

Hong et al. [Hong et al., 2018] proposed measuring the transferability as the average objective value of the hyper-heuristic when training and testing two *problem classes*. Their experimental approach is different, as they study the *transferability among problem classes* instead of *transferability among problems*. They use *problem class* to refer to an infinite set of optimization problems parameterized by one or more real values (for example, $f(x) = ax^2$, with the parameter $a \in [1, 2]$).

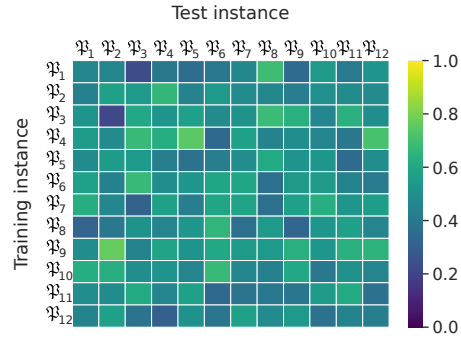
Our experimental setup on transferability improves the methodology of Hong et al. [Hong et al., 2018] in four key aspects. Firstly, by defining the transferability with ranks instead of objective values, we can compare the values across different problems. When objective values are used directly, the differences are not comparable[1].

Secondly, by repeating several measurements of the transferability and computing the average ranks, we can distinguish between noise and the actual difference in performance (this is not possible with average objective values). Additional details on *repeated measurements* are given in Appendix 7.3.5.

[1] For example, let us assume we have the optimization problems $f_1(x) = x^2$ and $f_2(x) = 10^{10} \cdot x^2$ both defined in the interval $[0, 1]$. Given two solutions, if the difference in objective value is 0.5, in problem f_1 it would be considered a bigger difference than in problem f_2.



(a) One measurement.



(b) Ten measurements.

Fig. 7.13: The transferability of a random search algorithm with one measurement (a) and with ten averaged measurements (b).

Thirdly, we experimentally show that the analysis carried out with our approach is correlated with the properties of the optimization problems, which suggests that the proposed technique is useful to find similarities and differences in the properties of optimization problems. Finally, we show that our approach works in both the combinatorial and continuous domains, while the approach of Hong et al. [2018] was only shown to work in the continuous domain.

7.3.6 Response

In the previous section, we proposed an embedding for a set of optimization problems via the performance of the hyper-heuristic. In this section, we propose another embedding by focusing instead on the behavior of the hyper-heuristic.

7.3.6.1 Defining the response

The behavior of the hyper-heuristic, *how* it performs the optimization, is determined by every response Y (the output) of the neural network, generated during the optimization: the decoder modifies each solution according to its corresponding response Y. Therefore, it makes sense to summarize the behavior of the hyper-heuristic in an optimization problem by averaging every response Y that the neural network produces when solving the problem.

Definition 23 *Response*
Let $\mathfrak{P}_1, \mathfrak{P}_2$ be two problems. We define the response from $\mathfrak{P}_1$ to $\mathfrak{P}_2$, $R(\mathfrak{P}_1 \to \mathfrak{P}_2)$, as the average of all response Y generated[1] when solving problem $\mathfrak{P}_2$, with a neural network obtained by training the hyper-heuristic in problem $\mathfrak{P}_1$. The response can be interpreted as a summary of the behavior of the hyper-heuristic when solving problem $\mathfrak{P}_2$ with a controller trained in problem $\mathfrak{P}_1$.

The response is a random variable in the sense that two independent computations of the response will not produce the same result. the response depends on two factors that cannot be fixed or predicted, the first one a) is related to the training stage, and the second b), is related to the testing stage. Firstly a), two executions of the neuroevolution algorithm on the same problem will (almost) never produce two identical neural networks. Secondly b), due to the probabilistic nature of the hyper-heuristic, two executions on the same problem with the same neural network are likely to diverge into a different final solution. This is in part due to the random initialization of the population. Luckily, variability regarding b) is easy to overcome by averaging the response of many executions of the hyper-heuristic in the same test problem.

[1] The response-hyper-heuristic behavior mapping is not always injective, and depends on the implementation of the decoder. The decoder for permutation problems proposed in this chapter is not injective. In other words: two controllers with different responses may produce the same behavior in the decoder. A possible solution is to transform the output so that injectivity is preserved. These transformations are simple functions, such as the indicator function $\chi_{(0.25,1]}$. For more details on the exact functions used to reduce the duplicity, we refer the interested reader to the source code provided.

The variability introduced during training a) is determined by the random seed used to train the neural network. Therefore, each observation of the response can be notated as $R(i, j, s)$ where i denotes the training problem, j denotes the test problem, and s denotes the random seed. An observation of the response, hence, averages several measurements of the response in the same test problem.

To find out what determines the variability in response, we measured the average distance between two responses

$$||R(i_1, j_1, s_1) - R(i_2, j_2, s_2)||_1$$

in these four cases:

1. $i_1 = i_2$, $j_1 \neq j_2$ and $s_1 = s_2$
2. $i_1 = i_2$, $j_1 \neq j_2$ and $s_1 \neq s_2$
3. $i_1 \neq i_2$, $j_1 = j_2$ and $s_1 \neq s_2$
4. $i_1 \neq i_2$, $j_1 \neq j_2$ and $s_1 \neq s_2$

for the four problem sets i) - iv) as defined in Section IV-A.

As seen in Table 7.4 case 1), the distance is really small when the same training problem & training seed is used (even if tested in different problems). In addition, the distance is much larger in the rest of the cases. The interpretation is that, given the training problem i and the random seed s, the response of the hyper-heuristic is determined (disregarding some small variability associated with the test instance j).

Consequently, it makes sense to model the response as a function $R(i, s) = k^{-1} \sum_{j=1}^{k} R(i, j, s)$. For each problem i, we have 10 samples of the behavior of the hyper-heuristic $R(i, s)$ in that problem (one for each seed s). Notice that the response modeled like this depends on the training problem. Now we want to see if there is any correlation between the behavior of the hyper-heuristic and the optimization problem used to train it, regardless of the training seed.

A naive approach to achieve this would be to average all the responses in one problem $R(i) = 10^{-1} \sum_{s=1}^{10} R(i, s)$ and embed every $R(i)$ into $\mathbb{R}^2$ with principal component analysis [Wold et al., 1987]. However, this can hide the differences in behavior of the hyper-heuristics. A possible explanation is that $R(i_1, s_1)$ and $R(i_1, s_2)$ are almost as different from each other as $R(i_2, s_1)$ and $R(i_1, s_2)$ (see Table 7.4), with $i_1 \neq i_2$ and $s_1 \neq s_2$.

Linear Discriminant Analysis [Singh, 2020-08-18, 2020, Rao, 1948] (LDA) overcomes this limitation. First, we fit the LDA with every $R(i, s)$ and set i as the class. The LDA will try to maximize the distance between every two $R(i, s)$ with

Average L1 distance between two responses

Case	1)	2)	3)	4)
Train problem	same	same	different	different
Test problem	different	different	same	different
Training seed	same	different		
Problem set				
i)	0.050	0.436	0.516	0.520
ii)	0.004	0.371	0.367	0.370
iii)	0.174	2.718	2.832	2.812
iv)	0.181	3.172	3.122	3.182

Table 7.4: Average distance between two responses trained in the same problem (with or without the same seed) and tested in the same problem.

different i and minimize the distance between every two $R(i, s)$ with the same i. Once we have fitted the LDA, we project every $R(i)$ in the embedded space.

