ÉCOLE
POLYTECHNIQUE
DE BRUXELLES
UNIVERSITÉ LIBRE DE BRUXELLES

Empirical Analysis of Stochastic Local Search Behaviour: Connecting Structure, Components and Landscape

Thesis presented by Alberto FRANZIN
in fulfilment of the requirements of the PhD Degree in Engineering
("Docteur en Sciences de l'Ingénieur")
Année académique 2020-2021

Supervisor: Professor Thomas STÜTZLE
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Abstract

Stochastic Local Search algorithms (SLS) are a class of methods used to tackle hard combinatorial optimization problems. Despite not providing, in most cases, any guarantee on the quality of the final solution, they are often able to produce high quality solutions in a relatively short time. They are therefore routinely used in countless real world applications, and their wide applicability generates in turn a lot of research interest, both from a theoretical perspective and with applications on countless problems. There are several research lines that not only propose SLS algorithms to solve problems, but that also try to understand how SLS methods work, and to explain the results obtained by an algorithm. All these works are however focused on specific problems and instances, or address one particular point of view, and it is therefore difficult to connect the information provided. Hence, despite the amount of research, how exactly SLS algorithms work is still an open research question.

In this thesis I take an experimental approach to study how one of the most popular SLS algorithms, simulated annealing (SA), works. I note that many instantiations of SA have been proposed to solve different problems, implementing new ideas over the original formulation, and that other algorithms appearing in the SLS literature with a different name can be represented in the simulated annealing structure. Thus, I collect these ideas into an algorithmic framework, classifying them by the specific role they have in the algorithm. We use automatic algorithm configuration tools to automatically improve the behaviour of existing SA algorithms for a set of problems, and to automatically generate new SA algorithms able to outperform the existing ones. By analyzing the composition of the resulting algorithms, I observe the different characteristics that high quality SA algorithms need to exhibit to successfully tackle different problems.

I then investigate the relationship between the algorithm structure and a scenario, to understand how the characteristics of problems and instances impact the performance of SA algorithms. I consider two SA variants and evaluate them on a variety of scenarios. I then measure the conditions en-

countered by the algorithms, and relate them to the results obtained by the algorithms. I observe that a fixed-temperature variant of simulated annealing works well when the structure of the neighbourhoods is similar in different areas of the search space. The traditional cooling simulated annealing is instead a better option when the conditions to escape path towards locally optimal solutions change for different regions of the landscape.

Finally, I propose a causal framework that models the interplay between the elements involved in the solution of an optimization problem. I show how under this framework it is possible to represent several research directions, disambiguating the questions addressed by various works. I also show how the clarification of the relationships between problems, instances, algorithms and results is useful to approach open problems. We provide a proof of concept of a transfer learning approach for algorithm configurations that works across different problems.

Acknowledgments

The research presented in this thesis has been funded by the COMEX project (P7/36) within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office, and the Innoviris project 2018-SHAPE-25a "PHILEAS: smart monitoring par détection de comportements anormaux appliquée aux objets connectés".

First of all I thank my supervisor Thomas Stützle, for giving me the opportunity to join IRIDIA, guiding me in my research and showing me how to do solid research. I am also indebted to Francesco Sambo, who first suggested me to apply and is therefore the main responsible for me taking this road. I also thank Hugues Bersini for involving me in the Philéas project and thus giving me the chance of continuing working in IRIDIA.

If I made it this far it is also thank to my friends and colleagues in IRIDIA, who made, and continue to make, the lab a great place for both work and fun. I have to mention in particular Federico Pagnozzi, upon whose research my work is based. I also thank all the other people with whom I have worked, in particular my other coauthors: Leslie Pérez Cáceres, Raphaël Gyory and Moiés Silva at IRIDIA; Barbara di Camillo in Padova; Jean-Charles Nadé, Guillaume Aubert and Georges Klenkle at Degetel Belgium; and Gonzalo Calderon in Madrid.

I thank my family for their support all this time. Finally, in all these year I met many people, in and out schools and universities, whose contribution entered, in one way or another, in this thesis. They are too many to list, but from all of them I tried to learn new things.

Contents

List of Figures

List of Tables

Introduction 1

1.1 Context and motivation

Many real-world problems require to minimize or maximize a certain objective. For example, taking the shortest road to reach the desired destination, completing a task as early as possible, packing as many objects as possible in a certain space, minimizing the estimated error that a mathematical model will have when deployed to unseen data, designing the wings of an airplane, and many more. Sometimes the problem might require to optimize several, possibly conflicting goals. When we choose a restaurant for dinner, we consider both the quality of the food and the price, which are expected to be negatively correlated; that is, we have two or possibly more objectives to optimize. In this case we talk about multi-objective or multi-criteria optimization. When there is uncertainty in either the entities involved or the measurement of the outcome of the decision, we have stochastic optimization. For example, if two different roads between two locations have the same length, we normally prefer the one where we expect to find less congestion. Finally, a problem can change over time, like a navigation system that needs to recompute the best road to suggest to its user as the drive goes; this is an example of time-varying or dynamic optimization. Optimization problems are often complicated by the presence of constraints that we need to satisfy, such as a budget that cannot be exceeded, or a mimimum amount of certain objects to take. Constraints can take the softer form of preferences, that we would like to satisfy, but that we know they might not be possible to accommodate.

Several fields study how to solve optimization problems in theory and practice, often with important application in real life. In business, economics and management, optimization is used to guide the decision making process in companies [1, 2]. In computational biology, several tasks like aligning genomes or assembling DNA fragments can be framed as optimization problems [3]. Many machine learning problems are effectively optimization problems, often requiring to minimize an error or loss function or maximizing a certain score [4]. Researchers and practitioners use optimization to plan the allocation of resources to prepare for emergencies and to react in the most efficient way possible [5, 6, 7]. Even seemingly unrelated fields such as architecture face issues that can be defined as optimization problems. For the new Hamburg concert hall, the Elbphilharmonie, acoustic designers used an optimization algorithm to shape and place gypsum-fiber acoustic panels in order to enhance the listening experience of the audience.[1] The shape of the award-winning YEZO retreat in Hokkaido, Japan was designed by algorithms with the goal of minimizing both the production costs and the ecological impact of the building.[2]

Real world problems often include domain-specific details that might hide the real optimization proces. Cutting a wooden plank and allocating frequencies to several broadcasters are very different problems on the surface, but both are examples of a particular problem called *two dimensional bin packing* [8, 9]. To derive specific solutions we need of course to consider the characteristic of the specific problems, but the optimization part in itself can often be abstracted and treated independently, at least to a certain extent.

Unfortunately, many optimization problems with great practical relevance are *difficult* to solve, that is, there is no known algorithm that computes a provably optimal solution in time polynomial in the size of the input instance. In the general case it takes a time exponential in the size of the input. This makes the exact solution of mid-to-large instances impractical for most problems. Therefore often, and especially in practical applications, one has to settle for a solution that is *good enough* for the desired goal, renouncing to theoretical guarantees about the solution quality in exchange for a hopefully high quality solution computed in a relatively short time.

Many of such *heuristic* algorithms to compute suboptimal solutions belong to a family of algorithms called *stochastic local search* algorithms and the particular class of *metaheuristic* algorithms [10, 11]. These methods can be seen as problem-independent templates that provide general guidelines to design an effective heuristic algorithm for a certain problem [12].

[1] https://www.wired.com/2017/01/happens-algorithms-design-concert-hall-stunning-elbphilharmonie/

[2] https://www.yankodesign.com/2021/01/04/algorithms-helped-design-the-shape-of-this-japanese-holiday-retreat/

In principle, different instances generate a different set of solutions and therefore different algorithms are needed. In particular, a good SLS algorithm needs to implement a mechanism to move between solutions that is successful in both moving away rapidly from poor quality solution and thoroughly covering regions of good solutions. These are contrasting objectives that require a delicate balancing, known in the optimization literature as the *diversification/intensification* or, equivalently, *exploration/exploitation tradeoff*. For single-solution SLS algorithms, the ones to which this thesis is devoted, this tradeoff is achieved in two, non mutually exclusive, main ways: accepting worsening moves, and potentially excluding improving moves by considering only subsets of candidate solutions. The idea is to avoid the most immediate improvement to bet on a greater reward later on.

Over the years dozens of different such methods have been proposed and applied to thousands of problems, thus building a huge body of literature. There are, however, a few main families into which these methods can be usually be classified, depending on which mechanism they employ to balance diversification and intensification. Very often these method introduce a minor modification to the generic template, to better cope with the specific scenario considered. The literature on SLS algorithms is therefore extremely vast, and an interested practicioner is often left with the doubt of what algorithm to use for his or her specific case, or, after having made a choice and obtained some results, whether a different algorithm would have been better. To navigate the body of literature the practitioner has virtually no viable alternative than to follow the most common choice for related cases, trying a few alternatives and injecting his or her own biases and preferences in the process.

The problem is made worse by the current standard publication practices in the field. In order to publish a new algorithm or new algorithmic results, authors have to demonstrate the superiority of their method with respect to existing alternatives. This practice creates a pressure on authors to "raise the bar" in terms of performance, but at the same time no corresponding incentives are placed on properly explaining the results. It is therefore tempting to look for the case, the scenario or the settings where the new method shines in comparison to the alternatives, or the alternatives that make the new method shine. In some extreme cases, this opened the door to malpractices and documented frauds. But even when the authors are in good faith and the results are reliable, it becomes difficult to evaluate the extent to which they can be consider valid when a different scenario is to be considered. The outcome of this development process is, with any probability, a suboptimal algorithm.

The most efficient way of making use of this knowledge corpus is then to collect all these algorithms into flexible *algorithmic frameworks*, identifing

and classifying their particular contributions, and to make use of automatic configuration and design tools to obtain an algorithm tailored for a specific application scenario. These tools make it possible to build an algorithm that is tailored for a specific problem and set of instances, by performing extensive experiments without human intervention, exploring a broader set of alternatives than what a human operator could do, removing not only the tedium but also the user biases from the process. The resulting algorithm is therefore expected to be of high quality, and this new way of developing optimization algorithms has consistently obtained state-of-the-art results where applied. This approach is known as the automated algorithm design paradigm.

On the other hand, despite the widespread adoption and success of SLS algorithms, a proper understanding of how they actually work is still an open fundamental research question [13]. A precise mathematical modeling of the algorithm behaviour requires simplifications and assumptions that are not necessarily met in practice. The complex analyses required are also not easily transferrable onto the development of an algorithm. Experimental analyses are the only way of analyzing complex real-world scenarios, but the insights they can provide are limited by the boundaries and limitations of the experimental observations. For example, from a comparison between two algorithms for a specific problem we cannot infer any additional insight about the performance of the algorithms on a different scenario, or about the possible existence of a third, better algorithm for that problem. Moreover, often the specific research questions discussed in different papers that deal with the same algorithms are not easily related to each other. For example, the simple definition of superiority of an algorithm over a competing alternative is ambiguous and requires to be framed in a precise context.

It is therefore not easy to relate the many insights about SLS performance discussed in the literature and also for this reason the field of understanding how optimization algorithms work has not been able to keep the pace of advancement of its applicative counterpart. The lack of a unified, structured understanding of SLS algorithms is the ultimate reason behind the statement of some authors who claimed that the field of metaheuristics has not yet reached its maturity, despite decades of existence and several thousands of published works [14]. The focus on competition at the detriment of understanding and empirical rigor has been observed and decried in another fast-advancing field, machine learning, which shares with the field of metaheuristics a widespread drive towards better results above anything else. Some authors even argued that this sort of practice is actually slowing down the progress in the field [15]. While the advancement in terms of better methods and better results is of course a desired outcome, this should not happen at the detriment of our understanding and, ultimately, control.

1.2 Contributions

My research focused on using automatic methods to generate efficient algorithms to study algorithmic behaviour. The goal of an optimization algorithm is to obtain the best possible results on a certain scenario, so its goodness is defined solely on the quality of the results it obtains. The idea at the heart of this thesis is therefore that only by taking an algorithm to its maximum potential we can really understand how and why it works, and why it does not. There are in fact many reasons why an algorithm can obtain bad results, but only by identifying what makes an algorithm good we can gain knowledge that is useful in developing new, more powerful algorithms.

I have studied the behaviour of one of the oldest and most popular SLS algorithms, simulated annealing (SA), on a set of unconstrained combinatorial optimization problems. SA is a metaheuristic based on the probabilistic acceptance of worsening moves, and under its most general definition we can also include several other SLS algorithms as SA variants. SA makes use of a parameter called *temperature* to control the exploration/exploitation tradeoff and alter its behaviour during the search. Starting from an extensive literature review, I have identified the exact modifications introduced over the original SA template in the literature, classified them according to their function, and collected them into an algorithmic framework. By pairing this framework with automatic configuration tools, I have been able to reproduce and improve existing SA algorithms from the literature and to automatically obtain new, high-performing SA algorithms. By performing extensive systematic experiments, I have observed how different SA implementations perform on the various scenarios and what a good SA for a certain scenario looks like. I have then identified the most important components of a SA algorithm, that is, what parts of a SA impact the most its results, and therefore to what a practitioner should pay attention when designing a SA algorithm for a given scenario [16]. In a subsequent study I analyzed the impact of the various choices for the most important component of a SA algorithm, the acceptance criterion [17].

Having established how different SA characteristics are required to perform well on different scenarios, we move on to relate these characteristics to the particular scenarios encountered by the algorithms. In particular, we focused on two of the most interesting variants of SA from both a theoretical and experimental perspective, its original formulation and the so-called fixed temperature variant (FTA). By again using the automated algorithm design paradigm I have obtained the best possible SA and FTA algorithms for various scenarios. I have subsequently analyzed the conditions under which the algorithms operated in the various scenarios, and I have deter-

mined what are the conditions required for each algorithm to work well. To the best of my knowledge, this is the first cross-problem analysis of SA and FTA in combinatorial optimization that relates the algorithm characteristics to the solution landscape [18].

The final main contribution of this thesis is the proposition of a novel causal framework that models the relationships between the entities involved in the solution of an optimization problem [19]. This work comes from an attempt to generalize the approaches followed in the previous works, providing a solid conceptual framework to support them. I show that this framework is useful in representing the specific research questions discussed in the optimization literature, disambiguating the goals and the findings reported. This is an essential step towards building a coherent connection between the many works in the literature that deal with analyzing, understanding and explaining the performance of optimization algorithms. I also show how this conceptual framework is useful in practice, by applying it to show how existing tools can be used to exploit the knowledge obtained on a set of problems to automatically instantiate an effective algorithm for a new, unseen problem [20]. To the best of my knowledge this is the first example of transfer learning in combinatorial optimization that works across different objective functions.

1.3 List of publications

The research reported in this thesis has been published in the several works.

The component-based analysis of simulated annealing and its acceptance criterion was published in

[16] Alberto Franzin, Thomas Stützle. "Revisiting Simulated Annealing: a Component-Based Analysis". Computers and Operations Research, 2019; 104, 191-206,

[17] Alberto Franzin, Thomas Stützle. "Comparison of Acceptance Criteria in Randomized Local Searches." 13th International Conference on Artificial Evolution (EA2017), 2017; 16-29.

A preliminary version of this research was presented in

[21] Alberto Franzin, Thomas Stützle. "Exploration of Metaheuristics Through Automatic Algorithm Configuration Techniques and Algorithmic Frameworks." 6th Workshop on Evolutionary Computation for the Automated Design of Algorithms (ECADA @ GECCO), 2016; 1341-1347.

The landscape-based analysis of fixed temperature and simulated annealing has been submitted to an international journal, and at the time of writing is under review and available as an institutional Technical Report:

[18] Alberto Franzin, Thomas Stützle. "A Landscape-based Analysis of Fixed Temperature and Simulated Annealing", Technical Report 2021-005, IRIDIA, Université Libre de Bruxelles, Belgium (2021).

The causal framework for optimization algorithms and its application to the transfer learning of algorithm configurations are the subjects of a manuscript currently in preparation to be submitted to an international journal. Preliminary versions of this work and of its results have been published in

[19] Alberto Franzin, Thomas Stützle. "A causal framework for understanding optimisation algorithms." Foundations of Trustworthy AI - Integrating Learning, Optimization and Reasoning (TAILOR @ ECAI2020), Santiago de Compostela, Spain (online), 2020; 140-145.

[20] Alberto Franzin, Thomas Stützle. "Towards transferring algorithm configurations across problems." Learning Meets Combinatorial Algorithms Workshop (LMCA@NeurIPS2020), Vancouver, Canada (online), 2020; 1-6.

1.3.1 Other contributions

In addition to the main contributions that compose this thesis, I have worked on other subjects, related to various extents to the topics of this thesis.

I have contributed to the automatic algorithm configuration problem by studying the impact of parameter transformations, aimed to improve the performance of automatic configurators, in general, and irace, in particular. This work has been published as

[22] Alberto Franzin, Leslie Pérez Cáceres, Thomas Stützle. "Effect of Transformations of Numerical Parameters in Automatic Algorithm Configuration." Optimization Letters, 2018; 12(8), 1741-1753.

The application of automatic methods in less traditional domains is the subject of another series of works I participated to. The automatic configuration of the `gcc` `C/C++` compiler is the focus of

[23] Leslie Pérez Cáceres, Federico Pagnozzi, Alberto Franzin, Thomas Stützle. "Automatic configuration of `gcc` using irace." 13th International Conference on Artificial Evolution (EA2017), 2017; 202-216.

I have also worked on the automatic configuration of databases, to show how an easy to use general-purpose configurator is effective in optimizing the performance of a database.

[24] Moisés Silva-Muñoz, Alberto Franzin, Hugues Bersini. "Automatic configuration of the Cassandra database using irace". PeerJ Computer Science, 2021; 7:e634.

To be successful, the configuration of a database requires a careful setup that allows for consistent observations. The determination of such experimental setup is the subject of another work, presented in the following paper.

[25] Moisés Silva-Muñoz, Gonzalo Calderon, Alberto Franzin, Hugues Bersini. "Determining a consistent experimental setup for benchmarking and optimizing databases." 6th Workshop on Industrial Application of Metaheuristics (IAM @ GECCO), 2021; 1614-1621.

During my time in IRIDIA, I have also worked on more different topics. I have developed and released an R package to learn and use Bayesian Networks from data that contains missing values, a common scenario in many practical applications such as medicine and biology.

[26] Alberto Franzin, Francesco Sambo, Barbara di Camillo. "bnstruct: an R package for Bayesian Network structure learning in the presence of missing data." Bioinformatics, 2017; 33 (8), 1250-1252.

[27] Francesco Sambo, Barbara Di Camillo, Alberto Franzin, Andrea Facchinetti, Liisa Hakaste, Jasmina Kravic, Giuseppe Fico, Jaakko Tuomilehto, Leif Groop, Rafael Gabriel, Tiinamaija Tuomi, Claudio Cobelli. "A Bayesian Network analysis of the probabilistic relations between risk factors in the predisposition to type 2 diabetes." 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2015; 2119-2122.

Finally, I have worked on a technology transfer project between academia and industry, contributing to the development of an anomaly detection application aimed at monitoring networks of Internet of Things (IoT) sensors using packet metadata.

[28] Alberto Franzin, Raphaël Gyory, Jean-Charles Nadé, Guillaume Aubert, Georges Klenkle, Hugues Bersini. "Philéas: Anomaly Detection for IoT Monitoring." 32nd BeNeLux Conference on Artificial Intelligence (BNAIC2020), 2020; 56-70.

1.4 Thesis outline

The next chapter is devoted to review the preliminary concepts of optimization problems and the algorithms to tackle them, with a focus on SLS methods. The chapter continues with an introduction to the problems of automatically configuring and generating an optimization algorithm for a given scenario, and a review of the main approaches proposed. These techniques are the starting point for our subsequent analysis of simulated annealing.

Chapter 3 is devoted to simulated annealing and its variants. By making use of automatic algorithm configuration and design tools, I am able to reimplement several existing SA algorithms from the literature, improve their performance, and identify the key components that determine how effective a SA is for a certain scenario.

In Chapter 4 we investigate the relationship between algorithmic structure and solution landscape. We compare the performance of two SA variants, with the traditional cooling scheme and with a fixed temperature, on different problems and instance classes. We measure properties of the landscape as seen by the algorithms and relate them to the results obtained by the algorithm, observing the conditions under which algorithm perform best.