Notice that the LDA takes into account which training problem was used to produce each $R(i, s)$, but it gets no information about the properties of the training problem i: every problem is equal for the LDA. This is an important point because it means that a projection obtained from the LDA does not require this information, and thus, it can be used in the clustering of unlabeled optimization problems or even the classification of previously unseen optimization problems.

7.3.7 Choosing the hyperparameters

In this Appendix, we justify each of the hyperparameters of the hyper-heuristic (shown in the box below).

Hyper-Heuristic parameters

- Population size: 8
- Stopping criterion: 400 evaluations

Training parameters (NEAT)

- Population size: 10^3 (default value [Dougherty, 2014])
- Stopping criteria: 4 days or 2000 generations (stop when either criterion is met)

Testing parameters

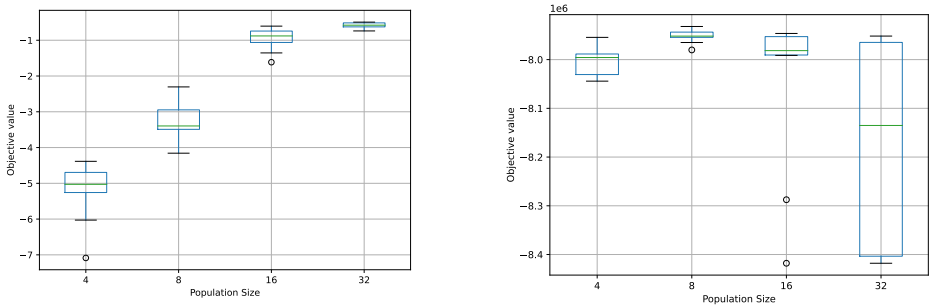
- Executions averaged: 10^4

Hyper-Heuristic parameters: Population size: 8

We conducted an experiment to choose a suitable population size. We trained and tested the hyper-heuristic on the continuous problem $\mathfrak{P}_1$[1], as well as on the Quadratic Assignment Problem instance *Tai60a* from the QAPlib [Burkard et al., 1997], with the hyper-heuristic's population size set to 4, 8, 16, and 32. We repeated the experiment 10 times and show the results in Figure 7.14 below.

In the continuous problem, the best performing population size was 32, which is also the worst performing population size in the combinatorial problem. This suggests that there is no one size fits all for the population size of the hyper-heuristic, each problem might have a different optimal population size. We believe that the purpose of this chapter is not to maximize the objective value in each problem, instead, we want to experiment on the hyper-heuristic as a framework that is applicable in different domains. Following this idea, we think it is desirable to use the same parameters for all problems. A population size of 16 has the second best performance in both problems, however, it has very bad performing outliers in the combinatorial problem. Therefore, we choose a population size of 8, which is the best performing population size for the combinatorial problem.

[1] The objective function of problem $\mathfrak{P}_1$ is $f(x_1, ..., x_{20}) = \sum x_i^2$ as defined in Section IV-A in the main manuscript



(a) Continuous problem $\mathfrak{P}_1$

(b) Permutation problem *Tai60a*

Fig. 7.14: Objective value of the hyper-heuristic when trained and tested in the same problem. The experiment was repeated 10 times with different seeds. Higher is better.

Hyper-Heuristic parameters: Stopping criterion: 400

A stopping criterion of 400 evaluations with a population size of 8 implies 50 generations, which we think is a reasonable amount. A higher number of evaluations obviously has associated a better objective value (the performance can only increase with additional evaluations), hence a parameter tuning makes no sense with this parameter.

Note that a stopping criterion that is either too small or too large will create problems in the problem analysis methodology. For example, if we use 8 as the stopping criterion, the hyper-heuristic will only do one generation, which means that the algorithm is basically random search (with a budget of 8 solutions). This means that there is no learning to be done, and thus the problem analysis methods will not work. Similarly, with an hypothetical stopping criterion of ∞ solutions, even random search will find the optimal value, and the problem analysis will also fail in this case, as no learning is necessary.

Based on the experimental results on the problem analysis methods, we can empirically say that a stopping criterion of 400 works well.

Training parameters (NEAT): Population size: 10^3

This is the default value, which seems to work well. We think that using the default value in this case is reasonable, as the goal of the chapter is not to maximize the performance, but rather to introduce a hyper-heuristic framework.

Stopping criteria: 4 days or 2000 *generations (stop when either criterion is met)*

This stopping criteria is associated to the maximum runtime of a job in our cluster. We also use 2000 generations as a secondary stopping criterion to ensure that the hyper-heuristic is not trained for more generations in problems with a more computationally efficient objective function. As an example, $\mathfrak{P}_1 : \sum_{i=1}^{d} x_i^2$ evaluates faster than $\mathfrak{P}_{12} : -20 \exp\left(\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d} \cos(2\pi x_i)\right)$.

Testing parameters: Executions averaged: 10^4 :

The theoretical ideal parameter would be ∞, which is (by definition) the expected value. Many authors use as little as 20 samples when averaging the results, so if anything, 10^4 would be overkill and is definitely enough. As a reference, Hong et al. [Hong et al., 2018] use 50 samples in their analysis, which is likely enough as well. We used 10^4 to be absolutely sure that there are no problems with too few samples.

References

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004a.

Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms*. Number 305-306 in Grundlehren Der Mathematischen Wissenschaften. Springer-Verlag, Berlin ; New York, 2nd corr. print edition, 1996. ISBN 978-3-540-56850-6 978-0-387-56850-8 978-3-540-56852-0 978-0-387-56852-2.

Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1): 61–75, 1956. doi: 10.1287/opre.4.1.61.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

Ander Zarketa-Astigarraga, Alain Martin-Mayor, Aimar Maeso, Borja De Miguel, Manex Martinez-Agirre, and Markel Penalba. A computationally efficient ga-based optimization tool for the design of power take-off systems in realistic wave climates: The wells turbine case. *SSRN*, 2023. doi: 10.2139/ssrn.4379648. URL https://www.ssrn.com/abstract=4379648.

Pedro Larrañaga and Jose A Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2. Springer Science & Business Media, 2001a.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010. JMLR Workshop and Conference Proceedings.

L.A. Wolsey. *Integer Programming*. Wiley, 2020. ISBN 9781119606529. URL https://books.google.es/books?id=knH8DwAAQBAJ.

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004b. ISBN 0-521-83378-7 978-0-521-83378-3.

David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Tsp cuts which do not conform to the template paradigm. *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, pages 261–303, 2001.

David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1(1): 215–310, 1997.

Seymour E Goodman and Stephen T Hedetniemi. *Introduction to the Design and Analysis of Algorithms*. McGraw-Hill, Inc., 1977.

Verena Heidrich-Meisner and Christian Igel. Evolution strategies for direct policy search. In *International Conference on Parallel Problem Solving from Nature*, pages 428–437. Springer, 2008.

Faustino J Gomez, Risto Miikkulainen, et al. Solving non-Markovian control tasks with neuroevolution. In *IJCAI*, volume 99, pages 1356–1361. Citeseer, 1999.

Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. 2020. doi: 10.48550/ARXIV.2005.14165.

Matthew Hutson. Artificial intelligence faces reproducibility crisis. *Science (New York, N.Y.)*, 359(6377):725–726, 2018. doi: 10.1126/science.359.6377.725.

Julián Domínguez and Enrique Alba. A Methodology for Comparing the Execution Time of Metaheuristics Running on Different Hardware. In *Evolutionary Computation in Combinatorial Optimization*, volume 7245, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-29123-4 978-3-642-29124-1. doi: 10.1007/978-3-642-29124-1_1.

R. P. Weicker. Dhrystone benchmark: Rationale for version 2 and measurement rules. *ACM SIGPLAN Notices*, 23(8):49–62, August 1988. ISSN 0362-1340, 1558-1160. doi: 10.1145/47907.47911.

Justin Matejka and George Fitzmaurice. Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1290–1294, 2017.

F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1): 17–21, 1973. doi: 10.1080/00031305.1973.10478966.

James P. Quirk and Rubin Saposnik. Admissibility and measurable utility functions. *The Review of Economic Studies*, 29(2):140–146, February 1962. ISSN 0034-6527. doi: 10.2307/2295819.

Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.

Borja Calvo, Ofer M. Shir, Josu Ceberio, Carola Doerr, Hao Wang, Thomas Bäck, and Jose A. Lozano. Bayesian performance analysis for black-box optimization benchmarking. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '19*, pages 1789–1797, Prague, Czech Republic, 2019a. ACM Press. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3326888.

Andrew Gelman. *Bayesian Data Analysis*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, Boca Raton, third edition edition, 2014. ISBN 978-1-4398-4095-5.

José M Bernardo and Adrian FM Smith. *Bayesian Theory*, volume 405. John Wiley & Sons, 2009.

Alessio Benavoli, Giorgio Corani, Francesca Mangili, Marco Zaffalon, and Fabrizio Ruggeri. A Bayesian Wilcoxon signed-rank test based on the Dirichlet process. In *International Conference on Machine Learning*, pages 1026–1034. PMLR, 2014.

Tommaso Schiavinotto and Thomas Stützle. A review of metrics on permutations for search landscape analysis. *Computers & operations research*, 34(10):3143–3153, 2007. doi: 10.1016/j.cor.2005.11.022.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. A review of distances for the Mallows and Generalized Mallows estimation of distribution algorithms. *Computational Optimization and Applications*, 62 (2):545–564, November 2015a. ISSN 0926-6003, 1573-2894. doi: 10.1007/s10589-015-9740-x.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. The linear ordering problem revisited. *European Journal of Operational Research*, 241(3):686–696, March 2015b. ISSN 03772217. doi: 10.1016/j.ejor.2014.09.041.

R. Martí and G. Reinelt. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Applied Mathematical Sciences. Springer Berlin Heidelberg, 2011. ISBN 9783642167294. URL https://books.google.es/books?id=4Lk8pPYYI3cC.

Tjalling C. Koopmans and Martin Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25(1):53, January 1957. ISSN 00129682. doi: 10.2307/1907742.

Gabriela Ochoa and Katherine Malan. Recent advances in fitness landscape analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1077–1094, Prague Czech Republic, July 2019. ACM. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3323383.

Gabriela Ochoa, Sébastien Verel, Fabio Daolio, and Marco Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. In *Recent Advances in the Theory and Application of Fitness Landscapes*, pages 233–262. Springer, 2014.

John R Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

Ana Kostovska, Anja Jankovic, Diederick Vermetten, Jacob de Nobel, Hao Wang, Tome Eftimov, and Carola Doerr. Per-run algorithm selection with warm-starting using trajectory-based features. In Günter Rudolph, Anna V. Kononova, Hernán Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tušar, editors, *Parallel Problem Solving from Nature – PPSN XVII*, pages 46–60, Cham, 2022. Springer International Publishing. ISBN 978-3-031-14714-2.

Diederick Vermetten, Hao Wang, Kevin Sim, and Emma Hart. To switch or not to switch: Predicting the benefit of switching between algorithms based on trajectory features. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 335–350. Springer, 2023.

Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 829–836, Dublin Ireland, July 2011a. ACM. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001690.

Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Springer US, Boston, MA, 2003. ISBN 978-0-306-48056-0. doi: 10.1007/0-306-48056-5_16.

Libin Hong, John H. Drake, John R. Woodward, and Ender Özcan. A hyper-heuristic approach to automated generation of mutation operators for evolutionary programming. *Applied Soft Computing*, 62:162–175, January 2018. ISSN 15684946. doi: 10.1016/j.asoc.2017.10.002.

Léni K. Le Goff, Edgar Buchanan, Emma Hart, Agoston E. Eiben, Wei Li, Matteo De Carlo, Alan F. Winfield, Matthew F. Hale, Robert Woolley, Mike Angus, Jon Timmis, and Andy M. Tyrrell. Morpho-evolution with learning using a controller archive as an inheritance mechanism. *arXiv:2104.04269 [cs]*, September 2021.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248. PMLR, 2016.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, July 2018a.

Josh C. Bongard. Innocent Until Proven Guilty: Reducing Robot Shaping From Polynomial to Linear Time. *IEEE Transactions on Evolutionary Computation*, 15(4):571–585, August 2011. ISSN 1089-778X, 1941-0026. doi: 10.1109/TEVC.2010.2096540.

William D. Nordhaus. Two centuries of productivity growth in computing. *The Journal of Economic History*, 67(1):128–159, 2007. doi: 10.1017/S0022050707000058.

William Jay Conover. Practical nonparametric statistics. 1980.

P.W.D. Dougherty. Accneat, 2014.

GeeksforGeeks. 0-1 Knapsack Problem. https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/, 2022.

GeeksforGeeks. N Queen Problem. https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/, 2022.

Ronald L. Wasserstein and Nicole A. Lazar. The ASA Statement on p-Values: Context, Process, and Purpose. *The American Statistician*, 70(2):129–133, April 2016. ISSN 0003-1305, 1537-2731. doi: 10.1080/00031305.2016.1154108.

Una Benlic and Jin-Kao Hao. Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 42(1):584–595, January 2015a. ISSN 09574174. doi: 10.1016/j.eswa.2014.08.011.

Etor Arza, Aritz Pérez, Ekhiñe Irurozki, and Josu Ceberio. Kernels of Mallows Models under the Hamming Distance for solving the Quadratic Assignment Problem. *Swarm and Evolutionary Computation*, page 100740, July 2020. ISSN 22106502. doi: 10.1016/j.swevo.2020.100740.

Sangit Chatterjee and Aykut Firat. Generating Data with Identical Statistics but Dissimilar Graphics: A Follow up to the Anscombe Dataset. *The American Statistician*, 61(3):248–254, 2007. ISSN 00031305.

François Chollet et al. Keras. 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Angel Cruz-Roa, Hannah Gilmore, Ajay Basavanhally, Michael Feldman, Shridar Ganesan, Natalie N.C. Shih, John Tomaszewski, Fabio A. González, and Anant Madabhushi. Accurate and reproducible invasive breast cancer detection in whole-slide images: A Deep Learning approach for quantifying tumor extent. *Scientific Reports*, 7(1):46450, June 2017. ISSN 2045-2322. doi: 10.1038/srep46450.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000071.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013.

Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M. Paixão, Filipe Mutz, Lucas de Paula Veronese, Thiago Oliveira-Santos, and Alberto F. De Souza. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, March 2021. ISSN 09574174. doi: 10.1016/j.eswa.2020.113816.

Lei Zhang, Shuai Wang, and Bing Liu. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4), July 2018. ISSN 1942-4787, 1942-4795. doi: 10.1002/widm.1253.

Olivier Regnier-Coudert, John McCall, Mayowa Ayodele, and Steven Anderson. Truck and trailer scheduling in a real world, dynamic and heterogeneous context. *Transportation research part E: logistics and transportation review*, 93:389–408, 2016.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. ISSN 00034851.

Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80, December 1945. ISSN 00994987. doi: 10.2307/3001968.

Ander Carreño, Iñaki Inza, and Jose A. Lozano. Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework. *Artificial Intelligence Review*, 53(5):3575–3594, June 2020. ISSN 0269-2821, 1573-7462. doi: 10.1007/s10462-019-09771-y.