Chapter 5 is then devoted to the development of a causal framework whose goals are to relate the elements involved in the solution of an optimization problem and to relate various approaches in the literature proposed to understand optimization algorithms. We then show how to exploit this conceptual framework to perform transfer learning of algorithm configurations across different objective functions.

In Chapter 6 we finally summarize the contribution of this thesis and outline several potential research directions to improve, apply and expand the work here presented.

Two additional documents are included as appendices. In Appendix A we include a comparison of acceptance criteria in SLS algorithms. We experimentally compare several acceptance criteria that appear in the literature in algorithms that can be considered SA variants, and observe which ones perform consistently well. Appendix B contains instead the analysis of the impact of different transformations on the parameter space. We study how certain modifications of the parameter space affect the efficiency of configuration tools, both with positive and negative effects, and how the choice of the appropriate transformation to use is non-trivial. We then propose an automatic procedure to select the appropriate transformation of the parameter space to use in a practical configuration task.

Background 2

In this section we revise the preliminary concepts upon which this research is based. We start by reviewing the field of optimization, focusing on computationally hard problems. We then describe the various approaches to tackle those problems, in particular the family of methods called Stochastic Local Search algorithms. We then proceed to review automatic methods to optimize and generate state-of-the-art methods with limited human intervention.

2.1 Optimization

2.1.1 Optimization problems

One classic example of an optimization problem is called Traveling Salesperson Problem (TSP). Given a set of locations and the distances between every two locations, the TSP requires a salesperson to visit all the clients one single time before returning to the starting point, traveling the shortest distance possible [29, 30]. The TSP is arguably one the most studied optimization problems, and the one often taken as example, due to the simplicity of the formulation and the practical importance. It arises not only when planning a trip, but also when drilling holes in a circuit board or sequencing DNA segments, just to name a couple of examples. Several variants of the classic TSP definition also exist: partitioning the locations into different subroutes results in a new problem called Vehicle Routing Problem [31, 32], while if each location can be visited only in predefined instants we have the Traveling

Figure 2.1: *Commis Voyageur* tour through 25 cities in Germany, from the manual "Der Handlungsreisende - wie er sein soll und was er zu thun hat, um Auftraege zu erhalten und eines gluecklichen Erfolgs in seinen Geschaeften gewiss zu sein - Von einem alten Commis-Voyageur" (1832) [29]

Salesperson Problem with Time Windows [33]. A common variant of the TSP is the *metric* TSP, whose distances satisfy the triangle inequality: going from point a to point b, and from point b to point c cannot be quicker than going directly from a to c. The TSP is symmetric, without directionality on the edges; another variant is the Asymmetric TSP, where the cost of (u, v) can be different from (v, u) [34, 35].

Formally, we define an instance $\mathcal{I}$ of a problem P as a pair (S, f), where f is a question to be answered, and S, called the *search space* or *solution space*, is a set of possible (candidate) *solutions* s, that is, answers to f that are correct for $\mathcal{I}$. The problem P is therefore the collection of all possible $\mathcal{I}$ that share the same f.[1]

An optimization problem is a problem for which f defines a (possibly non strict) total ordering over S, usually $f : S \mapsto \mathbb{R}$. In this case f is called *objective function*. If the problem is a multiobjective optimization one, then $f : S \mapsto \mathbb{R}^{N_{obj}}$, where N_{obj} is the number of objectives; the total

[1] Here we distinguish between different problems only based on the objective function, without considering the different possible structures for the S. We consider different structures for the S as different variants for the same problem; however, certain special structures are sometimes considered standalone problem. We will ignore this ambiguity and treat different structures for S as different problems or different problem variants on a case-by-case basis according to existing conventions, as it does not affect the rest of this thesis.

ordering is relative to each objective, while across objectives there may be only a partial ordering, and the relations between solutions are defined in terms of *dominance*. In what follows we consider single objective problems; the extension to multiobjective ones is straightforward.

An optimization problem can be either a minimization or a maximization one. The set of optimal solutions $S^* \subseteq S$ of an instance $\mathcal{I}$ of a generic minimization problem P is defined as

$$S^* = \{s^* \mid f(s^*) \leq f(s)\, \forall s^* \in S^*, s \in S\}. \tag{2.1}$$

The goal is to find one $s^* \in S^*$, and for ease of notation we write

$$s^* \mid f(s^*) \leq f(s)\, \forall s \in S. \tag{2.2}$$

Sometimes we will denote $f(s^*)$ as z^*. If the problem is instead a maximization one, the goal is to find a s^* such that $f(s^*) \geq f(s)\, \forall s \in S$. Since $f(s^*) \geq f(s)\, \forall s \in S \Leftrightarrow -f(s^*) \leq -f(s)\, \forall s \in S$, it is easy to transform a maximization problem into a minimization one. Without loss of generality, unless otherwise specified, we will consider minimization problems.

A *decision* problem is a problem for which $f : S \mapsto \{\mathrm{Y}, \mathrm{N}\}$. Typical decision problems ask whether feasible solutions exist, or there are feasible solutions that respect some desired characteristics. For example, the decision form of a minimization problem requires to find, given a parameter k, if there exists $s \in S$ such that $f(s) \leq k$.

If S has finite cardinality for any possible $\mathcal{I}$ of P, then P is a *combinatorial* or *discrete optimization* problem [36]. Combinatorial optimization problems arise in many fields, whenever a problem can be formulated in terms of choosing subset of elements from a (finite) set, ordering a set of objects, and in general, counting elements. Several typical combinatorial optimization problems, including the TSP, are defined over graphs, or translated into graph problems. A problem where at least some of the variables are restricted to integer values is called an *integer programming* problem. Combinatorial optimization and integer programming problems compose the class of *discrete optimization* problems. These three classes are closely related, and in practice, with a slight and often innocuous abuse of terminology, can be considered the same.

Conversely, *continuous optimization* deals with problems for which S has infinite possible solutions, usually because the solutions in S are real-valued. Many machine learning problems belong to this category. Some problems are said to be mixed-integer, if the solutions contain both a discrete and a real-valued part.

Sometimes it is convenient to represent a problem as a *mathematical program*, that explicitly relates the objective function, the constraints, and the variables of the instance. A generic formulation is

$$\min \quad f(\mathbf{x}) \text{ subject to} \tag{2.3}$$
$$\mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{2.4}$$
$$\mathbf{x} \in X \tag{2.5}$$

where the objective function f is explicitated for the set of variables $\mathbf{x}$, $\mathbf{A}$ is a matrix containing the coefficient for the constraints, $\mathbf{b}$ is the vector of right-hand-side coefficients, and X is the domain of admissible values for the $\mathbf{x}$.

If the objective function is linear, the constraints can be formulated as linear inequalities, and $\mathbf{x} \in \mathbb{R}$, the problem can be formulated as a *linear program* (LP). If the model is linear but $\mathbf{x} \in \mathbb{Z}$ the formulation is an *integer programming* (IP) model; if variables can be either real-valued or integer we have a *mixed-integer (linear) program* (MIP or MILP, that usually include IP). The same formulation can, however, be used to represent nonlinear, nonconvex, or semidefinite programs, whose definition goes beyond the scope of this thesis.

For our example the TSP, an instance $\mathcal{I}$ is a complete graph $G = (V, E)$ with n nodes and $n \cdot (n-1)/2$ edges. The graph is weighted, meaning that there is cost function $c : E \mapsto \mathbb{R}$ that for every edge $e = (u, v)$, $c(e) = c_{uv}$ computes the cost of going from u to v; the cost function is symmetric, that is, $c_{uv} = c_{vu}\ \forall (u, v) \in E$. If an edge e_m is missing, we can consider its cost $c(e_m) = \infty$. The objective is to find a permutation π of the nodes in V that minimizes

$$c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}. \tag{2.6}$$

The TSP is an IP problem for which several formulations have been proposed [37]. In such models the variables $\mathbf{x}$ are often binary variables associated to the edges, where $x_{ij} = 1$ if the edge (i, j) is in the solution, and 0 otherwise. From this point of view the solution is composed by a set of n edges, that is, n variables set to 1, and the remaining ones to 0. The constraints are needed to ensure that, for example, each node has degree 2, or no cycle of length shorter then n exists. However in our case we will always consider feasible solutions for the TSP, as permutations of the n nodes; in cases like this, the constraint matrix is implicit and can be omitted.

2.1.2 Computational complexity

Optimization problems can be classified according to how difficult it is to solve them, that is, how fast it is the best algorithm for them, when measured in terms of worst-case runtime with respect to the size of $\mathcal{I}$. Here we give a brief and somewhat rough overview of computational complexity; a more precise explanation can be found in references such as [38]. We say that an algorithm is $\mathcal{O}(f(x))$ if its runtime for sufficiently large sizes grows asymptotically with $f(x)$, in the worst case [39].

Many problems of interest are said to belong to the complexity class $\mathcal{NP}$, for non-deterministic polynomial. A problem is in $\mathcal{NP}$ if, given a solution s for an instance $\mathcal{I}$, we can verify that s is actually a valid solution for $\mathcal{I}$ in polynomial time.

Let's consider another classic optimization problem, the Minimum Spanning Tree (MST). Given a weighted graph $G = (V, E)$, a graph whose set of nodes is V and whose edges $e \in E$ have an associated cost $c(e)$, the goal is to find a set of edges E^* such that (i) every node in V is touched by at least one edge in E^*, and (ii) the total cost of the edges in E^* is minimum. For the MST there are algorithms (Kruskal and Prim, [40, 41]) that run in time $\mathcal{O}(|V|^2 \log |V|)$, as the size of a graph is usually defined with respect to the number of its nodes $|V|$. These algorithms are provably optimal, meaning that they return the correct solution, and that no algorithm that does it in a lower asymptotic time exists. Prim's algorithm can be faster for dense graphs when appropriate data structures are used; other more complex algorithms can be faster for specific classes of graphs. The MST is also trivially in $\mathcal{NP}$, since the construction of the optimal solution serves also as verification.

Another notable optimization problem is the Assignment problem that models the assignment of n tasks to n different people, and admits an optimal algorithm that runs in $\mathcal{O}(n^3)$ [42]. This is higher than the algorithms for MST, meaning that the problem is more difficult, but it still runs in a time polynomial in the size of the instance. Both problems belong to the complexity class $\mathcal{P}$, the class of algorithms for which an optimal, polynomial-time algorithm is known; problems in $\mathcal{P}$ are informally said to be easy.

For many other common optimization problem in $\mathcal{NP}$, such as the TSP, there is instead no known algorithm that produces one optimal solution in time polynomial in the instance size. They are informally said to be hard, as the best known algorithms run in superpolynomial or exponential time, and it can be proved that each one of these problems is "as hard" as the other ones; we call them $\mathcal{NP}$-hard [43]. This means that, for any one of them, we can transform (reduce) an instance of a $\mathcal{NP}$-hard problem P_1 into an

instance of another $\mathcal{NP}$-hard problem P_2 in polynomial time.

A problem can be as hard as any $\mathcal{NP}$ problem, meaning that any problem in $\mathcal{NP}$ can be reduced to it in polynomial time, but not be in $\mathcal{NP}$. Problems that are both in $\mathcal{NP}$ and $\mathcal{NP}$-hard are said to be in the $\mathcal{NP}$-complete class. In particular, decision problems, and the decision version of optimization problems, belong to this class.

The first problem shown to be $\mathcal{NP}$-complete is Circuit (or Boolean) Satisfiability (SAT) [44, 45], a basic problem in propositional logic that asks to find, if it exists, an assignment of Boolean variables that satisfy a given formula. The corresponding optimization problem requires to find the assignment that minimize the number of violated clauses. Soon after, several other problems have been reduced to SAT [46], and the list of $\mathcal{NP}$-complete problems continues growing today.

Whether the $\mathcal{NP}$-hard problems actually admit a polynomial time algorithm or not is still an open question, known as the $\mathcal{P}$ vs $\mathcal{NP}$ problem. Finding a polytime algorithm for any of the $\mathcal{NP}$-hard problems would mean that all the problems in $\mathcal{NP}$ can be solved in polynomial time; many researchers believe this is not the case, and such an algorithm does not exist. In practice, we prefer to assume that $\mathcal{P}$ is not equal to $\mathcal{NP}$, abandoning the idea of finding a polytime algorithm for $\mathcal{NP}$-hard problems, and revert to other techniques, described in Section 2.2.

It is not always easy to tell whether a problem is actually in $\mathcal{P}$ or not. For example, SAT instances of a particular variant called 2-satisfiability, where boolean formulas are in the form of disjunctions of conjunctions of couples of literals (2-CNF, where a literal is either a variable or its negation) can be solved in polynomial time, while if the literals are grouped in triplets the problem (3-CNF) is $\mathcal{NP}$-complete. A LP can be solved in polynomial time [47], but solving a MIP is $\mathcal{NP}$-hard [46]. Problems with certain structures are solvable in polynomial time; for example, problems structured as matroids can be solved optimally with (polynomial) greedy algorithms, such as the MST.

A common misconception is that $\mathcal{NP}$-hardness is related to the (exponential) size of the solution space. This is however not true, and a simple counterexample is the 2-CNF variant of SAT mentioned above: for n variables there are in total 2^n possible assignments, but as we have seen we can solve an instance in polynomial time. However, an instance of a $\mathcal{NP}$-hard problem necessarily has a superpolynomial number of candidate solutions, otherwise enumerating and evaluating all of them would be a polytime algorithm for the problem.

Some important problems, such as factoring a number into its prime factors, or checking whether a graph is isomorph to another one, are not

proven to be $\mathcal{NP}$-hard, but no polytime algorithm is known. For all the practical purposes, they are treated as hard ones.

One important point to make is that this problem classification holds at the worst case: while in theory all the $\mathcal{NP}$-hard problems are as hard as the other ones, in practice we can solve some of them more easily than other ones. We can solve to optimality TSP instances with tens of thousands of nodes, while for the quadratic assignment problem (QAP) we can find the optimal solutions for problems with few tens of variables only for instances belonging to special classes [48, 49]. Similar considerations can be made for the algorithms. LPs can be solved in polynomial time by the Karmarkar algorithm [47], but in practice the simplex algorithm is often preferred, despite having exponential complexity at the worst case: in fact, for the average case the simplex algorithm exhibits quadratic complexity [50]. Whether the worst case complexity of the simplex algorithm can be lowered to polynomial is still an open question.

2.1.3 Some $\mathcal{NP}$-hard problems

The TSP has already been introduced in the beginning of this section; here we introduce the other $\mathcal{NP}$-hard problems that will be used in the rest of this thesis. These problems are computationally hard at the general case; specific instance classes or instances of limited sizes can, however, be solved to optimality by exact or heuristic algorithms (see Section 2.2).

Quadratic Assignment Problem One of the most studied problems is the Quadratic Assignment Problem (QAP) is a problem that models the assignment of n facilities to n locations [51, 52]. Between two locations i and j we are given the distance d_{ij} and the flow f_{ij}. An optimal solution of a QAP instance is a permutation π of the facilities that minimizes

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}d_{\pi(i)\pi(j)}. \tag{2.7}$$

We will come back on the QAP in the remainder of the thesis.

Permutation Flow Shop Problem The Permutation Flow Shop Problem (PFSP) models several real-world scheduling problems [53, 54, 55]. It requires to schedule n jobs on m machines, in the same order on all the machines from 1 to m, finding a permutation π of the jobs such that some objective is minimized. Each job i takes p_{ij} units of time to run on machine j. All the jobs are available at time 0, and no preemption or concurrency are

allowed. In this thesis we consider two common objectives, the Makespan objective (PFSP-MS) [56], defined as the completion time of the last job on the last machine

$$C_{max} = C_{\pi(n),m} \tag{2.8}$$

and the Total Completion Time objective (PFSP-TCT) [57, 58], the sum of the completion times of all the jobs on the last machine

$$\sum_{i=1}^{n} C_{\pi(i),m} \tag{2.9}$$

where C_{ij} is the completion of job i on machine j.

2.2 Solving $\mathcal{NP}$-hard problems

Assuming that a problem does not admit a polytime algorithm does not mean we cannot cope with it.

Several techniques exist, that we can choose depending on our goals and possibilities. Some methods are guaranteed to obtain the (an) optimal solution, albeit in exponential time. The combinatorial explosion of the search space makes these approaches infeasible for sufficiently large instances (depending on the specific problem, and instance characteristics), so we might need to forsake optimality and settle for "good" solutions that we can obtain in a reasonably short time. Methods for this task, which we call *heuristics*, can be divided in several categories: problem-specific algorithms, that in some cases are guaranteed to find solutions that approximate the optimal ones by a certain factor, and local search methods, that traverse the search space following general ideas and need to be instantiated to the specific problem.

This classification is however not fixed, and a method from one category can be hybridized with other ones or modified to fall into a different class: heuristic algorithms can be made evaluating the entire search space, and exact algorithms can be sped up by using heuristic search strategies, while maintaining the certification of optimality for the final solution. Analoguously, local search methods can easily be combined together, adopt simultaneously different approaches, or integrate limited exact exploration.

2.2.1 Exact methods

Exact or complete methods are algorithms that always return, in theory, an optimal solution, or prove that no feasible solution exist.

The simplest idea is to enumerate all the solutions in S and evaluate them. As this quickly becomes infeasible for modest-sized problem instances, smarter but still generic techniques perform an implicit enumeration and employ clever techniques to prune the areas of the search space for which it can be established that it does not contain the optimal solution. At the worst case they have the same complexity of brute-force enumeration, but in practice they can work quite well. The main example of such techniques is the *branch-and-bound* [59, 60]. The set of solutions S is seen as a rooted tree; each node contains a subset S_v of S, with the root containing the whole S. Branching on a node means to partition S_v according to some choices, often assigning to a variable its possible values. Subtrees are therefore consistent with the choices made at the upper levels, and disjoint between them. At each node it is also possible to estimate upper and lower bounds on the optimal solutions, or to establish infeasibility in the whole subtree. This is useful to discard entire subtrees, when it can be proved that the optimal solution is somewhere else in the tree. The search in the tree can be sped up by evaluating at each node additional cuts [61, 62], derived constraints that can rule out more solutions. This technique is called *branch-and-cut* [63].

Implicit enumeration techniques can be coupled with problem relaxations that attempt to solve the problem at every node. Notably, MILP models are often relaxated by removing the integrality constraint from the variables; they are then solved using branch-and-bound techniques where, at each node, the relaxated version is tackled as an LP. Variables that do not take integer values after this step are branched on, and the process is repeated on the reduced subproblem. This is the general idea underlying software packages such as CPLEX or Gurobi [64, 65, 66].

For some problems, specific algorithms have been developed. The Held-Karp algorithm for the TSP is a dynamic programming approach that performs the same enumeration of the solutions ($\mathcal{O}(n!)$) but "collapsing" the several symmetric branching paths in a single evaluation. The complexity of the algorithm is now $\mathcal{O}(2^n\sqrt{n})$ [67].

When exact method become infeasible, one must revert to algorithms that provide possibly suboptimal solutions in a short time. These methods are called *heuristics*, from the Greek "discover".

2.2.2 Approximation algorithms

An intermediate class is composed by algorithms that return solutions that are provably not too worse with respect to the optimal ones [68]. They are called approximation algorithms, and certify that the final solution differs at

most by factor $\rho = \hat{z}/z^*$ from the optimal ones, where $\hat{z} = f(\hat{s})$ is the cost of the solution $\hat{s}$ found by the algorithm; for maximization problems instead $\rho = z^*/\hat{z}$. For example, the Christofides algorithm for the metric TSP [69] is based on a MST relaxation and has $\rho = 3/2$. A simple greedy algorithm for KP that sorts the items by their profit-to-weight ratio has $\rho = 2$.

A polytime approximation algorithm that can return a solution within a factor $(1 + \epsilon)$ of the optimal solution, for any $\epsilon > 0$, is called Polynomial-Time Approximation Scheme (PTAS). Usually, also the runtime is a function of ϵ; a subclass of PTAS algorithms if called Fully Polynomial Time Approximation Schemes (FPTAS) and requires the algorithm to run in time polynomial both in the size of the problem n and in $1/\epsilon$.

Not all the problems admit a FPTAS, a PTAS or even an approximation algorithm, unless $\mathcal{P} = \mathcal{NP}$. In the following Sections we describe some of the search paradigms that can be used to derive heuristic algorithms.

2.2.3 Stochastic local search algorithms

Local search algorithms are heuristic algorithms that do not (necessarily) evaluate the whole solution space; they are therefore not guaranteed neither to find the optimal solution nor to establish infeasibility of an instance. Therefore, when using a local search we usually settle for a locally optimal solution.

2.2.3.1 Problem-specific heuristics

Some of these methods are algorithms tailored for specific problems, relaxating constraints or making assumptions to solve an easier version of the original problem.

For example, we will use the NEH heuristic for the PFSP. First, the jobs are sorted by non-increasing sums of processing times on the m machines; then, the first two jobs in this order are scheduled, as if they were the only jobs in the instance. Then the remaining jobs are inserted, one by one, such that they minimize the objective function value at the local step. This procedure evaluates $n \cdot (n + 1)/2 - 1$ sequences instead of the $n!$ ones of the complete problem.

In the case of the TSP, assuming that a locally optimal decision must apply also to the global problem gives us the Nearest Neighbour (NN) heuristic: we start from a node, and visit all the nodes by moving each time to the closest one among the nonvisited nodes; as last step, we return to the starting point. The algorithm is very simple but, as the TSP is not a matroid, this greedy algorithm is not optimal; it also performs rather poorly in practice

[70]. More sophisticated examples exist, such as the Lin-Kernighan algorithm (LK with its evolution LKH [71, 72]), that follows a generalization of the 2-opt swap move [73, 74], one of the best performing heuristics for the TSP. LK and LKH, however, follow patterns that we describe in more detail in the remainder of this section.

2.2.3.2 Constructive and perturbative local search algorithms

Constructive heuristics iteratively build a solution at each iteration, starting from an empty one. A notable example is the aforementioned Nearest Neighbour algorithm for the TSP. These methods can be made systematic by structuring the search space as a tree. Starting from an empty solution (or a conventional starting point) at the root node, we can branch following all possible choices; descending into the tree level after level the available choices reduce, until the leaves where we find one feasible solution. If at any point the partial solution observed is already infeasible (or can be ruled out e.g. for being surely pejorative of another one we already know) we can *backtrack*, stop the descent and move to another branch of the parent node. This tree structure also prevents visiting each solution more than once, but as the number of paths to be evaluated is the number of solutions (hence, at least exponential for $\mathcal{NP}$-hard problems) this approach quickly becomes infeasible, and the search tree has to be pruned somehow.

Conversely, perturbative heuristics start from an *initial solution* and at each iteration change one part of the solution: this change is called *perturbation*.

We give here some definitions. A *neighbourhood* $\mathcal{N}$ is a function $\mathcal{N} : S \mapsto 2^S$ that maps a solution s to a set of *neighbouring solutions* (or simply neighbours) $\mathcal{N}(s) \subseteq S$. The cardinality of $\mathcal{N}(s)$ is the *size* of the neighbourhood. A *local optimum*, or locally optimal solution, is a solution $\hat{s}$ such that $f(\hat{s}) \leq f(s)\ \forall s \in \mathcal{N}(\hat{s})$. If $f(\hat{s}) < f(s)\ \forall s \in \mathcal{N}(\hat{s})$ we say that $\hat{s}$ is a *strict* local optimum, otherwise it is a *non-strict* one. Choosing one solution s' in the neighbourhood of s is called a *move*; when a move is done we also say it is *accepted*. A neighbourhood is often *simple*, that is, it is defined as the minimum perturbation necessary to change a solution. Simple neighbourhoods are relatively quick to explore, a feature useful to evaluate as many moves as possible; other strategies choose, however, complex or large neighbourhoods, to cover a larger portion of the search space. The neighbourhood is also not necessarily static during the entire search, and it can be enlarged, restricted, or even replaced by a different one. These strategies are discussed later in this section.

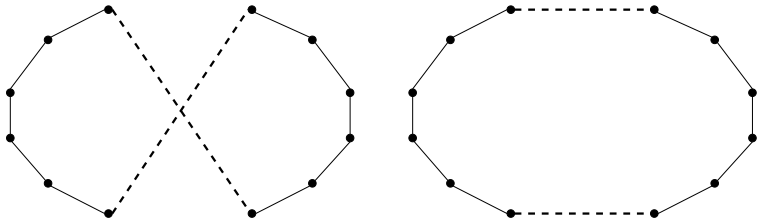

Figure 2.2: Representation of a 2-opt move: crossing edges are removed and replaced with the two shortest edges that close the cycle.

A desirable feature for a neighbourhood is the possibility of quickly computing the solution quality difference between two solutions, thanks to the local change, rather than having to re-evaluate the entire solution: this is called *delta evaluation*.

A common example is the 2-opt exchange neighbourhood for the TSP, represented in Figure 2.2. It swaps two edges in a solution. Intruitively, if two edges are crossing, we can make the total tour shorter by untangling it. From the base solution we remove edges (u_1, v_1) and (u_2, v_2) and we add (u_1, v_2) and (u_2, v_1), ensuring that the solution remains a tour, not two disjoint subtours. Since the solution has n edges, we can choose $\binom{n}{2}$ edges to remove; of these, we also have to exclude the edges that are adjacent in the solution to the ones we have selected, since exchanging them would leave the solution unchanged. This leaves us with $n \cdot (n-3)/2$ possible moves in the neighbourhood. For all these possible moves we can compute the difference in terms of solution quality using the delta evaluation by subtracting the cost of the edges removed, and adding instead the cost of the new edges: so, the cost of a new solution $s' \in \mathcal{N}(s)$ obtained by replacing edges e_1 and e_2 with edges e_3 and e_4 is $f(s') = f(s) - (c(e_1) + c(e_2)) + (c(e_3) + c(e_4))$. The cost of this operation is constant, compared to the linear cost of entirely re-evaluating $f(s')$; when evaluating the whole set of possible moves in the neighbourhood, using the delta evaluation thus reduces the total cost from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

In general, a neighbourhood that changes k components of the solution is called a k-opt move. The higher the k, the more candidate solutions analyzed, but also the higher the cost of the total evaluation. In practice, a tradeoff between covering the search space and focusing the search around good solutions has to be chosen, and this problem is called the *exploration versus exploitation tradeoff*. Before delving into this topic in the next section, we present here the two most basic paradigms of perturbative local search algorithms, called *Iterative Improvement* algorithms; the rule followed to

Algorithm 2.1: BI LOCAL SEARCH

Data: Initial solution s_0, neighbourhood $\mathcal{N}$
Result: local optimum $\hat{s}$

```
ŝ = s = s_0;
while ∃ s' ∈ N(s) | f(s') ≤ f(s) do
    s = arg min_{s'} f(s');
end
return ŝ = s;
```

Algorithm 2.2: FI LOCAL SEARCH

Data: Initial solution s_0, neighbourhood $\mathcal{N}$
Result: local optimum $\hat{s}$

```
ŝ = s = s_0;
for i ∈ 1 ... |N(s)| do
    if f(s_i) ≤ f(s) then
        s = s_i;
        continue;
    end
end
return ŝ = s;
```

identify the next best solution is called *pivoting rule*.

Best improvement A best improvement (BI) local search starts from a given initial solution one and terminates when a locally optimal one is found, each time evaluating all the possible moves in the neighbourhood and accepting the one that yields the maximum improvement (Algorithm 2.1). BI is a deterministic algorithm, and it is also known as Hill Climbing or Steepest Ascent/Descent – depending on whether the problem is a maximization or minimization one [75].

First improvement A first improvement (FI) algorithm also starts from an initial solution and traverses neighbouring solutions until it finds a local optimum (Algorithm 2.2). The difference with BI is that the first improving solution encountered when scanning the neighbourhood is immediately accepted. FI is also deterministic, but the evaluation order in the neighbourhood matters. It can be much faster than BI, since usually it does not need to scan the whole neighbourhood.

In practice, FI and especially BI can be used as described only with neighbourhoods that can be explored thoroughly in an efficient way. Large neighbourhoods can even be exponential in size, so in practice we may not be able to cover all of them, and settle for a limited exploration. On the other hand, FI and BI will discover the optimal solution of certain problems called *convex*, which admit only one local optimum, which is thus the global one.

2.2.3.3 Intensification vs diversification

Heuristic algorithms do not cover the whole search space, except for small instances. So, unless we already know a way of pruning the search space effectively, we are faced with the task of choosing a local search paradigm and a neighbourhood that covers as much of the search space as possible, while converging to good solutions in the most efficient way possible. Assuming that only a limited number of solutions can be evaluated, the two objectives are in contradiction: the first goal, called *exploration* or *diversification* requires to distribute the candidate solutions on the search space, having therefore very diverse solutions. One unfortunate consequence is that potential improving moves near the ones we are evaluating are overlooked. The second goal, called *exploitation* or *intensification*, aims instead to focus the search in narrow regions around the good solutions encountered, discovering more likely similar, better solutions. This comes at the expense of looking at more diverse moves, in different regions of the search space, where potentially better solutions could be found.

Let's consider the family of k-opt moves for the TSP as an example. The 2-opt neighbourhood has quadratic size, and can be explored relatively easily: thus, we can afford to find an improving move near the one we currently have even for relatively large instances. However, if we currently are in a region of the search space with only bad solutions, it will take a lot of moves to finally reach good solutions; and when we find a local optimum and decide to stop the search, this one might be quite poor. As k grows, we will cover always more solution space, thus locating more easily better solutions; and moving far away from bad regions will also be easier. The drawback is the computational cost required for the scan, significantly higher than for lower values k: for the TSP the complexity of exploring a k-opt neighbourhood is $\mathcal{O}(n^k)$ – even without taking into account the increasing cost of the delta evaluation.

So when designing a local search algorithm we aim to balance, and possibly oscillate between, these two goals: moving towards more diverse solutions when the quality of the current ones is not low, focus around the better

ones when we find them, balancing. This is know as the *diversification-intensification* or *exploration-exploitation tradeoff*, and how to achieve this is the dilemma of local search algorithms. There is no general solution to this problem, as each problem and problem instance is essentially different from the other ones, and requires therefore different settings; as tailoring a specifc method for each instance is clearly infeasible, we need to maintain a certain generality. The design of an effective local search algorihm essentially boils down to coping with this dilemma, and the approaches we are discussing more in detail in the following sections can be considered attempts to balance the two behaviours.

2.2.3.4 Simple local search techniques

First Improvement and Best Improvement strategies are usually fast but do not perform well in terms of final solution quality, as they converge to the first local optimum available. To move the search towards hopefully better solutions, one first strategy is to expand the range of the search. Some methods are presented here. They introduce some diversification with respect to pure iterative improvement methods, but the following algorithms and techniques maintain an overall strong intensification.

Large neighbourhoods and pruning. Simple neighbourhoods allow an easy intensification, but limited exploration. This can mitigated by having FI or BI analyzing larger neighbourhoods, which allow to find possibly better solutions and to more easily transition towards diverse solutions, at the expense of a higher cost of a full neighbourhood evaluation.

One example is to use a k-exchange neighbourhood (with $k > 2$) instead of the simple 2-opt for the TSP. The size of the neighbourhood is $\mathcal{O}(n^k)$, with a higher chance of containing better solutions but also a size that can be practically intractable for sufficiently large values of k. Large neighbourhoods can even be of size exponential in the size of the problem, making their evaluation a possibly $\mathcal{NP}$-hard problem in itself to be tackled using exact or heuristic methods.

One possibility, however not limited to large neighbourhoods, is to prune the neighbourhood using some heuristic rules (e.g. k-nearest neighbours for each node in the TSP). Another general technique is to employ so-called *don't look bits*, flags associated to solution components that indicate whether in the previous steps a move involving the relative components was successful or not. Such techniques have been observed to yield improvements in both runtime and final solution quality obtained.

Variable Neighbourhood Descent. The Variable Neighbourhood Descent (VND) algorithm is based on the idea that a locally optimal solution for a neighbourhood may not be one for a larger neighbourhood. The basic VND starts with a regular FI or BI over a neighbourhood, and once it finds a local optimum the search continues considering a larger neighbourhood. A simple case is to use the k-opt neighbourhood, with k ranging from 2 to a given $k_{\max}$ for the TSP. Once the last neighbourhood structure fails to improve the incumbent solution, the algorithm terminates.

Variable Depth Search. One alternative approach is to compose a sequence of elementary steps into a single complex move. The length of the sequence of simple steps is variable from one complex step to another. Algorithms using this approach are called Variable Depth Search algorithms (VDS).

A notable example is the Lin-Kernighan heuristic (LK) for the TSP, an algorithm later improved by Helsgaun (and therefore also known as LKH), that builds a complex step by composing exchange moves on top of each other; several strategies are used to guide this composition, for example ensuring that one move does not revert another move in the complex step.

Dynasearch. Another strategy is called Dynasearch, that evaluates the improvements yield by all the possible complex steps that can be composed, and chooses the one that gives the highest improvement; this composition evaluation is done using dynamic programming. To ensure the correct evaluation of the improvement, the simple steps need to be independent, in the sense that their impact can be measured locally and is not influenced by the other simple steps performed in the complex move.

2.2.4 Stochastic local search methods

One radically different approach to balance intensification and diversification is to introduce stochasticity in the search process.[2] Randomization in algorithms is very effective, in particular when a deterministic choice is expensive, difficult, or impossible, and the cost of a wrong choice is very high when compared to the best or average case. Local search algorithms following this philosophy are called stochastic local search algorithms (SLS). As these methods are general templates to follow when instantiating an effective heuristic algorithm for a specific problem, they are also called *metaheuristics*, and we will use these two definitions interchangeably.

[2] In a sense, methods based on pure intensification can also be considered stochastic, because their results still depend on the initial solution provided, and they cannot, in general, find a global optimum.

Randomization can be applied to many operations in local search algorithms. The algorithms we are going to discuss can use stochasticity when picking a starting point, choosing the solutions to evaluate, determining whether worsening moves are to be accepted, update the control parameters, and in general when making decisions that involve uncertainty in the input or in the output.

Decisions that in the simplest iterative improvement algorithms were implicit, such as when to terminate, have to be made in a different way. Thus, we need to introduce some additional algorithmic mechanisms to deal with this increased degree of freedom, and in particular we need to introduce a *termination condition*. There are several possibilities to choose from: fixed conditions such as a given runtime or a certain number of moves evaluated, or adaptive ones, e.g. terminating when no improvement has been recorded in the latter stages of the search; we will present more in detail these possibilities in Section 3.3.

In this section we are going to present some of these stochastic local search algorithms, starting from the most basic ones and going to more advanced strategies. As the more complex algorithms often extend and combine simpler ones, we will try to relate each method to each other, emphasizing the connections and the novel contributions of each algorithm. We will follow a component-based view of algorithms, that considers an algorithms as a combination of basic building blocks, each one serving a particular purpose. This decomposition is crucial in the entire development of this work, and will be discussed more thoroughly in Section 2.3.3.

Before proceeding, we note that randomization can also be applied to exact methods to improve the performance of the search. Complete algorithms (and commercial software packages) in practice exhibit an erratic behaviour, and rely heavily on heuristics methods in the both attempting to solve the instance at each node and in the tree traversal [64, 76, 77].

2.2.4.1 Simple stochastic local search algorithms

Some stochastic local search algorithms are simple but introduce an idea of local search that is very different from the one seen so far. Rather that evaluating an entire neighbourhood, or a significant portion of it, looking for the best solution, these new algorithms often, though not always, compare solutions one against another. The neighbourhood becomes therefore a tool for generating one new solution, rather than the boundary of the search. As a consequence, these algorithms may lose the ability of determining when a solutions is a locally optimal one; to mark this difference, we will denote the final solution of a SLS as $\hat{s}$.

Algorithm 2.3: PII local search

Data: Initial solution s_0, neighbourhood $\mathcal{N}$, parameter T
Result: final solution $\bar{s}$

1 $\bar{s} = s = s_0$;
2 **while** *termination condition is not met* **do**
3 | choose $s' \in \mathcal{N}(s)$ randomly;
4 | **if** $f(s') \leq f(s)$ *or* $\exp(-\Delta(s', s)/T) \geq U[0, 1]$ **then**
5 | | $s = s'$;
6 | **end**
7 **end**
8 return $\bar{s} = s$;

Probabilistic Iterative Improvement. Probabilistic Iterative Improvement (PII) is the first algorithm in this overview that occasionally accepts worsening moves in the search. It does so stochastically, with a probability that depends on the relative worsening of the solution quality: moves that yield little worsening are more likely to be accepted than moves towards solutions that are much worse than the incumbent one; improving moves are always accepted.

The novelty of this method is located in a specific component, called the *acceptance criterion* (line 4 of Algorithm 2.3). This criterion is known as *Metropolis condition*, and it is rooted in the work of Nicholas Metropolis in statistical physics; because of this, PII is also known as Metropolis algorithm among the other names, see Chapter 4. It accepts a worsening move with a probability $\exp(-\Delta(s', s)/T)$, where T is a parameter called *temperature* and $\Delta(s', s)$ is the worsening of solution quality of the new solution s' over the incumbent solution s. The value of T controls the exploration/exploitation tradeoff: "high" values for T yield an explorative behaviour, while "low" values enforce more intensification – where "high" and "low" depend on the specific problem instance.

The Metropolis condition, and the algorithms derived from it, will be described more in detail in the following sections, in particular, Section 3.3.

Simulated Annealing. Simulated Annealing (SA) extends PII by introducing the possibility of adapting the value of T during the search, with the idea of gradually transitioning from an explorative to an eploitative behaviour by gradually going from a "high" value of T to a "low" one. SA needs therefore a control mechanism for T; this is achieved by three functions,

Algorithm 2.4: SA LOCAL SEARCH

Data: Initial solution s_0, neighbourhood $\mathcal{N}$, control parameters
Result: final solution $\bar{s}$

```
1  s̄ = s = s_0;
2  initialize T parameter;
3  while termination condition is not met do
4  |   choose s' ∈ N(s) randomly;
5  |   if f(s') ≤ f(s) or exp(−Δ(s', s)/T) ≥ U[0, 1] then
6  |   |   s = s';
7  |   end
8  |   update T parameter;
9  end
10 return s̄ = s;
```

namely the *cooling scheme* that governs the amount of lowering, the *temperature length* or *epoch length* that counts the number of moves that are evaluated using the same value of T, and the *temperature restart scheme* that controls when and how the temperature should be reset to a higher value. In this thesis we will focus on SA both as a standalone SLS and as a generic template from which several other metaheuristics can be derived, such as Threshold Acceptance, Great Deluge Acceptance, Late Acceptance Hill Climbing or the Record-to-Record Travel algorithm. The general template of SA is given in Algorithm 2.4. We postpone a more detailed discussion to Chapter 3.

Tabu Search. Tabu Search (TS) uses instead a history-based approach to diversification, by forbidding certain solutions or moves for some time. More precisely, TS in its basic formulation builds on top of an iterative improvement method such as BI or FI, keeping a list (the *tabu list*) of the last k solutions visited or moves performed; the parameter k is called *tabu tenure*. This mechanism favours diversification by preventing the search to return to the same set of good but likely suboptimal solutions. Rather than forbidding solutions, one more efficient possibility is to forbid single moves. This usually forbids a much higher number of solutions than a tabu list as previously defined would do, so some balancing mechanisms have been introduced: the *aspiration criterion* defines when the tabu list can be ignored, for example when a tabu solution improves over the incumbent one.

Like the temperature in PII and SA, the value of the tabu tenure is crucial for TS: values "too high" will restrict the admissible (non-tabu) solutions

too much and make the evaluation very expensive, while choosing a value too low will void make the tabu list ineffective. TS is one of the most popular metaheuristics, and several improvements have been proposed.

Dynamic Local Search and Guided Local Search. Dynamic Local Search (DLS) introduces the distintion between the objective function f of the problem, and the way it measures the quality of the solutions. More precisely, DLS makes use of an *evaluation function* $g(s) = f(s) + h(s)$, where $h(s)$ represents a *penalty* on the components of the locally optimal solutions, introduced to push the underlying search algorithm (e.g. a BI or FI) towards other solutions, rather than remaining stuck in the local optimum. The penalties $h(s)$ can also decrease, to avoid excluding optimal or very good solutions that share some of the components with the previously encountered local optima.

One kind of DLS is called Guided Local Search (GLS). It penalizes specific solution components with the maximum *utility*, defined as

$$\text{util}(s, i) = \frac{f_i(s)}{1 + \text{penalty}(i)} \tag{2.10}$$

where i is a solution component, $f_i(s)$ the contribution of i to the objective function value, and $\text{penalty}(i)$ the penalty associated to i.