Peter J. Rousseeuw and Mia Hubert. Robust statistics for outlier detection. *WIREs Data Mining and Knowledge Discovery*, 1(1):73–79, January 2011. ISSN 1942-4787, 1942-4795. doi: 10.1002/widm.2.

Friedrich Schmid and Mark Trede. Testing for First-Order Stochastic Dominance: A New Distribution-Free Test. *The Statistician*, 45(3):371, 1996. ISSN 00390526. doi: 10.2307/2988473.

Christopher J. Bennet. Inference for Dominance Relations. *International Economic Review*, 54(4):1309–1328, 2013. doi: 10.1111/iere.12038.

R. C. H. Cheng and T. C. Iles. Confidence Bands for Cumulative Distribution Functions of Continuous Random Variables. *Technometrics*, 25(1):77–86, February 1983. ISSN 0040-1706. doi: 10.1080/00401706.1983.10487822.

G. P. Steck. Rectangle probabilities for uniform order statistics and the probability that the empirical distribution function lies between two distribution functions. *The Annals of Mathematical Statistics*, 42(1):1–11, 1971. ISSN 00034851.

Russel CH Cheng and TC Lies. One-sided confidence bands for cumulative distribution functions. *Technometrics*, 30(2):155–159, 1988.

Jiangyan Wang, Fuxia Cheng, and Lijian Yang. Smooth simultaneous confidence bands for cumulative distribution functions. *Journal of Nonparametric Statistics*, 25(2):395–407, June 2013. ISSN 1048-5252, 1029-0311. doi: 10.1080/10485252.2012.759219.

Julian J. Faraway and Myoungshic Jhun. Bootstrap choice of bandwidth for density estimation. *Journal of the American Statistical Association*, 85(412): 1119–1122, 1990. doi: 10.1080/01621459.1990.10474983.

P. J. Bickel and A. M. Krieger. Confidence bands for a distribution function using the bootstrap. *Journal of the American Statistical Association*, 84(405):95–100, 1989. doi: 10.1080/01621459.1989.10478742.

Peter Hall and Joel Horowitz. A simple bootstrap method for constructing nonparametric confidence bands for functions. *The Annals of Statistics*, 41(4), August 2013. ISSN 0090-5364. doi: 10.1214/13-AOS1137.

Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall, New York, 1993. ISBN 978-0-412-04231-7.

Jelle J. Goeman and Aldo Solari. Multiple hypothesis testing in genomics. *Statistics in Medicine*, 33(11):1946–1978, May 2014. ISSN 02776715. doi: 10.1002/sim.6082.

P. Bauer. Multiple testing in clinical trials. *Statistics in Medicine*, 10(6):871–890, June 1991. ISSN 02776715, 10970258. doi: 10.1002/sim.4780100609.

Olivier Thas. *Comparing Distributions*. Springer, 2010.

scikit-learn developers. Density Estimation. 2021.

Jerry L Hintze and Ray D Nelson. Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.

Sander Greenland, Stephen J. Senn, Kenneth J. Rothman, John B. Carlin, Charles Poole, Steven N. Goodman, and Douglas G. Altman. Statistical tests, P values, confidence intervals, and power: A guide to misinterpretations. *European Journal of Epidemiology*, 31(4):337–350, April 2016. ISSN 0393-2990, 1573-7284. doi: 10.1007/s10654-016-0149-3.

Daniel J. Benjamin, James O. Berger, Magnus Johannesson, Brian A. Nosek, E.-J. Wagenmakers, Richard Berk, Kenneth A. Bollen, Björn Brembs, Lawrence Brown, Colin Camerer, David Cesarini, Christopher D. Chambers, Merlise Clyde, Thomas D. Cook, Paul De Boeck, Zoltan Dienes, Anna Dreber, Kenny

Easwaran, Charles Efferson, Ernst Fehr, Fiona Fidler, Andy P. Field, Malcolm Forster, Edward I. George, Richard Gonzalez, Steven Goodman, Edwin Green, Donald P. Green, Anthony G. Greenwald, Jarrod D. Hadfield, Larry V. Hedges, Leonhard Held, Teck Hua Ho, Herbert Hoijtink, Daniel J. Hruschka, Kosuke Imai, Guido Imbens, John P. A. Ioannidis, Minjeong Jeon, James Holland Jones, Michael Kirchler, David Laibson, John List, Roderick Little, Arthur Lupia, Edouard Machery, Scott E. Maxwell, Michael McCarthy, Don A. Moore, Stephen L. Morgan, Marcus Munafó, Shinichi Nakagawa, Brendan Nyhan, Timothy H. Parker, Luis Pericchi, Marco Perugini, Jeff Rouder, Judith Rousseau, Victoria Savalei, Felix D. Schönbrodt, Thomas Sellke, Betsy Sinclair, Dustin Tingley, Trisha Van Zandt, Simine Vazire, Duncan J. Watts, Christopher Winship, Robert L. Wolpert, Yu Xie, Cristobal Young, Jonathan Zinman, and Valen E. Johnson. Redefine statistical significance. *Nature Human Behaviour*, 2(1):6–10, January 2018. ISSN 2397-3374. doi: 10.1038/s41562-017-0189-z.

John PA Ioannidis. The proposal to lower P value thresholds to. 005. *Jama*, 319 (14):1429–1430, 2018.

Teresa Ledwina and Grzegorz Wyłupek. Nonparametric tests for stochastic ordering. *TEST*, 21(4):730–756, December 2012. ISSN 1133-0686, 1863-8260. doi: 10.1007/s11749-011-0278-7.

W. Baumgartner, P. WeiB, and H. Schindler. A Nonparametric Test for the General Two-Sample Problem. *Biometrics*, 54(3):1129, September 1998. ISSN 0006341X. doi: 10.2307/2533862.

Munmun Biswas and Anil K. Ghosh. A nonparametric two-sample test applicable to high dimensional data. *Journal of Multivariate Analysis*, 123:160–171, January 2014. ISSN 0047259X. doi: 10.1016/j.jmva.2013.09.004.

Borja Calvo and Guzmán Santafé Rodrigo. Scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal, Vol. 8/1, Aug. 2016*, 2016a.

M. B. Wilk and R. Gnanadesikan. Probability Plotting Methods for the Analysis of Data. *Biometrika*, 55(1):1, March 1968. ISSN 00063444. doi: 10.2307/2334448.

Valentino Santucci, Josu Ceberio, and Marco Baioletti. Gradient search in the space of permutations: An application for the linear ordering problem. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1704–1711, Cancún Mexico, July 2020. ACM. ISBN 978-1-4503-7127-8. doi: 10.1145/3377929.3398094.

Tommaso Schiavinotto and Thomas Stützle. The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, 2004. ISSN 1570-1166. doi: 10.1023/B:JMMA.0000049426.06305.d8.

Danica and power. What is the minimum number of data points required for kernel density estimation?, 2009.

Michael R Chernick. *Bootstrap Methods: A Guide for Practitioners and Researchers*, volume 619. John Wiley & Sons, 2011.

Peter Hall. *The Bootstrap and Edgeworth Expansion*. Springer Science & Business Media, 2013.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.

Tjalling Koopmans and Martin J. Beckmann. Assignment problems and the location of economic activities. Cowles Foundation Discussion Papers 4, Cowles Foundation for Research in Economics, Yale University, 1955. URL https://EconPapers.repec.org/RePEc:cwl:cwldpp:4.

Josu Ceberio. *Solving permutation problems with estimation of distribution algorithms and extensions thereof*. PhD thesis, Universidad del País Vasco-Euskal Herriko Unibertsitatea, 2014.