2.2.4.2 Hybrid stochastic local search algorithms

Simple SLS algorithms are often easy to implement and quick to run, and can obtain extremely good results on certain problems. However, we can also implement more complex methods. In the category of hybrid SLSs we include algorithms obtained by extending and combining simpler metaheuristics.

Restarting a search from different randomly generated solutions can also be considered as the combination of a sufficiently large random step with a properly converging local search. While random restarts usually improve the outcome of the search, they also "reset" the knowledge about good solutions and solution components that is implicitly generated by iterative improvement methods. Thus, by controlling this diversification step we can hope to obtain a better tradeoff between diversification and permanence of quality components in the solution. This is the idea of Iterated Local Search.

Iterated Local Search. Iterated Local Search (ILS) [78, 79] is a method that repeats three steps until a given termination condition: (i) a diversifica-

tion step, called *perturbation*, (ii) a subsidiary local search, and (iii) an acceptance criterion, that determines whether the search should proceed from the newly found locally optimal solution or from the previous one. The purpose of the perturbation is to escape local optima and diversify the search, but without moving too far away from what are anyway good quality solutions. What differentiates ILS from other methods is the fact that its two separate parts control the two phases of the search, respectively intensification and diversification, which can therefore be chosen and tuned separately. This contrasts with the methods seen so far, which for the most part employ one single mechanism that needs to maintain a delicate equilibrium between the two phases.

ILS is a very generic framework, that can be implemented using several different components. The perturbation directly affects the solution value, and is therefore often problem-dependent. Many works in the literature proposing an ILS often use a basic local search such as a FI or a BI, or a simple one like simulated annealing or tabu search, but it can also be an arbitrarily complex one [80, 81]. Many ILS use a greedy algorithm as local search, and the resulting algorithms is also called Iterated Greedy [82].

The main drawback of ILS is the fact that a single perturbation configuration is most likely not suitable for both good and bad areas of the search space. To overcome this issue and make ILS more adaptive to the landscape, a more recent variant of ILS called Breakout Local search makes use of a VND to control the perturbation strength [83]. It follows the earlier Breakout diversification mechanism, whose idea is to find the minimal effort needed to escape a local optimum [84].

Variable Neighbourhood Search. Variable Neighbourhood Search (VNS) algorithms employ different neighbourhood structures, one at a time. The search starts with one neighbourhood, and when it remains stuck in a local optimum it continues with a different one; whenever a new best solution is found, the algorithms returns to its first neighbourhood. Once the last neighbourhood structure fails to improve the incumbent solution, the algorithm terminates. It should be noted that in general the local optimum for one neighbourhood N_i may not be a local optimum for another neighbourhood N_j. Ideally, the neighbourhoods should be used in order from the one yielding the strongest intensification to the one allowing the maximum diversification. VNS can be considered a generalization of the VND local search previously discussed.

Greedy Randomized Adaptive Search Procedure. GRASP is based on the idea of applying randomization to a greedy construction procedure [85, 86]. Greedy algorithms build solutions in a short time by making at each step a locally optimal decision. For many problems, this procedure produces a poor final solution. GRASP restricts the space for each construction move by considering only a restricted candidate list (RCL), whose goal is to introduce diversification in an otherwise purely intensificative procedure. The solution produced with this restricted greedy procedure is most likely not a locally optimal one, so an additional local search can be subsequently applied. These two steps are repeated until a given termination condition is reached.

2.2.4.3 Population-based metaheuristics

An important class of metaheuristics makes use of more than one solution. These algorithms are called *population-based*, and employ various strategies to harness a set of candidate solutions to find the best possible one. This class can be divided in two main groups, Evolutionary Algorithms and Swarm Intelligence methods. Evolutionary Algorithms are inspired by the phenomenon of natural evolution. They make use of a population of solutions that are used, in each iteration, to generate a set new solutions (*individuals*) by combination and mutation mechanisms, from which a subset of solutions is chosen according to some selection scheme. Swarm Intelligence methods are inspired by natural occurrencies of cooperation of agents to achieve a common goal, that would be impossible for a single agent by itself. The cooperation is distributed, meaning that there is no central authority that gives orders or instructions, but the agents need some mechanism of communication or coordination. For example, social insects such as ants or bees looking for food sources are able to converge to the shortest route to the source; flocks of bird take an aerodynamic shape when they fly, and fish schools move together in a way that resembles a single, larger fish, to deter potential predators. Such cooperation has suggested researchers alternative ways of making use of a population in metaheuristics and the first swarm intelligence algorithm proposed was *Ant Colony Optimization*.

Evolutionary algorithms. Evolutionary Algorithms (EAs) are a family of methods that draw inspiration from the evolutionary process of a population of individuals [87]. Starting from a set of initial solutions, EAs aim to replace its elements with elements of higher quality. They iterate three main steps: (i) *selection*, a function that chooses the population elements from which the new ones will be generated, (ii) *mutation* applies some modifica-

tion to each chosed solution individually, and (iii) *recombination* generates new solutions by combining the selected individuals. The population for the subsequent iteration is then composed with some criterion from the former and the new solutions. The underlying idea is that solutions of good quality include good components, and by merging them together we can eventually obtain solutions that contain more high quality components than the starting ones.

EAs include several subfamilies, among which here we mention the most important ones. Genetic Algorithms use a population of strings or vectors, called *genes*, and are particularly suited for discrete optimization [88]. Evolution strategies are instead used to tackle continuous problems, and the solutions are represented as real-valued vectors [89, 90, 91, 92]. Differential evolution also uses vectors of real values as solutions, but the update is performed by considering the difference between values in the respective positions in the parent solutions [93, 94].

One important limitation of many EAs is their scarce intensification potential. To compensate, a local search can be performed on every newly generated solution. Algorithms combining EAs with a local search are knows as *memetic algorithms* [95, 96, 97].

Other than being powerful general-purpose templates, EAs can also be improved with problem-specific knowledge. For example, the EAX genetic algorithm makes use of a crossover operator based on edge assembly, and is one of the state-of-the-art heuristics for the TSP [98].

Ant Colony Optimization. Ant Colony Optimization (ACO) uses a population of agents or artificial ants [99, 100, 101, 102]. These agents individually construct candidate solutions in a probabilistic way, converging iteration after iteration towards the solution components that belong to the higher quality solutions found. This convergence is achieved thanks to the *artificial pheromone*, which is information that is added to the solution components to indicate their "attractiveness" for the agents, and thus increase or decrease the probability for each component to be part of the final solution. The artificial pheromone mimics the pheromone left by ants during their foraging activity.

An ACO algorithm is composed of three main steps. In the first one, each agent constructs a solution, taking into account the cost of each choice and the pheromone. In the second one, additional operations may be performed, such as running a local search on the solutions found by the agents. This step is not necesarily present. The third main step is the pheromone update, which increases the pheromone for components of good solutions,

thus increasing the likelihood of their choice in the following iterations, and reduces it for components of poor quality solutions. These three main steps are iterated until a termination condition is met.

ACO is a family of algorithms with plenty of variants introduced over the basic template descibed above, called Ant System. Among them we mention Ant Colony System, which introduces two modifications [103]. The first one is the pseudo-proportional decision rule in the solution construction, which strengthens intensification by choosing with a certain probability the best option available. To compensate this greedy rule, each agent performs a local pheromone update, in addition to the global one of the original formulation. Other notable variants are Max-Min Ant System [104, 105], that bounds the pheromone levels, and Elitist Ant System, which considers only the best solutions in the pheromone update. While ACO was proposed and is used mainly for combinatorial problems, variants for continuous problems also exist [106].

Particle Swarm Optimization. Particle Swarm Optimization (PSO) is instead an algorithm originally proposed for continuous optimization, inspired by the flock of birds [107]. Opposite to ACO, where each agent constructs a solutions, in PSO each agent (particle) is a solution, normally in the form of a vector of real values. The agents are connected to each other in a given *topology*, that determines with which other agents each particle exchanges information. Starting from an initial set of solutions, generated at random or computed in some other way, the algorithm iteratively updates the value of each solution until convergence. The difference of each solution value from the previous iteration is called *velocity*. The velocity of each solution is updated taking into account three components: (i) the current value of the velocity, weighted by a factor called *inertia* that preserves the direction of the agent, (ii) the *personal best* of the solution, that is, the best solution value encountered by the solution in the past, and (iii) the *global best*, the best value registered in the past by any solution that is connected to the agent by the topology. Personal and global best are weighted by scaling factors called *acceleration coefficients*, and by random diagonal matrices. Variants of PSO can be generated by defining alternative topologies and update rules.

2.2.4.4 Matheuristics

Matheuristics is the name given by the hybridization of SLS algorithms with exact methods, in particular, methods based on mathematical programming. They are sometimes also called (Very) Large Scale Neighbourhood Search,

because the exact method is employed as a search in a large, albeit bounded, area of the search space, defined by the incumbent solution.

In this family we can find a broad range of algorithms, often designed for binary MIPs. Here we limit ourselves to some notable examples. The simplest one is probably *variable fixing*, or *diving*, that consists in fixing the value of a certain number of variables in a given initial solution, and solve to optimality the remaining subproblem.

Another approach, called *Local Branching* [108], adds a constraint to the original IP model whose purpose is to bound the Hamming distance between the incumbent solution and the locally optimal solution to be found. This is effectively a MIP formulation of the k-opt neighbourhood previously described: starting from an initial solution, the algorithm iteratively tries to move through the best solutions in the neighbourhoods defined by the Hamming distance. The similar *Proximity Search* [109] is a more flexible version of Local Branching. Starting from a given initial solution $\tilde{x}$, it modifies the original MIP model in two parts: it adds a penalty term to the objective function value that penalizes the solutions according to the Hamming distance from x, and it introduces a *cutoff* constraint $f(x) \leq f(\tilde{x}) + \theta$ for some $\theta > 0$. When properly tuned, the effect is a local search algorithm that looks for neighbouring solutions that improve over the incumbent, preferring "close-by" solutions when they yield limited improvement, but allowing longer jumps towards more distant solutions if a great improvement is observed.

The *Relaxation Induced Neighbourhood Search* (RINS) algorithm [110] starts from a feasible solution x of a MIP problem and the solution $\tilde{x}$ of the linear relaxation of the problem; the variables in $\tilde{x}$ matching the value in x are fixed, and the algorithm continues solving the remaining supbroblem.

Introduced to obtained initial, high quality feasible solutions, the Feasibility Pump [111, 112, 113, 114] also starts from the solution $\tilde{x}$ of the linear relaxation of the problem with the goal of turning it into an integer solution x, but in this case x can be infeasible. Iteratively, in the basic formulation x is obtained by rounding the variables of $\tilde{x}$ to the nearest integer, and thanks to an additional constraint to the model, whose purpose is to minimize the Hamming distance between the LP solution and the rounded solution x.

The *polishing* algorithm [115] is instead a genetic algorithm targeted at improving the solutions found during the search in a MIP solver.

2.3 From automatic algorithm configuration to automatic algorithm design

In practical applications, a good algorithmic idea is not enough to solve a problem efficiently. We need to understand whether such idea is suitable for the problem under consideration and for the specific instances we need to solve. We possibly need to choose a different algorithm, we need an efficient implementation and we need to adapt it to the problem and problem instance. All these issues fall under the field of *algorithm engineering*.

In this section we review the main areas of algorithm engineering that are relevant for this thesis. First, the task of choosing the most suitable algorithm for a problem and problem instance when several ones are possible is the *Algorithm Selection problem*. The task of adapting an algorithm to a problem and problem instance by choosing the right value for its parameters is the *Algorithm Configuration problem*. The task of building an algorithm for a problem or problem instance is the *Algorithm Design problem*. We will, in particular, focus on automatic approaches for all these problems that can outperform human practitioners. Here we introduce these problems, along with some approaches used in practice; the relationship between them is discussed in more detail in Chapter 5.

2.3.1 Algorithm selection

Given a set (called *portfolio*) of algorithms $\mathcal{A}$ available to solve a problem P (possibly all the algorithms that can defined for P), and a set of instances $\mathcal{I}$ of P (possibly all the instances that can defined for P), the Algorithm Selection (AS) Problem [116] is defined as the problem of assigning each instance $\mathcal{I}$ to the algorithm in $\mathcal{A}$ that gives the best results according to some performance measure [117, 118]. Figure 2.3 shows the Algorithm Selection process as depicted in the original work [116]; the AS problem requires to find the mapping $S(x)$.

Of course, for any practical application the mapping $S(x)$ should be feasible to compute and should generalize to unseen instances. The naive approach of testing all the algorithms on all the instances is clearly not suitable for any sufficiently large set of instances and/or set of algorithms due to computational infeasibility and will fail to generalize to new instances. The AS problem is thus defined as a learning problem. For each instance we compute a set of *features*, measures for some characteristics of the instances that are deemed relevant when choosing the best algorithm. For a set of training instances we relate the performance of the algorithms on the instance features available, possibly identifying the subset of features that

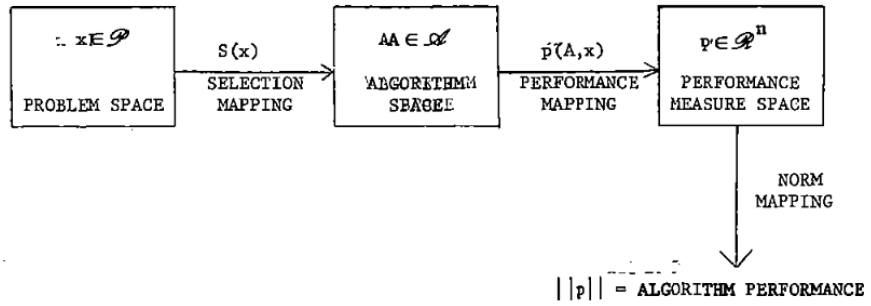

FIGURE 2.3: The algorithm selection process (image taken from the original paper [116]).

correlate the most with the performance. For new instances, it will be then possible to compute their features and map them onto the best algorithm(s) identified in the training phase.

Features can be of two kinds, *static* and based on *probing*. Static features are computed simply by observing the instance, for example, its size. For combinatorial optimization problems, it is very common to use problem-specific features, such as cluster distance for the TSP or the ratio between clauses and variables in SAT. Probing features instead include measures obtained by running some algorithm to "test" the instance, such as executing a local search to observe how many moves are necessary to converge to a locally optimal solution [119, 120]. For continuous optimization problems, instead, where the objective function *is* the landscape, features can be computed by exploring the landscape [121, 122, 123, 124].

2.3.2 Algorithm configuration

In practice, the actual performance of many algorithms relies on a proper setting of their parameters. The behaviour of a simulated annealing is determined by a certain sequence of temperature values, usually defined by providing an initial value and a function to update it, whose coefficients are of course parameters of the algorithm; a tabu search needs the tabu tenure, the length of the list containing the forbidden solutions or moves; for an ant colony optimization algorithm we have to choose the number of ants and the α and β parameters. Through parameters we can also define alternative choices, such as enabling or disabling a certain function. For example using the aspiration criterion in a tabu search, or to select one among several options such as the linear programming solver in a MIP package. While in

this thesis we focus on optimization algorithms, the same applies also for several other domains, from Machine Learning to compilers and databases [23, 24, 125, 126].

2.3.2.1 Parameter definition

Parameters are defined by some attributes like the name, the type, and the set of possible values; in practice, we might need to include also the way they are handled by the algorithm (e.g. the flag in the command line sequence), as this may differ from the name.

Parameters can be classified in several different ways, such as continuous or discrete, representing choices or values, strings or numbers. From a functional perspective we can partition the parameters of an algorithm in four classes:

- *categorical* (discrete) parameters, representing alternative choices. Such choices do not have any particular relation between them. Some examples are the choice of the LP solver in commercial MIP packages (simplex, dual simplex, etc), or which local search to use in an ILS. The set of possible values is defined as an unordered set $a \in A = [a_1, a_2, a_3, \ldots, a_k]$, where k is the cardinality of A, and the values $a_1 \ldots a_k$ are not necessarily ordered. In practice, they might be represented as strings or integers;

- *ordinal* (discrete) parameters, representing alternative choices that have a certain relation between them. For example, the strength of the perturbation in an ILS could be defined as `small`, `medium` or `large`; MIP solvers let the user choose whether the policy for cut generation should be aggressive, moderate, or not employed at all. The set of possible values is defined as an ordered set $a \in A = [a_1, a_2, a_3, \ldots, a_k]$, with k being the cardinality of A, and the values $a_1 \ldots a_k$ are listed following the ordering relation between them;

- *integer numerical* parameters, such as the number of ants in an ACO, or the tabu tenure in a TS. While formally integer parameters are discrete, they are often better treated as the discretization of a continuous (sub)set. The parameter range is defined as $a \in [a_l, a_u]$, where a_l and a_u are, respectively, the upper and lower bound of the parameter values;

- *real-valued numerical* (continuous) parameters, as the probability p of choosing a random move in a RII, or the initial temperature value in

a SA. Also in this case, the range is given in terms of its upper and lower bounds $a \in [a_l, a_u]$.[3] In practice, also real-valued parameters necessarily can only take a finite set of values, and a given precision has to be provided;

Parameters can depend on each other. If parameter a is used, and therefore has to take a certain value, only if parameter b takes a certain value b_k or a value in an certain subset $B_s \subset B_t$ of the complete allowed range B_t, then we say that a is *conditional* on b. For example, if the local search in an ILS is a tabu search, then we need to choose the tabu tenure; if the local search is instead a PII, then we will choose its probability w_p without having to consider the tabu tenure.

As our context is always the comparison of algorithmic performance, we need to pay attention to those parameters that define the environment and the metrics used to evaluate the algorithms. Thus, we cannot consider as parameters the integrality gap in a MIP solver or the maximum runtime or number of moves evaluated in a local search. These parameters can be changed only when the environment of the evaluation changes, for example when we want to benchmark one or more algorithms over different runtimes. Analoguously, the random seed, that is usually implemented as a parameter, has to be considered part of the input to the algorithm, like the instance.

2.3.2.2 The algorithm configuration problem

The *algorithm configuration problem* requires to find an assignment of values for the parameters of an algorithm such that the algorithm obtains the best results over the instances under consideration. The performance can be measured in different ways, according to the scope of the algorithm. When looking for an optimal solution, the best performance can mean the shortest time to converge on an instance, or the highest probability to converge in a given time. Over a set of instances, it can be the minimum total time, the average time, or the highest number of instances solved to optimality in a given time. For a heuristic algorithm, typical performance measures are the final solution quality obtained in a given runtime or a given number of solutions evaluated, or the cumulative statistics over a set of instances.

What underlies all the possibilities listed above is the optimization process of a certain metric in a fixed environment. This process is also called *parameter tuning* in the optimization community. In Machine Learning, a parameter is usually a coefficient of the model to be found or optimized, hence the configuration of the algorithm used to find those coefficients is

[3] The bounds can also be excluded from the range, e.g. $a \in (a_l, a_u)$.

called *hyperparameter tuning*; we note however that there is no conceptual difference in the two distinct levels of optimization.

In the following description, we assume without loss of generality a minimization problem, where the goal is either to minimize the solution quality of an heuristic algorithm for a minimization problem, or to minimize the time to find the optimal solution for an exact algorithm; a maximization setting is equivalent, and will be presented when necessary. Here we also limit ourselves to the optimization of a single performance measure; the evaluation of a configuration under multiple objectives can be easily derived from the single-objective case, and will be introduced later in this thesis.

Formally, a *configuration* $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\} \in \Theta$ is an assignment of values for each one of the parameters θ_i of a *target algorithm* $\mathcal{A}$ with n parameters, in the set of all the possible assignments Θ for $\mathcal{A}$. As this task is an instantiation of the generic optimization task defined in Section 2.1.1, we will also call θ a *candidate configuration*, and Θ the set of candidate configurations, or *parameter space*. The problem requires to find a configuration $\theta^* \in \Theta$ such that a performance measure $\mathcal{C}(\theta, \mathcal{I}) \to \mathbb{R}$ for a set of instances $\mathcal{I}$ is minimized:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{C}(\theta, \mathcal{I}). \tag{2.11}$$

This is, usually, a stochastic black-box process defined over a mixed set of variables. It is *black-box* since $\mathcal{C}(\theta, \mathcal{I})$ is usually the final measure (runtime or solution quality) observed at the end of the search of the algorithm $\mathcal{A}$ over $\mathcal{I}$ when using configuration θ. There may be *stochasticity* in the algorithm $\mathcal{A}$ (e.g. a stochastic local search, or an exact algorithm with a capped runtime), in the metric $\mathcal{C}$ (e.g. an imprecise way of measuring the runtime on a certain machine), or in the set of instances $\mathcal{I}$, that can be drawn from a certain distribution, in particular, for stochastic optimization problems. The variables involved (the parameters) can be either discrete or continuous.