Peter Hahn and Jakob Krarup. A hospital facility layout problem finally solved. 12, 05 2000.

Cesar Scarpini Rabak and Jaime Simão Sichman. Using a-teams to optimize automatic insertion of electronic components. *Advanced Engineering Informatics*, 17(2):95–106, 2003.

Ali Haghani and Min-Ching Chen. Optimizing gate assignments at airport terminals. 32:437–454, 08 1998.

Hans D. Mittelmann and Domenico Salvagnin. On solving a hard quadratic 3-dimensional assignment problem. *Mathematical Programming Computation*, 7(2):219–234, June 2015. ISSN 1867-2949, 1867-2957. doi: 10.1007/s12532-015-0077-3. URL http://link.springer.com/10.1007/s12532-015-0077-3.

Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, July 1976. ISSN 0004-5411. doi: 10.1145/321958.321975. URL http://doi.acm.org/10.1145/321958.321975.

Matteo Fischetti, Michele Monaci, and Domenico Salvagnin. Three ideas for the quadratic assignment problem. *Operations research*, 60(4):954–964, 2012.

Nathan W Brixius and Kurt M Anstreicher. The Steinberg wiring problem. In *The Sharpest Cut: The Impact of Manfred Padberg and His Work*, pages 293–307. SIAM, 2004.

Kurt Anstreicher, Nathan Brixius, Jean-Pierre Goux, and Jeff Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588, 2002.

Zvi Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 15(3):320–330, 2003.

David M Tate and Alice E Smith. A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, 22(1):73–84, 1995.

Jadranka Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on computing*, 2(1):33–45, 1990.

Tabitha James, Cesar Rego, and Fred Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195(3):810 – 826, 2009a. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2007.06.061. URL http://www.sciencedirect.com/science/article/pii/S0377221707011058.

P.E. Mickey R. Wilhelm Ph.D. and P.E. Thomas L. Ward Ph.D. Solving quadratic assignment problems by 'simulated annealing'. *IIE Transactions*, 19(1):107–119, 1987. doi: 10.1080/07408178708975376.

Luca Maria Gambardella, É D Taillard, and Marco Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the operational research society*, 50 (2):167–176, 1999.

Peter Merz and Bernd Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE transactions on evolutionary computation*, 4(4):337–352, 2000.

Hongbo Liu, Ajith Abraham, and Jianying Zhang. A particle swarm approach to quadratic assignment problems. In *Soft computing in industrial applications*, pages 213–222. Springer, 2007.

Tabitha James, Cesar Rego, and Fred Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195(3):810–826, June 2009b. ISSN 03772217. doi: 10.1016/j.ejor.2007.06.061.

Una Benlic and Jin-Kao Hao. Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 42(1):584–595, 2015b.

Tansel Dokeroglu and Ahmet Cosar. A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 52:10–25, June 2016. ISSN 09521976. doi: 10.1016/j.engappai.2016.02.004.

Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17(4-5):443–455, 1991.

Una Benlic and Jin-Kao Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, January 2013. ISSN 00963003. doi: 10.1016/j.amc.2012.10.106.

Pedro Larrañaga and Jose A Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001b.

Qingfu Zhang, Jianyong Sun, Edward Tsang, and John Ford. Combination of guided local search and estimation of distribution algorithm for quadratic assignment problems. In *Proceedings of the Genetic and Evolutionary Computation Conference, Chicago, IL, USA*, pages 42–48, 2003.

T.G. Pradeepmon, Vinay V. Panicker, and R. Sridharan. Hybrid estimation of distribution algorithms for solving a keyboard layout problem. *Journal of Indus-*

trial and Production Engineering, 35(6):352–367, August 2018. ISSN 2168-1015, 2168-1023. doi: 10.1080/21681015.2018.1508080.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A Lozano. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1(1):103–117, 2012.

Peter AN Bosman and Dirk Thierens. Crossing the road to efficient ideas for permutation problems. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 219–226. Morgan Kaufmann Publishers Inc., 2001.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(2):193–202, 1975.

David R. Hunter. Mm algorithms for generalized bradley-terry models. *Ann. Statist.*, 32(1):384–406, 02 2004. doi: 10.1214/aos/1079120141. URL https://doi.org/10.1214/aos/1079120141.

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

M. A. Fligner and J. S. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 48(3):359–369, 1986a. ISSN 00359246. URL http://www.jstor.org/stable/2345433.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A. Lozano. A review of distances for the Mallows and generalized Mallows estimation of distribution algorithms. *Computational Optimization and Applications*, 62(2): 545–564, Nov 2015c. ISSN 1573-2894". doi: 10.1007/s10589-015-9740-x. URL https://doi.org/10.1007/s10589-015-9740-x.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. Kernels of Mallows Models for Solving Permutation-based Problems. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*, pages 505–512, Madrid, Spain, 2015d. ACM Press. ISBN 978-1-4503-3472-3. doi: 10.1145/2739480.2754741.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A Lozano. Extending distance-based ranking models in estimation of distribution algorithms. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2459–2466. IEEE, 2014a.

Murilo Zangari, Ademir Aparecido Constantino, and Josu Ceberio. A decomposition-based kernel of Mallows models algorithm for bi-and tri-objective permutation flowshop scheduling problem. *Applied Soft Computing*, 71:526–537, 2018.

Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and Jose A Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18 (2):286–300, 2014b.

Ekhine Irurozki, Borja Calvo, and Jose Lozano. PerMallows: An R package for Mallows and generalized Mallows models. *Journal of Statistical Software, Articles*, 71(12):1–30, 2016. ISSN 1548-7660. doi: 10.18637/jss.v071.i12. URL https://www.jstatsoft.org/v071/i12.

Ekhine Irurozki, Borja Calvo, and Jose A. Lozano. Mallows and Generalized Mallows model for matchings. *Bernoulli*, 25(2):1160–1188, 05 2019a. doi: 10.3150/17-BEJ1017. URL https://doi.org/10.3150/17-BEJ1017.

Guy Lebanon and Yi Mao. Non-parametric modeling of partially ranked data. *Journal of Machine Learning Research*, 9(Oct):2401–2429, 2008.

Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, January 2007. ISSN 03772217. doi: 10.1016/j.ejor.2005.09.032.

Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-370-0. URL http://dl.acm.org/citation.cfm?id=645514.657929.

N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. A000166. Subfactorial or rencontres numbers, or derangements: number of permutations of n elements with no fixed points.

Thomas Brendan Murphy and Donal Martin. Mixtures of distance-based models for ranking data. *Computational statistics & data analysis*, 41(3-4):645–655, 2003.

Pranjal Awasthi, Avrim Blum, Or Sheffet, and Aravindan Vijayaraghavan. Learning mixtures of ranking models. In *Advances in Neural Information Processing Systems*, pages 2609–2617, 2014.

Josu Ceberio, Roberto Santana, Alexander Mendiburu, and Jose A Lozano. Mixtures of generalized Mallows models for solving the quadratic assignment problem. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2050–2057. IEEE, 2015e.

Zvi Drezner, Peter M Hahn, and Éeric D Taillard. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations research*, 139(1):65–94, 2005.

Rainer E Burkard, Stefan E Karisch, and Franz Rendl. QAPLIB–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997.

Panos M Pardalos, Henry Wolkowicz, et al. *Quadratic Assignment and Related Problems: DIMACS Workshop, May 20-21, 1993*, volume 16. American Mathematical Soc., 1994.

Borja Calvo, Ofer M. Shir, Josu Ceberio, Carola Doerr, Hao Wang, Thomas Bäck, and Jose A. Lozano. Bayesian performance analysis for black-box optimization

benchmarking. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 1789–1797, New York, NY, USA, 2019b. ACM. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3326888. URL http://doi.acm.org/10.1145/3319619.3326888.

Borja Calvo and Guzmán Santafé Rodrigo. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal, Vol. 8/1, Aug. 2016*, 2016b.

Josu Ceberio, Alexander Mendiburu, and Jose A Lozano. Kernels of Mallows models for solving permutation-based problems. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 505–512. ACM, 2015f.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. The Plackett-Luce ranking model on permutation-based optimization problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 494–501, Cancun, Mexico, June 2013. IEEE. ISBN 978-1-4799-0454-9 978-1-4799-0453-2 978-1-4799-0451-8 978-1-4799-0452-5. doi: 10.1109/CEC.2013.6557609.

Pedro Larrañaga, Ramón Etxeberria, José A Lozano, and José M Peña. Optimization in continuous domains by learning and simulation of gaussian networks. 2000.

Jeremy S De Bonet, Charles Lee Isbell Jr, and Paul A Viola. Mimic: Finding optima by estimating probability densities. In *Advances in neural information processing systems*, pages 424–430, 1997.

Endika Bengoetxea, Pedro Larranaga, Isabelle Bloch, Aymeric Perchant, and Claudia Boeres. Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867–2880, 2002.

Shigeyoshi Tsutsui, Martin Pelikan, and David E Goldberg. Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms. *IlliGAL Report*, 2003022, 2003.

Shigeyoshi Tsutsui, Martin Pelikan, and David E Goldberg. Node histogram vs. edge histogram: a comparison of pmbgas in permutation domains. *MEDAL Report*, (2006009), 2006.

Mayowa Ayodele, John McCall, and Olivier Regnier-Coudert. RK-EDA: A novel random key based estimation of distribution algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 849–858. Springer, 2016. doi: 10/gf3t68.

Mayowa Ayodele, John McCall, Olivier Regnier-Coudert, and Liam Bowie. A random key based estimation of distribution algorithm for the permutation flowshop scheduling problem. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2364–2371. IEEE, 2017.

Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, December 2013. ISSN 0160-5682, 1476-9360. doi: 10.1057/jors.2013.71.

Robert E Keller and Riccardo Poli. Self-adaptive hyperheuristic and greedy search. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3801–3808. IEEE, 2008.

Jacomine Grobler, Andries P. Engelbrecht, Graham Kendall, and V. S. S. Yadavalli. Alternative hyper-heuristic strategies for multi-method global optimization. In *IEEE Congress on Evolutionary Computation*, pages 1–8, Barcelona, Spain, July 2010. IEEE. ISBN 978-1-4244-6909-3. doi: 10.1109/cec.2010.5585980.

David Meignan, Abderrafiaa Koukam, and Jean-Charles Créput. Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, December 2010. ISSN 1381-1231, 1572-9397. doi: 10.1007/s10732-009-9121-7.

Edmund K. Burke, Matthew R. Hyde, and Graham Kendall. Grammatical Evolution of Local Search Heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3):406–417, June 2012. ISSN 1089-778X, 1089-778X, 1941-0026. doi: 10.1109/tevc.2011.2160401.

Simon Martin, Djamila Ouelhadj, Patrick Beullens, Ender Ozcan, Angel A. Juan, and Edmund K. Burke. A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research*, 254(1):169–178, October 2016. ISSN 03772217. doi: 10.1016/j.ejor.2016.02.045.

Vladimir Stanovov, Shakhnaz Akhmedova, and Eugene Semenkin. Neuroevolution for Parameter Adaptation in Differential Evolution. *Algorithms*, 15(4):122, April 2022. ISSN 1999-4893. doi: 10.3390/a15040122.

Nasser R. Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Grammatical Evolution Hyper-Heuristic for Combinatorial Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 17(6):840–861, December 2013. ISSN 1089-778X, 1089-778X, 1941-0026. doi: 10.1109/tevc.2013.2281527.

Nasser R. Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325, 2015. doi: 10.1109/TEVC.2014.2319051.

Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A. Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J. Parkes, Sanja Petrovic, and Edmund K. Burke. HyFlex: A benchmark framework for cross-domain heuristic search. In Jin-Kao Hao and Martin Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 136–147, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29124-1.

Jorge M. Cruz-Duarte, Ivan Amaya, Jose Carlos Ortiz-Bayliss, Santiago Enrique Conant-Pablos, and Hugo Terashima-Marin. A Primary Study on Hyper-Heuristics to Customise Metaheuristics for Continuous optimisation. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE. ISBN 978-1-72816-929-3. doi: 10.1109/CEC48606.2020.9185591.

Fabio Caraffini, Ferrante Neri, and Michael Epitropakis. HyperSPAM: A study on hyper-heuristic coordination strategies in the continuous domain. *Information Sciences*, 477:186–202, 2019.

Nelishia Pillay and Rong Qu. *Hyper-Heuristics: Theory and Applications*. Natural Computing Series. Springer International Publishing, Cham, 2018. ISBN 978-3-319-96513-0 978-3-319-96514-7. doi: 10.1007/978-3-319-96514-7.

Anja Jankovic and Carola Doerr. Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pages 841–849, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-7128-5. doi: 10.1145/3377930.3390183.

Gresa Shala, André Biedenkapp, Noor Awad, Steven Adriaensen, Marius Lindauer, and Frank Hutter. Learning Step-Size Adaptation in CMA-ES. In Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature – PPSN XVI*, volume 12269, pages 691–706. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58111-4 978-3-030-58112-1. doi: 10.1007/978-3-030-58112-1_48.

Riccardo Poli, William B Langdon, and Owen Holland. Extending particle swarm optimisation via genetic programming. In *European Conference on Genetic Programming*, pages 291–300. Springer, 2005.

Erik Pitzer and Michael Affenzeller. A comprehensive survey on fitness landscape analysis. *Recent advances in intelligent engineering systems*, pages 161–191, 2012.

Kenneth O. Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, June 2002. ISSN 1063-6560, 1530-9304. doi: 10.1162/106365602320169811.

J Kennedy and R Eberhart. Particle swarm optimization (PSO). In *Proc. IEEE International Conference on Neural Networks, Perth, Australia*, pages 1942–1948, 1995. doi: 10.1109/ICNN.1995.488968.

AP Engelbrecht and CW Cleghorn. Recent advances in particle swarm optimization analysis and understanding. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, pages 747–774, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-7127-8. doi: 10.1145/3377929.3389850.

Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated Annealing*, pages 7–15. Springer Netherlands, Dordrecht, 1987. ISBN 978-90-481-8438-5 978-94-015-7744-1. doi: 10.1007/978-94-015-7744-1_2.

Persi Diaconis and Ronald L Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2): 262–268, 1977.

M. A. Fligner and J. S. Verducci. Distance Based Ranking Models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 48(3):359–369, July 1986b. ISSN 00359246. doi: 10.1111/j.2517-6161.1986.tb01420.x.

Ekhine Irurozki, Borja Calvo, and Jose A. Lozano. Mallows and generalized Mallows model for matchings. *Bernoulli*, 25(2):1160–1188, May 2019b. ISSN 1350-7265. doi: 10.3150/17-BEJ1017.

Wade D. Cook and Lawrence M. Seiford. On the Borda-Kendall Consensus Method for Priority Ranking Problems. *Management Science*, 28(6):621–637, June 1982. ISSN 0025-1909, 1526-5501. doi: 10.1287/mnsc.28.6.621.

Jean C de Borda. Memoire sur les Elections au Scrutin. *Histoire de l'Academie Royale des Sciences, Paris*, 1781.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers operations research*, 13(5):533–549, 1986. doi: 10.1016/0305-0548(86)90048-1.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997. doi: 10.1016/S0305-0548(97)00031-2.