The configuration space grows extremely quickly with the number of parameters n, as $\Theta \subseteq \theta_1 \times \theta_2 \times \cdots \times \theta_n$ (not necessarily all the combinations of values are valid configurations).[4] Combined with the usually expensive and stochastic black-box evaluation of the configurations, this makes the

[4] While $|\Theta|$ is theoretically infinite whenever there is at least one real-valued parameter, or an integer parameter with an unbounded range, in practice the limited precision available on machine implementations makes it a finite value. When several categorical parameters are involved, we might prefer to not include in any practical estimate of $|\Theta|$ the numerical parameters conditional on them, as they are normally fine-tuned versions of structurally different alternatives. In practice, however, counting the exact number of valid configurations might not be an easy task when categorical parameters or invalid configurations can be defined.

quest for the best configuration a computationally infeasible task for any moderately-sized configuration process.

While manual trial-and-error is often the immediate choice for this configuration task, any practitioner soon realizes how even a deep knowledge of the target algorithm is not enough to obtain the best results. Exact methods are extremely naive, and alternative, heuristic approaches to find sufficiently good configurations $\hat{\theta}$ are therefore required (see Section 2.3.2.4). The use of algorithms for the parameter tuning task is called the *Automatic Algorithm Configuration* approach.

2.3.2.3 Offline versus online tuning

We distinguish two different contexts for algorithm configuration. The first one, called *offline tuning*, is a learning process that aims to discover the best configuration to be used at the beginning of the algorithm execution for a certain set of instances, keeping the values fixed. Being a learning process, we need to partition the set of instances $\mathcal{I}$ into a *traning set*, over which we will look for a good $\hat{\theta}$ whose final quality will be evaluated on the instances assigned to the *test set*. To achieve good results on the test set, the distribution of the instances in the training set should be as close as possible to the distribution of the test instances. In the remainder of this thesis we focus only on offline tuning.

The alternative context requires instead to optimize the set of parameters during the search, possibly finding different parameter settings for each different instance. This approach is called *online tuning* or *parameter control*. An example is the reactive tabu search, that learns the length of the tabu list during the search [127, 128]. Online tuning methods require anyway an offline tuning of their parameters and possibly also for an initial configuration.

2.3.2.4 Tuning approaches

In this section we review some of the methods that are or have been used to tackle the Algorithm Configuration problem.

Traditional methods. We already mentioned that the manual trial-and-error process is often the first choice for parameter tuning. This is clearly suboptimal, because the huge configuration space cannot be efficiently explored by human operators. Practitioners can test a limited set of alternatives, following existing conventions and past experience, introducing bias in the process which results in poor configurations.

One way to introduce objectivity in the process is to partition the parameter space in regions of equal area by dividing each parameter range into a fixed number of intervals, and considering the boundary value between intervals. This method can be visualized as placing configurations on a fixed grid, and is therefore called grid search. Grid search does not scale well with the number of parameters. One possible alternative is to randomly sample the configurations, in order to observe a higher number of different values for each parameter [129]. Another possibility is to follow a *design of experiments* methodology such as Latin Square Hypercube or the Taguchi method to select candidate configurations that cover as much as possible the entire search space [130, 131]. Also design of experiment methods, however, do not scale well with the number of parameters: the growth of the configurations required is in fact exponential in the number of parameters. They also are of limited help in case of stochastic evaluations, as their application requires multiple evaluations of the same configuration to be considered reliable.

Continuous optimization methods. Many practical applications require the optimization of a continuous space, so plenty of methods are available to configure algorithms that only expose real-valued parameters. In this case, the problem becomes a continuous optimization problem. Hence, several continuous optimization methods have been used for configuring algorithms. Examples include entropy-based methods [132], derivative-free methods [133], racing methods [134], kernel density estimation [135], and local search algorithms [136].

Model-based optimization. These methods aim to build a surrogate model of the response of the algorithm to the different parameter configurations. They iteratively repeat three main steps: (i) the evaluation of the configurations on an instance, (ii) the fit of a model of the response, (iii) which is then used to simulate the results obtained by a set new configurations, among which the most promising ones are selected for the next effective evaluation. SPOT relies on several statistical models, from linear models to regression trees and random forests to Gaussian Processes [137]. The most notable representative of these methods is SMAC (Sequential Model-based Algorithm Configuration), which uses random forests to build the surrogate model [138]. Random forests have been integrated also into configurators belonging to other categories [139, 140].

Metaheuristic algorithms. The Automatic Configuration Problem is an optimization problem over a mixed set of variables, so it can be tackled by any suitable optimization algorithm, including metaheuristics. ParamILS is an iterated local search that operates on a discrete space [141, 142]. It starts from a set of configurations which include the default one, and alternates an improvement phase, based on the one exchange neighbourhood, and a perturbation phase, which alters k parameters at random. Numerical parameters need to be discretized to be handled by ParamILS.

GGA implements instead a gender-based genetic algorithm [143]. A configuration is represented as a tree, assigned a gender, and has an age. Crossover is performed by merging the trees of two parent configurations. Each new configuration is then mutated, and configurations with an age over a certain threshold are eliminated. EVOCA is another evolutionary algorithm proposed to configure algorithms [144]. OPAL extends the direct search of the continuous optimizer MADS [133] to consider also discrete parameters [145].

Racing methods. The last family of methods we present here is based on the observation that the most expensive part of a configuration task is the evaluation of candidate configurations. Thus, in order to make a good use of the scarce computational budget available, these methods aim to quickly identify poor performing configurations to discard them and focus the search on a more precise assessment of the good configurations. The first algorithm to adopt this strategy in the context of algorithm configuration was the racing algorithm [146]. It starts from a set of randomly sampled configurations, including the default one if provided, and benchmarks them on a subset of instances. After having evaluated the configurations on a certain number of instances, a statistical test removes from the pool of candidate configurations the ones that performed significantly worse than the best one. The default test employed is the Friedman analysis of variance by ranks (F-race). The process then continues only with the surviving configurations, iterating evaluation and statistical-based elimination. This process, called *race*, terminates when the budget for the evaluations is exhausted, or when a minimum number of configurations remain.

This method is good in filtering bad configurations and selecting a robust winner among the starting ones, but is restricted by the set of candidate configurations initially sampled. The best configuration found by a race is anyway a representative of a possible good area of the parameter space, that can be better explored. Thus, the configurations that survive at the end of a race are used as seeds to sample new ones around them. A new race is

then launched using the new configurations. This process is called *iterated race*, and is the one implemented in the irace package [147, 148]. Since version 2.0 of irace, the best configuration of the last iteration, called *elites*, are included in the new race, and cannot be eliminated until the new configurations demonstrate to outperform them on the same set of instances seen. Race after race, the new configurations are sampled increasingly closer to the elites from which they stem. This cycle of sampling and race is iterated until termination, normally for the exhaustion of the total experimental budget.

This class of algorithms can be mathematically modeled using the multi-armed bandit framework [149, 150]. Hyperband builds over the SuccessiveHalving algorithm of [149], essentially implementing a single race with a very aggressive elimination policy [151]. Hyperband, in fact, starts discarding configurations after the first evaluation, keeping a fraction $(1/k)^i$ of the initial number after k^i rounds of evaluations, for a given parameter k and $i \in \mathbb{Z}$.

2.3.3 Automated algorithm design

The definition of the Algorithm Configuration Problem of Section 2.3.2 is very generic and allows for a broad interpretation of the term "parameter". In addition to numerical values to wiggle such as thresholds or number of agents, we can consider every decision that the algorithm designer has to make when implementing an algorithm for a certain problem as one option among a set of possible ones, that is, a categorical parameter. We can therefore refrain from committing to a definitive choice in the design phase and exposing all the possible options as high level parameters. An automatic tool can then be employed to make the best choices for our particular case.

This extension of what is considered a parameter leads naturally to an application of the algorithm configuration problem to the task of designing an algorithm, which we call Algorithm Design Problem (ADP). The set of tools we can employ to automate the solution of this task define what will be called in the remainder of this work the Automated Algorithm Design approach (AAD) [152].

This application of the algorithm configuration problem to the design of algorithms is part of the Programming by Optimization (PbO) paradigm. A graphical representation of the PbO paradigm is given in Figure 2.4. It relies on a *design space* composed by the set of options available to the algorithm designer and the set of rules on how to assemble an algorithm. We then translate this design space into a parameter space, that can be explored by an automatic configurator.

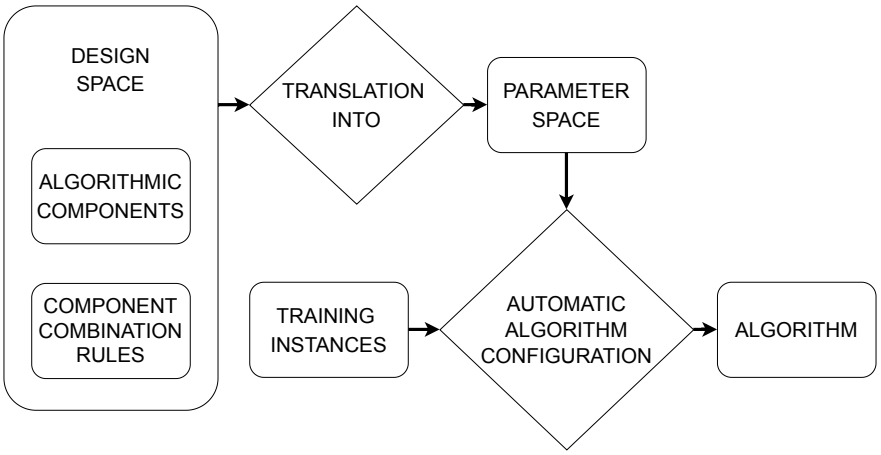

FIGURE 2.4: The workflow of the automated algorithm design approach following the Programming by Optimization paradigm.

2.3.3.1 Elements of programming by optimization

Having already treated in detail the various automatic configuration approaches available, we analyze the other elements of PbO we used in this thesis.

A component-based view of SLS algorithms Many if not all the algorithms we reviewed in Sections 2.2.3 and 2.2.4 can be divided into components. It becomes therefore possible to consider a SLS algorithm as a combination of basic building blocks that follows a certain structure. From this perspective, designing a SLS algorithm means therefore to select a certain such combination of blocks, following the desired structure. For example, every SLS needs to implement a mechanism to decide whether the solution currently evaluated needs to be accepted or discarded, or to determine how and when to terminate the search. All the possible options for these mechanisms can be grouped as, respectively, the acceptance criterion and the termination condition.

Once defined what components are part of an algorithm, we can then collect all the options available for each component into an *algorithmic framework*. The components are considered categorical parameters, and a component can have one or more associated (usually numerical) parameters, that are dependent on the specific options they refer to. This framework acts as a high level, extremely flexible algorithm, from which we instantiate the

desired combination by selecting the relative components and parameters. The way we combine these components determines what the resulting SLS will look like. Different SLSs can make use of the same components, but combining them in a different way.

The framework we use in the research here reported is the EMILI framework [153, 154]. It was originally designed for PFSP problems, but it is in principle open to SLS in general, that is, any problem and any SLS algorithm could be implemented in it. This component-based view of algorithms has anyway be applied to several other algorithms, such as ACO [155, 156, 157, 158], PSO [159, 160], multi-objective evolutionary algorithms [161, 162, 163] and differential evolution [164]. Several other frameworks, however, implement algorithms starting from basic components, such as EasyLocal++, ParadisEO or JMetal [165, 166, 167, 168, 169]. MIP solvers such as SCIP, CPLEX, Gurobi can be considered frameworks that implement several options for the various step that compose branch-and-cut algorithms [64, 65, 66].

Rules for combining components There are two main ways of combining components. In the first one, the *top-down approach*, we define the structure of the algorithm in advance, and let the configurator make the best choice for each part. This approach is the simplest one. Generating algorithms of a certain kind is particularly suited for comparisons, and this is the reason why we will follow it in the rest of the thesis. For the same reason, this approach has been used for example in [157, 160, 162].

The alternative approach is the *bottom-up* one, which does not assume an algorithmic template fixed in advance. The resulting algorithms can therefore be of any possible shape of arbitrary complexity, provided a careful definition of the various possibilities to combine the components. One possibility is to use context-free *grammars* to outline the possible combinations of components and parameters [170, 171]. An example, adapted from Figure 2.2 of [172], is given in Figure 2.5, where we see that an ILS is defined as a particular sequence of components. This sequence includes the local search component, which can in turn be defined as one among four (in this example) alternatives, including another ILS. We see therefore the potential of grammar-based approaches: it is possible to define an algorithm recursively, thus allowing an enormous flexibility. To prevent algorithms to grow indefinitely, it is therefore necessary to introduce a limit on the recursion potential [81].

The grammar is translated into a parameter file, that allows to use automatic configuration tools to build a fully fledged algorithm [171].

```
<LocalSearch> ::= <FirstImprov> | <BestImprov> | <ILS>
<FirstImprov> ::= 'first' <InitialSolution> <Termination> <Neighbourhood>
<BestImprov>  ::= 'best' <InitialSolution> <Termination> <Neighbourhood>
<ILS>         ::= 'ils' <LocalSearch> <Termination> <Perturbation> <Acceptance>
```

FIGURE 2.5: Example of a grammar that defines the rules to build a SLS algorithm (adapted from [172]).

For practical reasons, we use a grammar to define the structure of the algorithms in all the experiments described in this thesis.

2.3.3.2 Alternative approaches

Outsourcing the generation of code from a human developer to a machine is a task for which other techniques have been applied. Genetic programming applies evolutionary algorithms to solutions that represent programs [173]. Solutions are usually represented as trees. Grammatical evolution is instead a related approach that uses evolutionary algorithms to evolve programs in the form of a grammar [174]. Hyperheuristics employ a set of low-level heuristics to assemble or generate higher level heuristic algorithms [175, 176].

2.4 Discussion

In the previous sections we have reviewed many different stochastic local search algorithms, with a particular attention to the most popular single solution ones. This review was written from the perspective of the diversification mechanism. The list of algorithms is not exhaustive, and its subdivision is not the only possible classification of SLS and metaheuristic algorithms. Several taxonomies have in fact been proposed to categorize and classify optimization algorithms. A thorough comparison between these taxonomies is given in [177], listing 18 prior works aimed at categorizing or classifying algorithms, and discussing in detail why so many criteria exist. The task is further complicated by the fact that considering whether two algorithms are effectively the same method is, ultimately, a matter of subjective opinion. There is in fact no clear-cut criterion to decide whether two algorithms are variations of the same method, or two different algorithms. While in some cases it is clear that an algorithm is a close varianty of an older one [178, 179], of the two works that independently proposed the threshold ac-

ceptance Moscato and Fontanari considered it a simulated annealing variant [180], while Dueck and Scheuer called it a new algorithm [181]. This is, in my opinion, the problem at the root of the proliferation of metaphor-based metaheuristics [182, 183]: there is no objective criterion to discriminate between small variations and legitimately new algorithms. Keeping this proliferation under control and discriminating the novel contributions from mere redressings of existing methods is a task left to single researchers and editors. Fortunately, some venues implemented policies to oppose this phenomenon [184, 185, 186]

The perspective I followed in my work is to group algorithms by their structure and by the mechanism employed to control the exploration/exploitation tradeoff. Admittedly, a SLS can employ more than one of these mechanisms, so it may be a criterion that does not universally apply. However, the criteria fit sufficiently well the scope of this work, which is to analyze a simple SLS algorithm to understand their behaviour. Thus, for the rest of this work, we will consider two methods as belonging to the same family if we can identify a common structure, and the differences between them do not yield a radically different behaviour. In the rest of this work, the term "algorithm" in the context of SLSs will therefore be used to indicate both a broad family of instantiations that share a common template and contain many variants, and a particular instantiation of that template. The distinction will be clear from the context.

This categorization criterion again provides only a fuzzy definition, but it has two main advantages: it allows us to bypass the subjectivity in the taxonomy, and it applies not only to any existing SLS, but to any possible one. These advantages become evident when applying automatic methods to select, configure and design SLS algorithms.

As we will see in the next chapter, several SLS algorithms that appear in the literature under different names can be related to the general simulated annealing structure, of which they can be considered variants. To emphasize the characterization of algorithms by their search behaviour and highlight the contribution of each variant, we prefer presenting and discussing them in algorithmic terms whenever possible. For historical reasons and ease of discussion, it is however impossible to completely forgo the original metaphors.

Revisiting simulated annealing: A component-based analysis 3

Simulated Annealing (SA) is one of the oldest metaheuristics and has been adapted to solve many combinatorial optimization problems. Over the years, many authors have proposed both general and problem-specific improvements and variants of SA. We propose to accumulate this knowledge into automatically configurable, algorithmic frameworks so that for new applications that wealth of alternative algorithmic components is directly available for the algorithm designer without further manual intervention. To do so, we describe SA as an ensemble of algorithmic components, and describe SA variants from the literature within these components. We show the advantages of our proposal by (i) implementing existing algorithmic components of variants of SA, (ii) studying SA algorithms proposed in the literature, (iii) improving SA performance by automatically designing new state-of-the-art SA implementations and (iv) studying the role and impact of the algorithmic components based on experimental data. We experimentally demonstrate the potential of this approach on three common combinatorial optimization problems, the quadratic assignment problem and two variants of the permutation flow shop problem.

3.1 Introduction

Metaheuristics are a method of choice when dealing with computationally hard problems from a wide range of application areas [12, 187]. They can be

described as problem-independent general rules to follow to derive effective heuristic optimization algorithms. The field of metaheuristics has a long and often successful history that can be traced back to the 1960s and 1970s with the first proposals of evolutionary computation techniques [88, 90, 91, 188] or ideas related to search intensification and diversification [189] that later led to tabu search or scatter search. Despite the successes, a critical review of the history of the field given in [14] argues that "It is not an exaggeration to claim that the field of (meta)heuristics [...] has yet to reach a mature state". One reason can be summarized as a focus on *competition* rather than on *knowledge*, with unfortunate side-effects such as the proliferation of dubiously novel methods, generally based on natural metaphors [182] and "high" or "promising" performance claims sometimes backed with poor scientific practices (see e.g. [178, 190]) and lacking insights on how and why such methods work.

The main objective of this work is to propose an alternative way of addressing such issues. Instead of proposing yet another metaheuristic, we aim at exploiting the enormous body of knowledge available in specific metaheuristics and identifying the basic ideas that are available for the design of new variants of the known metaheuristics. For this purpose, we see a metaheuristic algorithm not as a monolithic procedure that is proposed as one block, but as being composed of a set of algorithmic components for each of which a number of different alternative instantiations exist. In other words, we take a component-based view of metaheuristics and we collect many options available in the literature into an *algorithmic framework*, classifying them according to their purpose for the metaheuristic under concern. From this point of view, algorithm design turns into the task of choosing the right set of basic component from the framework.

To make these ideas concrete, we build a framework for Simulated Annealing (SA), which is one of the oldest and most studied metaheuristics. In fact, at the time of writing this article, the Scopus bibliographic database indexes more than 6 000 articles with the keyword "Simulated Annealing" in the title, a number that increases to 30 000 if we expand the search to abstracts and keywords. SA also has shown to result in high-performing heuristics for many problems [191, 192]. Over the years, authors have proposed many variants of SA for different problems, offering by now a large number of implementation choices to an algorithm designer who would like to use SA. We build this framework by taking a component-wise perspective on the design of SA algorithms. We extract the algorithmic components (including alternative algorithm options and numerical parameters) form proposed variants of SA algorithms and classify them according to their use, to offer alternative choices for each main class of components.

From the point of view of algorithm configuration, each component can be seen as a categorical parameter, whose values are the various options provided in the framework; each of these components may have associated additional numerical parameters. By choosing the right options and the respective numerical parameters, one can re-instantiate existing algorithms; by choosing different options, instead, it is possible to build new variants. This point of view allows us to exploit the recent developments in automatic algorithm configuration techniques, which, given a set of training instances of the problem to be solved, search without manual algorithm designer interaction for the best parameter settings using computer experiments [193]. This task of automated algorithm configuration is supported by recent tools such as ParamILS [142], SMAC [138], or irace [148]. In this perspective, our work follows other proposals for the generation of automatically configurable algorithm frameworks such as Satenstein, a framework for local search algorithms for the satisfiability problem in propositional logic [194, 195], frameworks for multi-objective ACO algorithms [157], ACO algorithms for continuous optimization [196], or multi-objective evolutionary algorithms [162].

This automated process offers at least four advantages. First, it avoids the often applied manual trial-and-error process, which is time-consuming and biased by the personal experience of the algorithm developer. Second, in the framework typically many more algorithm components are made available than even an experienced developer of metaheuristic algorithms may be aware of due to the vast literature. Third, it allows to configure algorithms for a specific computational environment or application context in a transparent (and reproducible) way. Fourth, the data generated during the algorithm configuration process may be further analyzed and so insight into the importance of specific algorithm components can be obtained from these data. We experimentally show the advantages of our approach studying SA algorithms for three well-known combinatorial optimization problems, the quadratic assignment problem (QAP) and the permutation flowshop scheduling problem (PFSP) under the makespan and total completion time objectives, respectively.

This chapter is structured as follows. In the next section we review SA, and, in Section 3.3, we describe its component-based formulation and the set of components we have implemented. Section 3.4 describes the methodology and the experimental setup for the experiments reported in Sections 3.5 and 3.6. Additional analysis is given in Section 3.7 and we conclude in Section 3.8. Supplementary material for this work is available at [197].

3.2 Simulated annealing

In a nutshell, SA is a stochastic local search algorithm that, starting from some initial solution, iteratively explores the neighbourhood of the current solution. It always accepts improving solutions and worsening solutions probabilistically in dependence of the amount of deterioration and a parameter called temperature. SA is inspired by the work of Metropolis *et al.* [198], who proposed a Monte Carlo integration for solving equations of state of physical systems composed of particles in statistical mechanics. At high temperatures, the particles are rather free to move, and the structure is subject to substantial changes. The temperature decreses over time, and so does the probability for a particle to move, until the system reaches a state of lowest energy, its ground state. Kirkpatrick and co-authors [199], and independently Černy [200], turned these ideas into a heuristic method for tackling combinatorial optimization problems. The physical temperature is translated into a "temperature" parameter, the state of the physical system corresponds to a candidate solution, the ground state corresponds to the globally optimal solution, and a change of state corresponds to a move to a neighbouring candidate solution.

Let us first introduce the formal notation used in the remainder of this work. Let $s \in S$ be a candidate solution in the set S of all possible candidate solutions and $f : S \to \mathbb{R}$ be the objective function; thus, $f(s)$ is the objective function value of candidate solution s. An optimal solution s^* is a candidate solution for which holds $f(s^*) \leq f(s) \quad \forall s \in S$. With $\mathcal{N}(s)$ we denote the neighbourhood of s. $\Delta(s', s)$ is the objective function difference of two candidate solutions s and s'; we will also refer with $\Delta_{i,j}$ to the difference $f(s_i) - f(s_j)$ of objective function values of two candidate solutions in two different instants i and j for brevity. We denote the temperature as T; T_0 and T_f are, respectively, the initial and final temperature, while T_i is the temperature at a generic instant i. Without loss of generality, we assume the objective function to be minimized.

The distinguishing characteristic of SA at its inception was the possibility of probabilistically accepting worsening moves. The most commonly used acceptance criterion is the so-called Metropolis condition [198, 199], which always accepts a neighboring candidate solution if it is better or equal to the current one; a worse neighboring candidate solution is accepted with a probability of $\exp(-\Delta(s', s)/T)$. Hence, a worsening solution is accepted with a probability that depends on both the amount of worsening and T. With an equal worsening of the objective function value, a solution is more likely to be accepted when the temperature is high (that is, typically in the beginning of the search), while when the temperature is low (typi-

cally towards the end the search), improving candidate solutions are prioritized. The probabilistic acceptance of worsening moves makes SA able to reach the globally best solution when certain conditions are met. Several authors have studied these conditions, especially focusing on the cooling scheme, the function that controls the temperature iteration after iteration [191, 201, 202, 203]. Unfortunately, these analyses usually prove the convergence to the global optimum in time tending to infinity making the implications in practice less clear. As in this work we focus on SA from an empirical perspective, we do not delve into theoretical analyses but refer the reader for such to [191, 192] and cited works therein.

There are several reasons that make SA ideal for the approach outlined in Section 3.1. Deriving from a simple idea, in its original formulation it is also a simple algorithm, making it possible to clearly identify its components and their scope. It does not require complex operations, so its behaviour is easy to understand. The role and the impact of the numerical parameters is well understood: the temperature, transitioning from its initial value to its final one, controls the transition from an initial exploratory behaviour to a final exploitative one; if its values are too high, the algorithm will fail to converge towards good solutions, but for too low values it will be likely trapped in suboptimal regions, missing the chance to escape. At the same time, the task of making the right design choices and choosing the right values of numerical parameters is often tedious and error-prone; hence, in practice it is difficult to find the best setup.

In the literature there are several algorithms that can be related to SA. For example, keeping the same temperature value throughout the whole execution turns SA into an algorithm known under several names, such as Metropolis algorithm [204], generalized hill climbing [205], static simulated annealing [206], or simply fixed temperature schemes [207, 208]. Replacing the probabilistic acceptance criterion with a deterministic one, it is possible to generate a new class of local search algorithms, such as the threshold acceptance [180, 181], great deluge algorithm and record-to-record travel [209], or the more recent late acceptance hill climbing [210, 211]. All these variants are described in the next section. A discussion about the similarities and differences with other metaheuristics can be found in [191, 212]. Outside the optimization field, SA is also akin to the Markov chain Monte Carlo (MCMC) method that is extremely popular in several fields such as machine learning, statistics, physics, or economics [213].

Our analysis is limited to SA as a stand-alone search algorithm. Therefore, we do not consider the use of SA as a component within other hybrid algorithms such as the local search in a memetic algorithms. Also, technology-driven improvements such as parallelization techniques or GPU-

based implementations are beyond the scope of this article.

3.3 Component-based formulation of simulated annealing

For our purposes, we divide SA into nine different components, seven of which are algorithm-specific and two are problem-specific. These components define, respectively, how an SA algorithm can be specialized to tackle a specific problem. The two problem-specific components are the construction of an Initial Solution, and the generation of a new candidate solution in the Neighbourhood. While the choice of these two components has an important impact on algorithm performance [212], we delay any discussion about them to the following sections, where the specific problems are introduced and tackled.

We give a generic outline of an SA algorithm in Algorithm 3.1. The seven components that in our framework define an SA algorithm are:

- the choice of the Initial Temperature (line 3 of Algorithm 3.1);
- the Stopping Criterion, which determines when the execution is finished (line 4);
- the Exploration Criterion, which chooses a solution in the Neighbourhood (line 5);
- the Acceptance Criterion, which determines whether the new solution replaces the incumbent one (line 6);
- the Temperature Length, which indicates whether the temperature is updated (line 12);
- the Cooling Scheme, which updates the temperature (line 13);
- the Temperature Restart, the component responsible for resetting the temperature to its original or another high value (line 15).

While these seven components are the ones particular for SA, they may also be based on problem-specific settings, such as the initial temperature adopted in [214]. We will describe these problem-dependent algorithmic components only if they are used in our experiments.

An SA algorithm starts by taking as input the Initial Solution, the Neighbourhood, a problem instance π and the control parameters. It proceeds by initializing its internal status, in particular setting a value for the

Algorithm 3.1: Component-based formulation of SA. The components we have identified for our analysis are written in Smallcaps.

Input: a problem instance π, a Neighbourhood $\mathcal{N}$ for the solutions, an Initial Solution s_0, control parameters
Output: the best solution s^* found during the search

```
1  best solution s* := incumbent solution ŝ := s_0;
2  i := 0;
3  T_0 := initialize temperature according to Initial Temperature;
4  while Stopping Criterion is not met do
5      choose a solution s_{i+1} in the Neighbourhood of ŝ according
         to Exploration Criterion;
6      if s_{i+1} meets Acceptance Criterion then
7          ŝ := s_{i+1};
8          if ŝ improves over s* then
9              s* := ŝ;
10         end
11     end
12     if Temperature Length is met then
13         update temperature according to Cooling Scheme;
14     end
15     reset temperature according to Temperature Restart
         scheme;
16     i := i + 1;
17 end
18 return s*;
```

Initial Temperature. Starting from the initial solution, SA iteratively selects one candidate solution in the Neighbourhood according to the Exploration Criterion. The new candidate solution is evaluated against the incumbent candidate solution using the Acceptance Criterion; if it also improves over the best solution found so far (global-best), it becomes the new global-best candidate solution. The Temperature Length component determines whether the temperature parameter has to be updated; if yes, the Cooling Scheme sets the temperature to its new value. To favour a new phase of exploration, the Temperature Restart scheme controls whether the temperature should be reset to a higher value. At each iteration, the Stopping Criterion is checked–if it is met, the algorithm terminates returning the best candidate solution found. We next describe the options

for the seven algorithm-specific components we identified in the literature and which we make available in our framework.

3.3.1 Initial Temperature (line 3)

This component sets an initial value for the temperature parameter. The methods available may take into account some problem instance related information or not. The instance-based schemes can be based either on syntactical information, or on a limited exploration of the search space, typically by a random walk, from which some statistics are computed. Some of the following schemes also propose a final temperature value related to the initial one, but these are rather proposals than mandatory rules. A variation that we apply here is to include a multiplicative scaling user-defined constant k to make the methods more flexible.

Here and in the following we enumerate the available options we implemented for ease of later reference. Options for initial temperature are refered to as **ITx**, where **x** is a number. Other references are defined analogously in the text.

Fixed value. The simplest option **IT1** is to choose a fixed initial temperature $T_0 = k$. Another option **IT2** is to set an initial temperature proportional to the objective function value of the initial candidate solution $T_0 = k \times f(s_0)$ as in [215].

Random walk-based methods. Other criteria perform a random walk in the search space creating a sequence of candidate solutions $s_0, s_1, s_2, \ldots, s_l$, where l is the length of the random walk. The resulting objective function values $f(s_0)$, $f(s_1)$, $f(s_2)$, ..., $f(s_l)$ are treated as a time series and the temperature is set as a statistic of the time series. One simple option (**IT3**) is to take a value proportional to the maximum gap between two consecutive candidate solutions as initial temperature, and the minimum non-zero gap as final temperature, as for example in [216]:

$$T_0 = k \times \max_{1 \leq i \leq N} |\Delta_{i,i+1}| \tag{3.1}$$

$$T_f = k \times \min_{1 \leq i \leq N, >0} |\Delta_{i,i+1}|. \tag{3.2}$$

As the maximum value of a set can be unrepresentative or highly skewed, we also include (**IT4**) the possibility of choosing a value proportional to the

average of the absolute gaps between candidate solutions in the random walk

$$T_0 = k/N \times \sum_{i=1}^{N} |\Delta_{i,i+1}| \tag{3.3}$$

or a more elaborated scheme, as for example in [217] (**IT5**):

$$T_f = k \times \min_{1 \le i \le N, >0} min\Delta_{i,i+1} \tag{3.4}$$

$$T_0 = T_f + k \times (\max_{1 \le i \le N} \Delta_{i,i+1} - T_f)/10, \tag{3.5}$$

thus relating the initial temperature value to its supposed final one.

As the initial temperature is used to control the initial acceptance probability of worsening moves, the initial temperature can be determined such that it yields a desired initial acceptance probability, as done in [218, 219, 220]. This component (**IT6**) performs a random walk in the search space and computes the value $T_0 = |(k \times \Delta_{avg})/\log p_0|$, which gives an initial probability p_0, where Δ_{avg} is the average gap between two solutions in the random walk and k is a scaling coefficient. Equation **IT6** is derived from the Metropolis acceptance condition.

Misevicius [221] proposed an initial temperature scheme (**IT7**) for QAP, again based on a random walk in the search space. It extends **IT5** by taking into account also the average gap in the random walk. **IT7** is defined as

$$T_0 = k \times ((1 - \lambda_1 - \lambda_1')\Delta_{min} + \lambda_1\Delta_{avg} + \lambda_1'\Delta_{max}) \tag{3.6}$$

$$T_f = k \times ((1 - \lambda_2 - \lambda_2')\Delta_{min} + \lambda_2\Delta_{avg} + \lambda_2'\Delta_{max}), \tag{3.7}$$

where the real-valued weights $\lambda_1, \lambda_1', \lambda_2, \lambda_2' \in [0, 1]$ are chosen to satisfy the conditions $\lambda_1 + \lambda_1' \le 1$, $\lambda_2 + \lambda_2' \le 1$. By choosing $\lambda_1 = 0$, $\lambda_1' = 0.1$, $\lambda_2 = 0$, $\lambda_2' = 0$ we obtain **IT5**. Misevicius also uses a simplified version (**IT8**) of his scheme, by setting $\lambda_1' = \lambda_2' = 0$. By varying the λ_1 and λ_2 values, the behaviour of the search will differ, resulting in faster or slower cooling.

Problem-dependent schemes. While some of the previous schemes were introduced for some specific problems, they are general. In [214], the authors propose an initial temperature for an SA algorithm for the Permutation Flowshop Scheduling Problem (PFSP), which uses specific features of a problem instance. We consider this scheme as the PFSP is one of the problems we use in this work. The scheme **IT9** is defined as

$$T_0 = k \times \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}/(m \times n), \tag{3.8}$$

where n is the number of jobs, m is the number of machines, p_{ij} is the processing time of job i on machine j and parameter k is set to $1/5$ in the original work.

3.3.2 Stopping Criterion (line 4)

The stopping criterion controls the termination of the SA algorithm. The schemes can be based either on some predetermined value, the actual outcome of the search, or considerations when continuing the search is deemed too expensive or unlikely find further improvements.

Fixed termination. One possible choice for termination (**SC1**) is a fixed maximum amount of time [215, 220]. Another possible choice (**SC2**) is a fixed number of candidate moves [217]. The search can be terminated also when a certain minimum temperature value has been reached (**SC3**) [214]; such value can be either determined by the chosen Initial Temperature scheme, in case it computes also a final value, or given as input by the user. Other possible criteria include having a maximum number of cooling steps (**SC4**) [222], or a maximum number of temperature restarts (**SC5**) [216].

Adaptive termination. To provide more flexibility, criteria based on the observation of the actual algorithm execution have been implemented. One such criterion (**SC6**) is to stop the execution after a fixed number of candidate moves that did not result in accepted solutions. **SC7** terminates the execution as soon as the total acceptance rate falls below a certain threshold; another one (**SC8**) stops the search when the acceptance rate for the last k candidate moves is below a given threshold [218, 219]. Finally, criterion **SC9** stops the algorithm when none of the last k candidate moves found a new best solution. The number of candidate moves to be considered by the adaptive criteria can be expressed either in terms of an absolute value or proportional to the neighborhood size.

Note that except for the termination criterion **SC1**, with the other termination criteria the computation time used by an SA algorithm is not determined a priori but depends on the search progress, making the running time an independent variable.

3.3.3 Exploration Criterion (line 5)

The role of the Exploration Criterion is to choose one candidate solution to evaluate from the neighbourhood $\mathcal{N}(s)$. The first SA algorithm [199] explores the neighborhood randomly, that is, it generates and evaluates a

randomly generated neighbour at every step (**NE1**). This kind of default behaviour of SA is used in the vast majority of SA implementations.

Connolly [217] claims this approach to be inefficient, because for lower temperatures (that is, lower acceptance probabilities, see the next component) potential improvements might be missed, and at the same time it might be difficult to escape local optima. He proposes to compute and evaluate the neighbours in some sequential order (**NE2**), which guarantees at least to identify a solution as a local optimum in case no worsening move is accepted after scanning the full neighbourhood.

Ishibuchi et al. [223] propose two other schemes to select a solution in the neighbourhood. In the first one (**NE3**), k solutions are randomly generated in the neighbourhood, and the best of the k is then compared with the current incumbent. This scheme is modified to (**NE4**), which follows NE3 but stops the process of generating neighboring candidate solutions as soon as one candidate solution is found that improves on the current incumbent, which is then immediately accepted.

3.3.4 Acceptance Criterion (line 6)

This is the component that determines whether the solution $s' \in \mathcal{N}(s)$ generated by the Exploration Criterion is accepted. The traditional Metropolis criterion [198, 199] is the best known example for this component. Almost all criteria described here follow one simple pattern: always accept improving or same quality solutions and occasionally accept worsening solutions depending on a specific criterion. The acceptance of worsening moves allows to move the search away from the current area of the search space being explored, in the hope of finding regions with better solutions. It is, however, crucial to find a good balance between search intensification and diversification, which is managed by setting appropriately the temperature parameter.

Metropolis-based criteria. The Metropolis condition (**AC1**) [198] is the criterion proposed in the original SA formulation [199]. It always accepts moves to improving and same quality solutions, and accepts worsening moves with a probability that depends on the increase $\Delta(s', s)$ of the objective function value and the temperature T:

$$p_{Metropolis} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\Delta(s', s)/T\right) & \text{otherwise.} \end{cases} \tag{3.9}$$

As the exponential function is relatively expensive from a computational point of view, Johnson *et al.* **AC2** [218] proposed to pre-compute the expo-

nentials for a sequence of values in the interval where $\Delta(s', s)/T$ results in probabilities between 1 and ~ 0.0067. Worsening moves that entail probabilities lower than this latter value are immediately discarded. The actual values during the search are mapped to the closest values in this array. In their experiments, they estimate a saving of $1/3$ on the total runtime.

Chen and Hsieh [224] proposed a bounded version **AC3** of the Metropolis criterion, that rejects a move whose solution quality is worse with respect to the incumbent by a given parameter ϕ_{BM}:

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\Delta(s', s)/T\right) & \text{if} \quad f(s) < f(s') \leq f(s) \times \phi_{BM} \\ 0 & \text{if} \quad f(s') > f(s) \times \phi_{BM}. \end{cases} \tag{3.10}$$

Another acceptance criterion was proposed in [225] (**AC4**) as part of the Generalized Simulated Annealing (GSA) variant. It includes (a power of) the cost of the currently accepted solution in the probability calculation. The formula proposed is

$$p_{GSA} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\beta f(s')^g \Delta(s', s)\right) & \text{otherwise,} \end{cases} \tag{3.11}$$

where β and g are control parameters. GSA makes no explicit use of temperature, but following their notation the original SA employs $\beta = 1/T$, $g = 0$.

Geometric criterion A criterion proposed in [226] (**AC5**) always accepts an improving solution, and accepts a worsening solution with a probability that decreases in a geometric way[1]:

$$p^k_{Geom} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ p_0 \times r^{k-1} & \text{otherwise,} \end{cases} \tag{3.12}$$

where p_0 is the initial acceptance probability, $r < 1$ is the reducing factor, and k is the number of update steps. This criterion, thus, does not use the temperature in the evaluation of a solution. In fact, it is rather related to randomized iterative improvement as defined in [12].

[1] We describe this component as proposed in the original paper, even though this formulation combines the acceptance criterion with the Cooling Scheme.

Deterministic criteria. While the probabilistic acceptance of worsening moves is the distinctive feature of SA, some authors have questioned whether the stochasticity introduced by the acceptance criterion is necessary to obtain good results [95, 180, 181, 227]. In these works, the authors have proposed a deterministic version of the Metropolis criterion, called Threshold Accepting (TA, **AC6**) by [181], which accepts every worsening solution whose difference in objective function value is lower than a threshold $\bar{\phi}$:

$$p_{TA} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq \bar{\phi} \\ 0 & \text{otherwise.} \end{cases} \tag{3.13}$$

$\bar{\phi}$ is a parameter whose value decreases during the search process, just as the temperature in SA. In [181] the authors do not provide any guidance in how to initialize and update $\bar{\phi}$, while the authors of [180] explicitly use SA terminology such as "temperature" and "cooling".