Gorka Kobeaga, María Merino, and Jose A Lozano. An efficient evolutionary algorithm for the orienteering problem. *Computers & Operations Research*, 90: 42–59, 2018.

David E. Goldberg and Robert Lingle, Jr. Alleles, Loci and the Traveling Salesman Problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159. L. Erlbaum Associates Inc., 1985. ISBN 0-8058-0426-9.

Jatinder N.D. Gupta and Edward F. Stafford. Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699–711, March 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.02.001.

Yibing Li, Zhiyu Tao, Lei Wang, Baigang Du, Jun Guo, and Shibao Pang. Digital twin-based job shop anomaly detection and dynamic scheduling. *Robotics and Computer-Integrated Manufacturing*, 79:102443, February 2023. ISSN 07365845. doi: 10.1016/j.rcim.2022.102443.

Anja Janković and Carola Doerr. Adaptive landscape analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 2032–2035, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3326905.

Johann Dreo, Carola Doerr, and Yann Semet. Coupling the design of benchmark with algorithm in landscape-aware solver design. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 1419–1420, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3326821.

David Wölfle, Arun Vishwanath, and Hartmut Schmeck. A guide for the design of benchmark environments for building energy optimization. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, BuildSys '20, pages 220–229, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-8061-4. doi: 10.1145/3408308.3427614.

Takehisa Kohira, Hiromasa Kemmotsu, Oyama Akira, and Tomoaki Tatsukawa. Proposal of benchmark problem based on real-world car structure design optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 183–184, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 978-1-4503-5764-7. doi: 10.1145/3205651.3205702.

Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 829–836, New York, NY, USA, 2011b. Association for Computing Machinery. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001690.

Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *Acm Computing Surveys*, 41(1), January 2009. ISSN 0360-0300. doi: 10.1145/1456650.1456656.

Mario A. Munoz, Michael Kirley, and Saman K. Halgamuge. Exploratory Landscape Analysis of Continuous Space Optimization Problems Using Information Content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87, February 2015. ISSN 1089-778X, 1089-778X, 1941-0026. doi: 10.1109/TEVC.2014.2302006.

Prerna Singh. Dimensionality reduction approaches. https://towardsdatascience.com/dimensionality-reduction-approaches-8547c4c44334, 2020-08-18, 2020.

C Radhakrishna Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved November 8, 2021, from http://www.sfu.ca/~ssurjano.

Jani Rönkkönen, Xiaodong Li, Ville Kyrki, and Jouni Lampinen. A Generator for Multimodal Test Functions with Multiple Global Optima. In Xiaodong Li, Michael Kirley, Mengjie Zhang, David Green, Vic Ciesielski, Hussein Abbass,

Zbigniew Michalewicz, Tim Hendtlass, Kalyanmoy Deb, Kay Chen Tan, Jürgen Branke, and Yuhui Shi, editors, *Simulated Evolution and Learning*, volume 5361, pages 239–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-89693-7 978-3-540-89694-4. doi: 10.1007/978-3-540-89694-4_25.

Stefan Janson and Martin Middendorf. On Trajectories of Particles in PSO. In *2007 IEEE Swarm Intelligence Symposium*, pages 150–155, Honolulu, HI, USA, April 2007. IEEE. ISBN 978-1-4244-0708-8. doi: 10.1109/SIS.2007.368039.

Christopher K. Monson and Kevin D. Seppi. Exposing origin-seeking bias in PSO. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 241–248, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1-59593-010-8. doi: 10.1145/1068009.1068045.

Anne Elorza, Leticia Hernando, and Jose A Lozano. Taxonomization of combinatorial optimization problems in fourier space. *arXiv preprint arXiv:1905.10852*, 2019.

Thomas Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, November 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.01.066.

Jadranka Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on computing*, 2(1):33–45, 1990.

Thibaud De Souza. The Blind Game Designer - Darwinism in a fast pace, casual action game, 2014.

Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Evolving content in the Galactic Arms Race video game. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 241–248, 2009. doi: 10.1109/CIG.2009.5286468.

F. Hoffmann. Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE*, 89(9):1318–1333, Sept./2001. ISSN 00189219. doi: 10.1109/5.949487.

P.J Fleming and R.C Purshouse. Evolutionary algorithms in control systems engineering: A survey. *Control Engineering Practice*, 10(11):1223–1241, November 2002. ISSN 09670661. doi: 10.1016/S0967-0661(02)00081-3.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

Ken A. Dill, S. Banu Ozkan, M. Scott Shell, and Thomas R. Weikl. The Protein Folding Problem. *Annual Review of Biophysics*, 37(1):289–316, June 2008. ISSN 1936-122X, 1936-1238. doi: 10.1146/annurev.biophys.37.092707.153558.

Emma Hart and Léni K Le Goff. Artificial evolution of robot bodies and control: On the interaction between evolution, learning and culture. *Philosophical Transactions of the Royal Society B*, 377(1843):20210117, 2022.

Sheng-Hao Wu, Zhi-Hui Zhan, and Jun Zhang. SAFE: Scale-Adaptive Fitness Evaluation Method for Expensive Optimization Problems. *IEEE Transactions*

on Evolutionary Computation, 25(3):478–491, June 2021. ISSN 1089-778X, 1089-778X, 1941-0026. doi: 10.1109/TEVC.2021.3051608.

Gorka Kobeaga, María Merino, and Jose A Lozano. On solving cycle problems with Branch-and-Cut: Extending shrinking and exact subcycle elimination separation algorithms. *Annals of Operations Research*, 305(1):107–136, 2021.

Marcelo de Souza, Marcus Ritt, and Manuel López-Ibáñez. Capping methods for the automatic configuration of optimization algorithms. *Computers & Operations Research*, 139:105615, March 2022. ISSN 03050548. doi: 10.1016/j.cor.2021.105615.

John T. Hwang and Joaquim R.R.A. Martins. A fast-prediction surrogate model for large datasets. *Aerospace Science and Technology*, 75:74–87, April 2018. ISSN 12709638. doi: 10.1016/j.ast.2017.12.030.

Sascha Ranftl and Wolfgang von der Linden. Bayesian Surrogate Analysis and Uncertainty Propagation. In *The 40th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, page 6. MDPI, November 2021. doi: 10.3390/psf2021003006.

Reza Alizadeh, Janet K. Allen, and Farrokh Mistree. Managing computational complexity using surrogate models: A critical review. *Research in Engineering Design*, 31(3):275–298, July 2020. ISSN 0934-9839, 1435-6066. doi: 10.1007/s00163-020-00336-7.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. ISSN 22147160. doi: 10.1016/j.orp.2016.09.002.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pages 1238–1246. PMLR, 2013.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, September 1983. ISSN 0018-9472, 2168-2909. doi: 10.1109/TSMC.1983.6313077.

Rrichard S. Sutton. *Temporal Aspects of Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, 1984.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th international conference on machine learning*, volume 80 of *Proceedings of machine learning research*, page 1437–1446. PMLR, Jul 2018b. URL https://proceedings.mlr.press/v80/falkner18a.html. Citation Key: pmlr-v80-falkner18a.

Josh Bongard. The utility of evolving simulated robot morphology increases with task complexity for object manipulation. *Artificial life*, 16(3):201–223, 2010.

Christian Igel, Thorsten Suttorp, and Nikolaus Hansen. A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation - GECCO '06*, page 453, Seattle, Washington, USA, 2006. ACM Press. ISBN 978-1-59593-186-3. doi: 10.1145/1143997.1144082.