Starting from Threshold Accepting, Dueck proposed two alternative deterministic acceptance criteria under two different metaphors [209]. The first one (**AC7**, originally called Great Deluge Algorithm – GDA) accepts every move leading to a solution with objective function value $f(s)$ less than a threshold $\bar{\phi}^k$, therefore moving away from the idea of comparison between solutions:

$$p^k_{GDA} = \begin{cases} 1 & \text{if} \quad f(s) \leq \bar{\phi}^k \\ 0 & \text{otherwise,} \end{cases} \tag{3.14}$$

with $\bar{\phi}^{k+1} = \bar{\phi}^k - \lambda$, where λ is a fixed parameter.

The second criterion (**AC8**, in the original work, Record–to–Record Travel – RTR) accepts only solutions whose value is not larger than that of the best solution found so far plus a threshold γ (note that, differently, TA compares the new solution with the current one, which might already not be the best one found):

$$p_{RTR} = \begin{cases} 1 & \text{if} \quad \phi \leq f(s^*) + \gamma \\ 0 & \text{otherwise,} \end{cases} \tag{3.15}$$

where γ is a fixed parameter.

A more recent work by Burke and Bykov [210, 211] proposes another deterministic criterion called Late Acceptance Hill Climbing (LAHC, **AC9**). The idea of LAHC is to use the history of the search, by comparing the candidate solution also with an incumbent solution of the past. LAHC therefore can accept worsening moves, but it cannot accept a solution whose objective function value is not at least as good as the one of another solution

that was already accepted. The acceptance of a solution s is therefore controlled according to the formula

$$p^l_{LAHC} = \begin{cases} 1 & \text{if} \quad f(s_l) \leq \max\{f(s_{l-1}), f(s_{l-\kappa})\} \\ 0 & \text{otherwise,} \end{cases} \tag{3.16}$$

where $f(s_l)$, $f(s_{l-1})$ and $f(s_{l-\kappa})$ are, respectively, the cost of the solution at move l, $l-1$, $l-\kappa$, for a fixed κ, which is a parameter of LAHC.

The baseline for comparisons, and simplest deterministic acceptance criterion fitting in the algorithmic outline given in Algorithm 3.1 is to accept only improving or same quality solutions (**AC10**), discarding worsening ones:

$$p_{det} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

The effect of this choice is turning SA into a Hill-Climbing search [228], with its drawback of quickly getting stuck in local optima.

3.3.5 Cooling Scheme (line 13)

The cooling scheme is the component that governs the temperature updates, that is, it computes the temperature value T_{i+1} at instant $i+1$ as a function of the previous value T_i at instant i. The default desired behaviour in SA is a monotonic decrease of the temperature, making the acceptance of worsening solutions more and more unlikely. Since the inception of SA, cooling schemes have been the most studied components, both from a theoretical and an experimental point of view. Many schemes that we review adhere to this decreasing behaviour; the option of raising the temperature is governed by the Temperature Restart component. The possibility of having multiple proposed moves evaluated at the same temperature is defined by the Temperature Length scheme. In the literature there are, however, some non-decreasing schemes, that we also consider in this study.

Geometric schemes. In the original SA paper [199], the authors propose two decreasing geometric schemes, $T_{i+1} = \alpha \times \beta^{T_i}$ (**CS1**) and $T_{i+1} = \alpha \times T_i$ (**CS2**), where $0 < \alpha, \beta < 1$ are constant control parameters. Typically, these parameters are set to high values (e.g. ≥ 0.9), implying a slow decrease of the temperature.

Logarithmic schemes In [229], the authors use a logarithmic cooling scheme (**CS3**) $T_{i+1} = a/\log(b+i)$ (**CS3**) with $b = 1$ in the original work. We also implement the scheme $T_{i+1} = a/(b + \log i)$ (**CS4**) [230].

Lundy-Mees and variants A popular cooling scheme is the one proposed by Lundy and Mees in [201] (**CS5**), in which $T_{i+1} = T_i/(a + b \times T_i)$. a and b have to be chosen as $a + b \times T_i > T_i$ to guarantee final convergence.

Connolly in [217] develops a variant of the Lundy-Mees scheme called *Q8-7* (for: seventh variant of the eighth scheme tested) (**CS6**) for the QAP. It is a two-step scheme, that initially decreases the temperature (using the Lundy-Mees formula **CS5**), until too many consecutive candidate moves m are discarded. Then, the next move is accepted, the cooling is stopped and the temperature is set to the value at which the best solution was found. Connolly sets parameters as $a = 1$ and b, m in dependence of the initial and final temperature and the size of the neighbourhood. Another scheme (**CS7**) is proposed in [231] and uses $T_{i+1} = a/(1 + b \times T_i)$ with $a = b = 1$.

Quadratic schemes. In 1993 Andersen, Vidal and Iversen [232] developed a quadratic cooling scheme (**CS8**) for a network design problem. The formula proposed is

$$T_{i+1} = a \times K^2 + b \times K + c, \text{ where} \tag{3.18}$$

$$a = \frac{T_0 - T_f}{I^2}, \;\; b = 2 \times \frac{T_f - T_0}{I}, \;\; c = T_0,$$

and K is the current iteration, T_0 the initial temperature, $T_f = 0$ the final temperature and I the maximum number of iterations.

Arithmetic scheme. Another simple cooling scheme proposed in [209] uses an arithmetic decrease of the temperature value (in this case, of the threshold) $T_{i+1} = T_i - a$ for some fixed value a. The implementation of this criterion (**CS9**) requires a more careful control about the update, as the temperature value cannot drop below zero. We therefore choose to implement the criterion according to $T_{i+1} = \max\{T_i - a, 0\}$.

Non-decreasing schemes. Since the inception of SA, various authors have studed variants that keep the same temperature values throughout the whole search [203, 204, 206, 207, 208, 227, 233, 234] under different perspectives: SA variants, theoretical studies about convergence behaviours, different algorithmic paradigms; and different names, such as Metropolis algorithm, Static SA, Generalized Hill Climbing, Probabilistic Iterative Improvement or simply fixed-temperature SA. All these works essentially (re)propose and study the scheme $T_{i+1} = T_i = T_0 \;\forall i$ (**CS10**). Here, T_0 is a supposedly *optimal* temperature value that can obtain superior results with respect to cooling schemes that reduce the temperature parameter. The *Q8-7* cooling

scheme **CS6** of [217] may also be seen as a scheme where a fixed temperature value is discovered by the algorithm at runtime.

As non-decreasing schemes are dependent on good initial settings, a more robust scheme (**CS11**) sets a temperature band $[T_0, a \times T_0]$, $a > 1$, and, at each update, randomly chooses a value from it.

Another scheme is the old bachelor acceptance (OBA) [235]. It was originally proposed as a variation of threshold acceptance, which is discussed as a special case; it lowers the temperature if a solution is accepted, and raises it if the candidate solution is discarded. While OBA is conceived to be used with **AC6**, it can be paired with any acceptance criterion, and we describe it here as such, considering the two variants presented in the original paper. OBA1 (**CS12**) adjusts the temperature "symmetrically" according to the formula

$$T_{i+1} = \begin{cases} T_i + ((\text{age}/a)^b - 1) \times \Delta \times (1 - i/M)^c & \text{if} \quad s_i \text{ is discarded} \\ T_i - ((\text{age}/a)^b - 1) \times \Delta \times (1 - i/M)^c & \text{if} \quad s_i \text{ is accepted,} \end{cases} \tag{3.19}$$

where a, b, c are control parameters, M is the total number of candidate moves, Δ is the granularity of the update and *age* is the number of consecutively rejected moves. OBA2 (**CS13**) instead uses a "steepest descent, mildest ascent" strategy [236] that makes acceptance of a solution more likely in the first d proposed moves after an accepted move:

$$T_{i+1} = \begin{cases} T_i + (\Delta/d) \times (1 - i/M) & \text{if} \quad s_i \text{ is discarded} \\ T_i - \text{count} \times \Delta \times (1 - i/M) & \text{if} \quad s_i \text{ is accepted,} \end{cases} \tag{3.20}$$

where d and *count* are control parameters, which are updated by incrementing *count* by one if age is smaller than d; in the other case, age is set to 1.

3.3.6 Temperature Length (line 12)

This component controls the number of candidate moves L that are evaluated at a certain temperature.

Fixed temperature length. The options in this category include the following. The first is to update the temperature after a fixed number of candidate moves $L = k$ (**TL1**) with a value of $L = 1$ meaning that the temperature is updated after every move. The second is to update after a number of candidate moves proportional to the size of the neighbourhood $\mathcal{N}(s)$, $L = k \times |\mathcal{N}(s)|$ (**TL2**), or proportional to the square of neighbourhood size $L = k \times |\mathcal{N}(s)|^2$ (**TL3**) [222]. Also the size n of the problem instance

is used $L = k \times n$ (**TL4**) e.g. in [215], or its square $L = k \times n^2$ (**TL5**) [217].

Adaptive temperature length. Instead of fixed temperature lengths, it is possible to set these depending on the search progress. Abramson [237] updates the temperature after a certain number of accepted moves (**TL6**). In [222] the authors combine this approach with a maximum number of total candidate moves at a given temperature (that might be fixed or proportional to $|\mathcal{N}(s)|$), to avoid spending too many candidate moves at a certain temperature (**TL7**).

Variable temperature length. Other proposals [238, 239] update the temperature length according to some functions, to compensate the increased strictness in accepting worsening moves at low temperatures with an increased number of evaluations. We test arithmetic ($L_{i+1} = L_i + k$ **TL8**), geometric ($L_{i+1} = k \times L_i$ **TL9**), logarithmic ($L_{i+1} = k/L_i$ **TL10**) and exponential ($L_{i+1} = L_i^{1/\alpha}$ **TL11**) updates for temperature lengths, respectively, where k and $0 < \alpha < 1$ are fixed numerical parameters and T_i is the temperature at iteration i.

3.3.7 Temperature Restart (line 15)

As most cooling schemes decrease the temperature, it eventually happens that the acceptance of worsening solution is very rare, resulting in a Hill-Climbing type behavior. Therefore, authors have proposed to reset the temperature to the initial or another level once it reaches a critically low value [233], allowing in this way effective escapes from local optima. The reset of the temperature to its initial value is called *temperature restart*, while setting it to a possibly different value is called *reheating*. Of course, one may also choose to never reset the temperature value (**TR1**).

Restarting, fixed settings. The most immediate option is to reset the temperature value to the initial one, once some conditions are met. For example, once it reaches a minimum absolute value (**TR2**), when it reaches a certain percentage of its initial value (**TR3**), after a certain number of proposed moves (**TR4**, this number being fixed, proportional to the size (or its square) of the neighbourhood) or after a certain number of cooling steps (**TR5**).

Restarting, adaptive settings. In this category, options are to restart when the overall acceptance rate falls below a certain threshold (**TR6**), when the

acceptance rate of the last l candidate moves is below a certain threshold (**TR7**), or when the search is not progressing (no accepted moves in the last l iterations, **TR8**).

Reheating. Alternatively to restarting the temperature, it is also possible to set it to a higher value $T_{i+1} = T_i/k$, $0 < k < 1$. We can perform this operation once the acceptance rate falls below a given threshold, overall (**TR9**) or in the last k candidate moves (**TR10**), or the search has not accepted any new solution in the last k candidate moves (**TR11**). We also consider reheating after a certain number of candidate moves (**TR12**, again this number can be fixed or proportional to the size (or its square) of the neighbourhood) or cooling steps (**TR13**). A different version of reheat, called Enhanced Reheat (**TR14** [240]) performs a reheat according to the usual formula $T_{i+1} = T_i/k$, but at every reheating the parameter k gets reduced by a constant value ϵ. To prevent k to become negative, we actually update k using the formula $\max\{\epsilon, k - \epsilon\}$. The idea is to restart the search every time at a higher temperature, to increasingly push the algorithm towards an explorative behaviour.

Another option for reheating is to set the temperature value not to its original value or to another "generic" higher value, but to the temperature at which the best solution has been found, assuming that value being a good one in terms of acceptance probability. We can reset the temperature to this supposedly "optimal" value when the acceptance rate drops below a threshold (**TR15**) or when k consecutive candidate moves have been rejected (**TR16**).

3.4 Material and method

We have collected the algorithmic components described in Section 3.3 into an algorithmic framework, from which it is possible to instantiate, following the outline of Algorithm 3.1, a fully working algorithm. This outline also defines the ways the algorithmic components can be combined. The implementation itself is done in the EMILI framework [153], which aims at a flexible combination of algorithm components that makes it particularly amenable to automatic configuration.

In the following, we illustrate the possible uses of the framework at various levels of automatic configuration ranging, which we distinguish between three levels:

Level 1: instantiate known SA algorithms for algorithm comparisons (no configuration);

Level 2: tune known SA algorithms for fair comparisons;

Level 3: automatically configure the full SA framework.

These three levels correspond to different degrees of advancement in algorithm comparisons. In fact, in most current publications on metaheuristics, a new algorithm is compared to previously proposed ones using only published results. At Level 1, we instantiate all algorithms from a same code base and execute them in a same environment; such procedure already removes (noise) factors such as different implementation languages, implementation skills, and different computing environments. Level 2 advances over Level 1 by tuning the numerical parameters of each algorithm by automatic algorithm configuration techniques, reducing the side-effect of uneven tuning of the methods or using parameters fine-tuned for possibly other experimental conditions (e.g. short versus long computation times). Level 3 improves over Level 2 by a modern view on metaheuristic algorithm engineering as seeing these algorithms composed of different algorithm components [193, 241]. In fact, given the possibility of generating new, previously unseen SA algorithms potentially better performance may be reached. From an automatic configuration perspective, this is obtained by appropriate choices for the categorical parameters through automatic configuration. For the automatic configuration, we consider two setups. The first one configures the algorithm to reach the best solution quality within a given maximum computation time. While this is the most common setup when comparing metaheuristic algorithms, it has the disadvantage that minor changes in the available computation time or the computing environment (slower or faster machines) may have a major impact on algorithm performance. Therefore, in a second setup, we automatically configure SA algorithms for anytime behavior, which tries to obtain as good solutions as early as possible during the search [242]. To do so, we follow the methodology proposed in [243].

The data obtained during the automatic configuration process can be exploited to obtain insight into the importance of the various algorithm components and the numerical parameters. As a final step, we analyze the importance of the algorithm components from the data that are recorded during the configuration with irace by training a random forest model [244].

3.4.1 Experimental setup

As mentioned above, we implemented the SA components within the EMILI framework [153]. As the automatic algorithm configuration tool we use irace [147, 148], which is an R implementation of the Iterated Racing algorithm [245, 246]. irace begins with a set of uniformly sampled candidate

parameter configurations and tests them on a set of training instances. Configurations that are statistically worse get discarded during this process to save computational budget for the most promising candidates and to evaluate them on more instances. The best configurations are then used as seeds to sample new candidates, with a distribution skewed around the best performing ones. This process is iterated until the configuration budget is exhausted. The final configurations returned are the ones that performed best during the training phase. For the configurations to generalize to production environments or simply to an independent test set, it is responsibility of the user to provide a set of training instances that is representative for the desired use case. The random forest model is computed using the `ranger R` package [247].

For our experiments we consider two types of problems, the Quadratic Assignment Problem (QAP), and two objectives of the Permutation Flowshop Problem (PFSP), that we introduced in Section 2.1.3. We tune the numerical parameters 30 times, with a budget of 2000 experiments per tuning; the tuning of the whole framework is instead done 15 times, with a budget of 60000 experiments due to the much larger number of parameters. The tuning for anytime behaviour is performed three times. The possible values for the numerical parameters are reported in the Supplementary Material. The maximum time limit depends on the problem, and is specified in the following sections. The comparison between algorithms is always performed using common random seeds. All experiments have been run on a machine equipped with two Intel Xeon E5-2680 v3 CPUs running at 2.5GHz, with 16MB cache and 2.4 GB of RAM available per algorithm execution. Each algorithm execution is single-thread.

3.4.2 Quadratic assignment problem setup

The QAP is an assignment problem that models a variety of real world problems [51, 52]. Each QAP instance of size n has n *facilities* and n *locations*, and associated costs commonly referred to as *flow* between two facilities i and j, f_{ij}, and *distance* between two locations k and l, d_{kl}. An assignment of facilities to locations can be represented by a permutation π, where $\pi(i)$ gives the location to which facility i is assigned. The goal in the QAP is to find a permutation π that minimizes the objective function

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)}. \tag{3.21}$$

As per the problem-specific components, the initial solution is a permutation that is generated uniformly at random, while the neighbourhood is

the exchange neighbourhood $\mathcal{N}(\pi)$ defined as

$$\{\pi' \mid \pi'(j) = \pi(h) \wedge \pi'(h) = \pi(j) \wedge \forall l \notin \{j,h\} \; : \pi'(l) = \pi(l)\} \tag{3.22}$$

for a solution π. The size of this neighbourhood is $n(n-1)/2$. The objective function value of a solution s' can be computed from the objective function value of a neighbouring solution s in constant time. However, in practice, such advantage is beneficial only when the full neighbourhood is always explored, bringing down the computational complexity from a total of $O(n^3)$ to $O(n^2)$. It does not make too much difference when a single solution is evaluated in the neighbourhood, and the overall time needed to converge to a local optimum is comparable.

The QAP is a "difficult" NP-hard problem for which exact solutions can be found only for instances of small size: apart from few exceptions for very specially structured instances [49], instances of size $n = 40$ are often already out of reach for exact methods. We consider two sets of QAP instances. One is composed by instances whose data matrices are generated uniformly at random [215], and one where the matrices are generated randomly according to an Euclidean structure, closer to real-life instances [248]. When no ambiguity can arise we will refer to these two sets as random and structured instances, respectively. Each instance set is composed by 300 instances, equally divided in sizes 60, 80 and 100. Of each size, 50 instances are reserved for the training set in the configuration phase and 50 instances as the independent test set for the evaluation of the configured algorithms. The anytime behaviour is evaluated also on larger random and structured instances of size 500 from the same benchmark [215]. Unless otherwise specified, the runtime limit for QAP is 10 seconds; experiments with 30 and 100 seconds of runtime are reported in the supplementary material.

3.4.3 Permutation flowshop problem setup

The PFSP is a scheduling problem that arises in various industrial environments [53, 54, 55]. It is very well studied with many variants existing in the literature. We consider two of the most common variants, namely the PFSP under the *makespan* objective (PFSP-MS) [56] and under the *total completion time* objective (PFSP-TCT) [57, 58]. More formally, in the PFSP a set of n jobs have to be ordered for execution on a set of m machines, using the same execution order on all machines. Each job i takes p_{ij} units of time for processing on machine j. In the basic formulation of the PFSP, all jobs are ready for execution at time 0 and no concurrency or pre-emption is allowed.

A solution of the PFSP is a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of the n jobs. The PFSP-MS requires to minimize the makespan C_{max}, which is the completion time of the last job executed ($C_{max} = C_{\pi(n),m}$, where $C_{i,m}$ is the completion time of job i on the last machine m). The objective of the PFSP-TCT is to minimize the sum of the jobs' completion times, given by $\sum_{i=1}^{n} C_{\pi(i),m}$.

For both objectives, the initial solution is generated using the NEH heuristic [249]. Unless specified otherwise, the neighbourhood is the insert neighbourhood where a move (j,k) consists in selecting the element $\pi(j)$ in position j of the permutation π and inserting it in position $k \neq j$, resulting in a permutation $\pi' = [\pi(1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(k), \pi(j), \pi(k+1), \ldots, \pi(n)]$ if $j < k$ and $\pi' = [\pi(1), \ldots, \pi(k-1), \pi(j), \pi(k), \pi(k+1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(n)]$ if $j > k$. The size of the insert neighbourhood is $n(n-1)$, and each solution is evaluated in $O(n)$ time.

The training set is composed by 40 randomly generated instances with sizes between 50 jobs and 20 machines to 250 jobs and 50 machines [170]. The test set is the popular Taillard benchmark [250], consisting of 120 instances divided into 12 classes of 10 instances each, with sizes going from 20 jobs and 5 machines to 500 jobs and 20 machines. As additional benchmark for the anytime behaviour we use instances of size 800×60 from [251]. The runtime limit is based on the instance size and is computed as $(n \times m \times 0.015)/2$ seconds. Experiments with $10\times$ runtime are reported in the supplementary material.