Léni K. Le Goff, Edgar Buchanan, Emma Hart, Agoston E. Eiben, Wei Li, Matteo de Carlo, Matthew F. Hale, Mike Angus, Robert Woolley, Jon Timmis, Alan Winfield, and Andrew M. Tyrrell. Sample and time efficient policy learning with CMA-ES and Bayesian Optimisation. In *The 2020 Conference on Artificial Life*, pages 432–440, Online, 2020. MIT Press. doi: 10.1162/isal_a_00299.

Frank Veenstra and Kyrre Glette. How different encodings affect performance and diversification when evolving the morphology and control of 2D virtual creatures. In *ALIFE: Proceedings of the Artificial Life Conference*, pages 592–601. MIT Press, 2020.

Johann Dreo and Manuel López-Ibáñez. Extensible logging and empirical attainment function for IOHexperimenter. *arXiv preprint arXiv:2109.13773*, 2021.

Etor Arza, Josu Ceberio, Ekhiñe Irurozki, and Aritz Pérez. Comparing Two Samples Through Stochastic Dominance: A Graphical Approach. *Journal of Computational and Graphical Statistics*, pages 1–38, June 2022. ISSN 1061-8600. doi: 10.1080/10618600.2022.2084405.

Jussi Korpela, Kai Puolamäki, and Aristides Gionis. Confidence bands for time series data. *Data Mining and Knowledge Discovery*, 28(5):1530–1553, Sep 2014. ISSN 1573-756X. doi: 10.1007/s10618-014-0371-0.

Wolfgang Härdle, Marlene Müller, Stefan Sperlich, and Axel Werwatz. *Nonparametric and Semiparametric Models*, volume 1 of *Springer Series in Statistics*. Springer, 2004. ISBN 978-3-642-17146-8.

Vivek Verma. Applying Neural Networks and Neuroevolution of Augmenting Topologies to play Super Mario Bros, 2020.

Agoston E. Eiben, Emma Hart, Jon Timmis, Andy M. Tyrrell, and Alan F. Winfield. Towards Autonomous Robot Evolution. In Ana Cavalcanti, Brijesh Dongol, Rob Hierons, Jon Timmis, and Jim Woodcock, editors, *Software Engineering for Robotics*, pages 29–51. Springer International Publishing, Cham, 2021. ISBN 978-3-030-66493-0 978-3-030-66494-7. doi: 10.1007/978-3-030-66494-7_2.

Joel Lehman and Kenneth O. Stanley. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, 19(2):189–223, June 2011. ISSN 1063-6560, 1530-9304. doi: 10.1162/evco_a_00025.

Guodong Chen, Yong Li, Kai Zhang, Xiaoming Xue, Jian Wang, Qin Luo, Chuanjin Yao, and Jun Yao. Efficient hierarchical surrogate-assisted differential evolution for high-dimensional expensive optimization. *Information Sciences*, 542: 228–246, 2021.

P. C. Álvarez-Esteban, E. del Barrio, J. A. Cuesta-Albertos, and C. Matrán. A contamination model for the stochastic order. *TEST*, 25(4):751–774, December 2016. ISSN 1863-8260. doi: 10.1007/s11749-016-0494-2.

Milan Jelisavcic, Rafael Kiesel, Kyrre Glette, Evert Haasdijk, and A. E. Eiben. *Analysis of Lamarckian Evolution in Morphologically Evolving Robots*, volume ECAL 2017, the Fourteenth European Conference on Artificial Life of *ALIFE 2022: The 2022 Conference on Artificial Life*. September 2017. doi: 10.1162/isal_a_038.

Thomas Liao, Grant Wang, Brian Yang, Rene Lee, Kristofer Pister, Sergey Levine, and Roberto Calandra. Data-efficient Learning of Morphology and Controller for a Microrobot. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 15–22, 1994.

Hod Lipson, Vytas Sunspiral, Josh Bongard, and Nicholas Cheney. *On the Difficulty of Co-Optimizing Morphology and Control in Evolved Virtual Creatures*, volume ALIFE 2016, the Fifteenth International Conference on the Synthesis and Simulation of Living Systems of *ALIFE 2022: The 2022 Conference on Artificial Life*. July 2016. doi: 10.1162/978-0-262-33936-0-ch042.

Kelly H Zou, Kemal Tuncali, and Stuart G Silverman. Correlation and simple linear regression. *Radiology*, 227(3):617–628, 2003.

Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. The linear ordering problem revisited. *European Journal of Operational Research*, 241(3):686–696, March 2015g. ISSN 03772217. doi: 10.1016/j.ejor.2014.09.041.

William H Beyer. *Standard Probability and Statistics: Tables and Formulae*. CRC Press, 1991.

F. Liese and I. Vajda. On Divergences and Informations in Statistics and Information Theory. *IEEE Transactions on Information Theory*, 52(10):4394–4412, October 2006. ISSN 0018-9448. doi: 10.1109/TIT.2006.881731.

Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

Yury Polyanskiy and Yihong Wu. Lecture notes on information theory. *MIT (6.441), UIUC (ECE 563), Yale (STAT 664)*, 2012.

S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951. ISSN 0003-4851. doi: 10.1214/aoms/1177729694.

Alecos Papadopoulos. Interpretation of the Kullback-Leibler divergence, 2017.

Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer Series in Statistics. Springer, New York ; London, 2009. ISBN 978-0-387-79051-0 978-0-387-79052-7.

Vladimir Naumovich Vapnik. *Statistical Learning Theory*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. Wiley, New York, 1998. ISBN 978-0-471-03003-4.

Luc Devroye, Abbas Mehrabian, and Tommy Reddad. The total variation distance between high-dimensional Gaussians. *arXiv:1810.08693 [math, stat]*, May 2020.

an Xi. Differences between Bhattacharyya distance and KL divergence. 2017.

T. Kailath. The Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Transactions on Communications*, 15(1):52–60, February 1967. ISSN 0096-2244. doi: 10.1109/TCOM.1967.1089532.

Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.

Dominic Schuhmacher. Compute The Wasserstein Distance Between Two Univariate Samples, 2021.

Victor M. Panaretos and Yoav Zemel. Statistical aspects of wasserstein distances. *Annual Review of Statistics and Its Application*, 6(1):405–431, 2019. doi: 10.1146/annurev-statistics-030718-104938.

J David Schaffer, Darrell Whitley, and Larry J Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE, 1992.

K. O. Stanley and R. Miikkulainen. Competitive Coevolution through Evolutionary Complexification. *Journal of Artificial Intelligence Research*, 21:63–100, February 2004. ISSN 1076-9757. doi: 10.1613/jair.1338.

Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.

Paolo Pagliuca, Nicola Milano, and Stefano Nolfi. Maximizing adaptive power in neuroevolution. In *PloS One*, 2018.

Hirotaka Moriguchi and Shinichi Honiden. CMA-TWEANN: Efficient optimization of neural networks via self-adaptation and seamless augmentation. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference - GECCO '12*, page 903, Philadelphia, Pennsylvania, USA, 2012. ACM Press. ISBN 978-1-4503-1177-9. doi: 10.1145/2330163.2330288.

Yanan Sun, Gary G. Yen, and Zhang Yi. Evolving Unsupervised Deep Neural Networks for Learning Meaningful Representations. *IEEE Transactions on Evolutionary Computation*, 23(1):89–103, February 2019. ISSN 1089-778X, 1089-778X, 1941-0026. doi: 10.1109/TEVC.2018.2808689.

Francisco Erivaldo Fernandes Junior and Gary G. Yen. Particle swarm optimization of deep neural networks architectures for Image classification. *Swarm*