3.5 Simulated annealing algorithms for the quadratic assignment problem

Attempts to solve the QAP with Simulated Annealing can be traced back at least to 1984 [216]. In the following years SA remained a popular choice for the QAP, and was often compared with Tabu Search, without any clear consensus in the scientific community about which method is the most effective [252, 253, 254, 255]. In what follows we list SA implementations proposed for the QAP or closely related problems.

The first two schemes we implement, BR1 and BR2, are proposed in 1984 by Burkard and Rendl [216]. We consider also the SA of Burkard and Rendl as described by Connolly in [217] for comparing the results of his experiments (CBR1 and CBR2). Connolly [217] proposes several versions of SA for QAP: two of them employ the cooling scheme of Lundy and Mees (CLM1 and CLM2), while the third version uses the Q8-7 scheme (Q87). Two other SA algorithms for the QAP were proposed by Jajodia

Table 3.1: Results of the Friedman rank sum test for the algorithms for the QAP in their default settings (top block, Level 1 first step), with ten seconds of runtime (middle, Level 1, second step) and after tuning their numerical parameters on the QAP instances, including the algorithms generated automatically (ALL, bottom, Level 2 and 3). Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Instances	Δ_R	Algorithm ranking
Random	10.92	**Jaj** (0), BIN (50), Q87 (251), CLM2 (453), CLM1 (496), BR1 (700), TAM (850), BR2 (1000), CBR1 (1150), CBR2 (1300)
Structured	21.3	**Bin** (0), JAJ (148), Q87 (391), CLM1 (486), BR1 (607) CLM2 (612), TAM (899), BR2 (1049), CBR1 (1199), CBR2 (1349)
Random	26.44	**Bin** (0), JAJ (48), CBR1 (166), TAM (286), CLM2 (641) BR1 (650), CLM1 (684), Q87 (925), BR2 (1101), CBR2 (1249)
Structured	17.82	**Bin** (0), JAJ (250), CBR1 (296), TAM (350), BR1 (599) CLM1 (801), CLM2 (847), Q87 (1049), BR2 (1216), CBR2 (1332)
Random	23.71	**All** (0), JAJ (164), CBR2 (323), TAM (417), BR2 (685), CBR1 (851), BR1 (858), BIN (902), Q87 (1230), CLM2 (1320), CLM1 (1500)
Structured	39.32	**All** (0), BIN (79), BR1 (271), TAM (547), JAJ (587), CBR1 (653), CBR2 (660), BR2 (867), Q87 (1209), CLM2 (1212), CLM1 (1428)

et al. [222] (JAJ) and Tam [220] (TAM) in 1992. The last SA for the QAP that we evaluate is by Hussin *et al.* [215] (BIN). A synopsis of how these implementations can be described in terms of their components is given in the supplementary material. Figure 3.1 gives the results on the test sets in terms of the Average Relative Percentage Deviation (ARPD) from the best known solutions, while Table 3.1 reports the ranking of the algorithms.

As a first step at Level 1 in our analysis, we instantiate these algorithms in our framework using the parameter settings proposed in the original papers. Algorithms BIN and JAJ obtain the lowest ARPD values (less than 2% ARPD on random instances, and less than 1% on the structured ones), and are significantly better than the other SA algorithms on structured and random instances when using default algorithm parameter settings. Maybe sur-

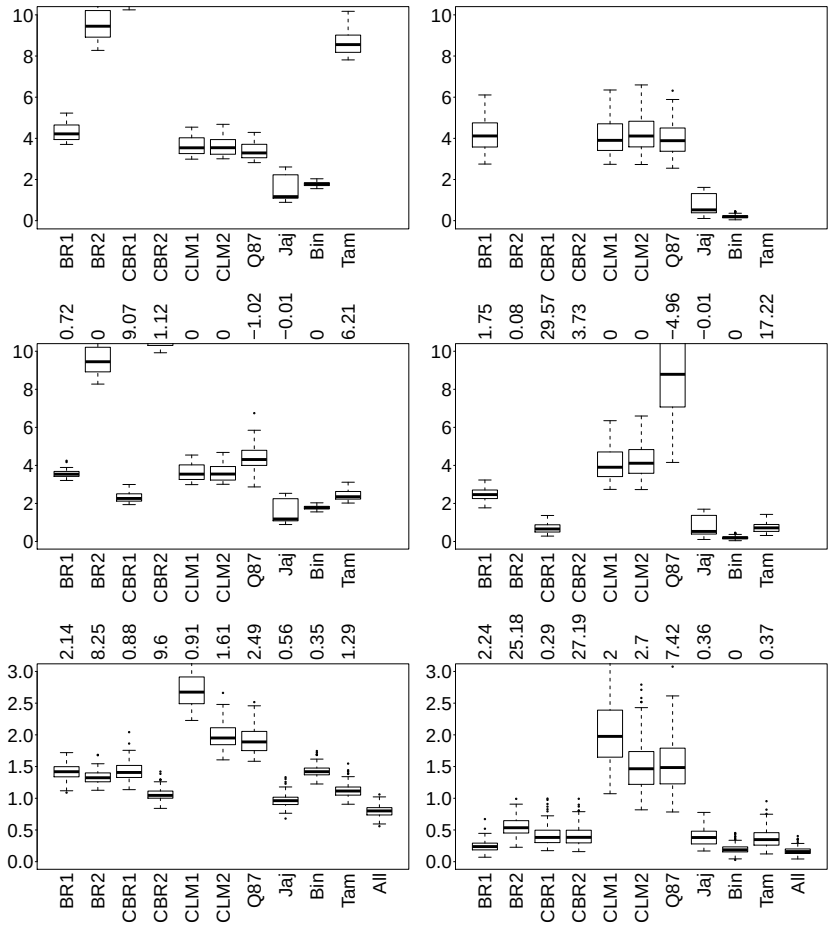

Figure 3.1: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms on random (first row, left plot) and structured instances (right plot). In the top row the algorithms are in their default settings (Level 1, step 1), in the middle row the results with ten seconds on runtime (Level 1, step 2), and in the bottom row the results obtained after tuning their numerical parameters, including the results obtained by the algorithms generated when tuning the whole framework (All, Level 2 and 3). Above the boxplots in the middle and bottom row is given the difference in terms of ARPD from the row above; negative differences correspond to a worsening of the ARPD.

prisingly, some algorithms (BR2, CBR1, CBR2 and Tam) report ARPDs around 10% or higher, much worse than a hill-climbing approach, which

yields an ARPD of about 5%. An explanation for this fact may be that many of these algorithms have been proposed when a much lower computational power was available[2], and have been tuned manually for, by current standards, very small instances and very short computation times; apparently, the parameter settings do not scale well to other settings.

This comparison is, however, unfair: Bin and Jaj run for the maximum allowed time of ten seconds, while the other algorithms in their default setting use a termination based on a maximum number of moves, resulting here in computation times of fractions of seconds. Hence, as a second step still on Level 1, we allow all algorithms to run for the same maximum ten CPU seconds. Surprisingly, only few algorithms (CBR1, CBR2 and Tam, and BR1 on the structured instances) benefit from the higher runtimes; probably because the original parameter settings force a (nowadays) fast convergence, independent from the computation time. Conversely, Q87 worsens its performance, notably on structured instances. This is probably due to a poor specification of the threshold of consecutive rejected moves that triggers the fixed temperature phase in the *Q8-7* cooling scheme. This value is originally set according to the total number of solutions evaluated. Consequently, we scaled this parameter up by estimating the number of moves that can be generated in the given time. Apparently, the choice is not robust to a much larger number of candidate moves.

The results for increased computation time indicate that a re-configuration of the algorithm parameters is necessary for a more fair and meaningful comparison. This we do in Level 2 of our analysis, where we tune the algorithms' numerical parameters. All algorithms now improve their performance; only for Bin on the structured instances no statistically significant difference is observed, probably as it was already automatically tuned originally; the differences on the random instances, may be due to the different tuning setup with respect to the original work.

Almost all algorithms report very good results, meaning that the original algorithmic ideas were seemingly good, but the originally chosen parameter settings did not generalize to settings (e.g. instances, computation times) different from the ones considered in the original papers. The rankings for the two different instance classes differ more than in the previous two cases, reflecting the difference in the two scenarios. The algorithms can be divided in two groups based on their performance after the tuning, with the ones

[2]The processing power alone for mid-range CPUs have increased roughly four orders of magnitude in the last 30 years: an Intel i486 in 1989 measured 8.7 MIPS at 33MHz, while an Intel i7 7500U in 2016 scores 49360 MIPS at 2.7GHz, (with a ratio of 5673). Source: https://en.wikipedia.org/wiki/Instructions_per_second#Timeline_of_instructions_per_second

using the Geometric cooling scheme outperforming those using the Lundy-Mees cooling scheme or its variant *Q8-7*. Re-tuning CLM1, CLM2 and Q87 with a higher budget of 5000 experiments does not result in any improvement on the structured instances, and only a slight improvement for CLM1 and Q87 on the random instances, though not sufficient to match the results obtained by the algorithms using the Geometric cooling scheme.

The tuning also allows us to observe the "potential" of the algorithms and their components. The case of CLM1 and CLM2 is very interesting in this regard. The only difference between these two algorithms is the neighbourhood exploration, with respectively a random and a sequential one (**NE1** and **NE2**). While we do not observe significant difference in the solution quality when using either default settings or extended running time (in both cases p-values of 0.4768 and 0.106 on random and structured instances, respectively), after tuning the numerical parameters there is a clear difference in favour of the sequential exploration **NE2**.

To examine the full potential of SA algorithms, we use Level 3 of our analysis and design automatically a completely new SA algorithm, which may combine the available algorithm components in previously un-explored ways. This is achieved by tuning seven categorical parameters, each related to one of the main components and 89 additional numerical parameters for the various options. In Figure 3.1 we compare the results obtained by these new SA algorithms (indicated by All) with the other ten algorithms tuned over ten seconds; the results of the Friedman rank sum test is reported in Table 3.1. On both instance classes the algorithms All (the rightmost boxplots) outperform the tuned existing algorithms. On the random instances the improvement is more substantial, while on the structured instances it is less strong but still statistically significant, probably because the margin for further improvement on structured instances is minor.

In the supplementary material we include the additional comparison with the SA algorithms automatically generated for anytime behaviour. On the random instances the convergence of All is very good, even for the largest instances. On the structured instances of the main test set, instead, the convergence behaviour is very good, though not as strong as for the SAs for anytime behaviour, while on the larger instances the results are mixed: in two cases the convergence is much slower, in the third case instead the algorithm designed for solution quality converged to significantly better solutions than the SA designed for anytime behaviour, continuing discovering better solutions during the whole runtime.

3.6 Simulated annealing algorithms for the permutation flowshop problem

We have identified three main articles employing SA for the PFSP, which do not use objective-specific tricks or hybridizations. The oldest method by Osman and Potts (OP) uses problem-specific components [214], in particular the initial temperature **IT9**. For it we consider four variants, with random and sequential exploration (**NE1** and **NE2**, indicated by R and S in the algorithm name) and either insert or exchange neighbourhood (indicated by I or E in the algorithm name). The second algorithm, OS, is proposed by Ogbu and Smith [226]; in the original paper the authors also evaluate both the insert and the exchange neighbourhoods. Finally, in the more recent work CH [224] two variants use two different values for one parameter. The details of these algorithms are summarized in the supplementary material. In what follows, we discuss where appropriate separately the results for the instances whose size is smaller than all the instances of our training set (`Tai001` to `Tai030`), the instances whose size is covered by the training set (`Tai031` to `Tai110`), and the instances whose size is instead larger (`Tai111` to `Tai120`). Separate plots for the subsets of instances are provided in the supplementary material. The plots with the results across the whole test set are instead reported in Figure 3.2 in terms of ARPD, obtained by the algorithms in their original settings (Level 1 of the analysis, top row), and after the tuning (Level 2 of the analysis, bottom row); the improvements of the tuning with respect to the default versions is reported on top of the boxplots. As CH1 and CH5 differ only for the value of one numerical parameter, they appear as a single algorithm CH in the tuning. With the tuned algorithms we also include the results obtained when automatically designing SA algorithms. In Table 3.2 we report the rankings obtained for the MS and TCT objectives by the algorithm default settings, and after the tuning.

Overall, we observe a more substantial improvement for the TCT, which is maybe explained by the usage of the MS objective in the original papers. CH1 and CH5 both employ a bounded Metropolis condition and reach results that in the default settings are not significantly different. Probably this is due to a too low setting of the threshold for evaluating worsening solution. In fact, after the tuning of the CH algorithm, the tuned parameter settings promote exploration based on a higher initial temperature, a slow temperature decrease, and a much more relaxed threshold for considering worsening moves, improving especially the performance of CH on the small instances. Interestingly, while for the TCT objective the CH algorithms are

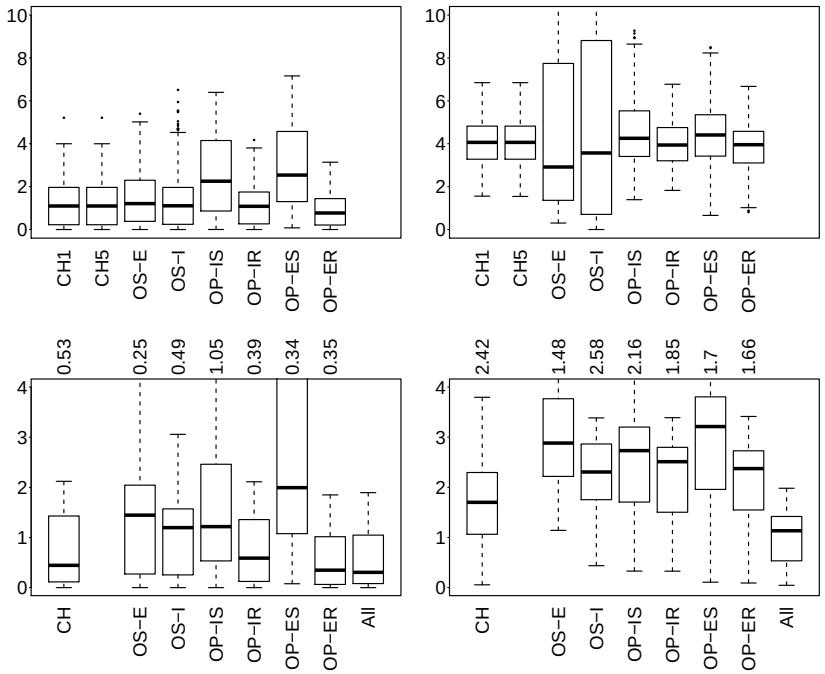

Figure 3.2: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms on the whole Taillard benchmark under the MS and TCT objectives (left and right plot respectively). In the top row the results are obtained with the algorithms in their default settings (Level 1). In the bottom row the results are obtained after tuning the algorithms, collapsing the two CH1 and CH5 into a single CH algorithm and including the results obtained by the algorithms generated when tuning the whole framework (All; Levels 2 and 3). Above the boxplots in the bottom row is given the improvement in terms of ARPD from the top row.

the two worst ones, the tuned CH algorithm is the best among the already existing SA algorithms.

For the OS algorithms, the tuning improves strongly results on larger instances with 50 or more jobs, at the expense of the results on the small instances. A more detailed analysis of the tuned parameters confirms that except for some differences in the resulting temperature length, the main difference between OS-I and OS-E is the neighborhood used. Overall, the OS algorithms exhibit scaling issues even after the tuning, with the results on the larger instances not being of the same quality as on the other instances,

Table 3.2: Comparison of the results of the Friedman rank sum test for the algorithms in their default settings (Level 1) and after the tuning of the numerical parameters (including the automatically generated SAs) on the Taillard benchmark instances for the PFSP-MS (top) and for the PFSP-TCT (bottom) (Level 2). Algorithms are ranked according to their results. In the tuned settings also results for **All** are included (Level 3). Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Settings	Δ_R	Algorithm ranking
Default	53.76	**OP-ER** (0), OP-IR (155.5), CH_5 (179.5), CH_1 (189.5), OS-I (243.5), OS-E (258), OP-IS (478.5), OP-ES (627.5)
Tuned	40.26	**OP-ER** (0), **All** (33.5), OP-IR (213), CH (233.5), OS-I (268), OS-E (476), OP-IS (516), OP-ES (700)
Default	72.47	**OS-E** (0), **OP-ER** (13), **OS-I** (71), OP-IR (122), OP-ES (134), OP-IS (156), CH_1 (159), CH_5 (161)
Tuned	46.82	**All** (0), CH (173), OP-ER (321), OP-IR (378), OS-I (444), OP-IS (547), OP-ES (612), OS-E (685)

especially for the TCT objective.

Of the four implementations from Osman and Potts (algorithms OP-*), the two with a random neighborhood exploration perform better both, before and after the tuning. This is in striking contrast with the findings of the experiments on the QAP, where we found the contrary behavior. Contrarily to the OS algorithms, the exchange neighbourhood outperforms the insert one for both objectives and both before and after the tuning.

As a next step, we apply Level 3 of our analysis, that is, we automatically design SA algorithms from the whole framework. On the MS objective the results are not statistically significantly different from the ones obtained by OP-ER. This may be due to the fact that under the given experimental conditions, the algorithms were already close to the best possible results obtainable by a simple SA with the insert or exchange neighbourhoods. On the TCT objective, instead, the automatically designed algorithms clearly outperform existing algorithms. In particular, we observe a much improved scaling behaviour.

On both objectives the anytime behaviour of the SA algorithms ALL is again good, sometimes even slightly better than the SAs designed for anytime behaviour (which, in turn, have a more regular convergence). On the largest instances the search takes more time to discover a good solution, but

from that point on the behaviour is similar to the SAs tuned for anytime behaviour. The relative plots are given in the supplementary material.

3.7 Analysis of simulated annealing algorithms

We now analyze the data generated for configuring the All variants in Sections 3.5 and 3.6, trying to understand which algorithm components and features are important to make an SA algorithm high-performing. The first step is to observe which components and numerical parameters have the highest importance in the design process. We do so by training a random forest model with the data generated during the configuration phase for All. From this model, we get as a byproduct an estimate of the variable importance. Random forests [244] combine several decision trees, each built using a bootstrap sample of the training data. For each tree t the out-of-bag data (observations missing from the relative bootstrap sample, OOB_t) can be used as validation set, for which we measure (i) the error in the prediction, $errOOB_t$, and (ii) the error in the prediction, $\widetilde{errOOB_t}$, when the OOB_t data is perturbed (by permuting the values of each variable). The importance of a variable V_j is defined as the difference in the prediction error when the values for V_j are perturbed, averaged over the trees. For further insights about the importance of variables in random forest models we refer to [256]. The results of our analysis, normalized to sum one, are reported in Table 3.3 for the four scenarios.

While some differences exist in the different scenarios, overall the single most important component is the acceptance criterion, along with some related numerical parameters, such as the LAHC tenure κ and the deviation factor for the RTR criterion. Another important parameter is the neighbourhood exploration, that is, how the algorithm chooses the next solution to evaluate; to this adds also the importance of the numerical parameter k for the neighborhood exploration options **NE3** or **NE4**. The impact of the acceptance is intuitive, as this component controls the exploration/exploitation tradeoff by determining which worsening moves are accepted. More surprising is maybe the high importance for the neighborhood exploration, a component often neglected in the SA literature. It is interesting to observe that the historically most studied component of SA from both a theoretical and experimental perspective, the cooling scheme, is indeed important (as it influences the behaviour of the acceptance criterion), but not as important as commonly expected: in none of the scenarios it or its associated variables are among the four most highly ranked components or numerical parameters.

On the random QAP scenario the instance is the second most important

QAP Random		**QAP Structured**	
Acceptance Criterion	19.8%	Acceptance Criterion	36.64%
Instance	15.71%	**NE4** k	12%
NE4 k	8.43%	Exploration Criterion	9.33%
Exploration Criterion	8.17%	LAHC κ	5.19%
NE3 k	6.7%	RTR γ	3.97%
CS2 α	4.4%	Instance	3.72%
Cooling Scheme	4.02%	**CS2** α	3.6%
Temperature Restart	3.99%	**NE3** k	3.13%
IT5 k	3.53%	Initial Temperature	2.84%
RTR γ	3.2%	Cooling Scheme	2.72%
PFSP-MS		**PFSP-TCT**	
Exploration Criterion	20.65%	LAHC κ	16.02%
Instance	14.55%	Acceptance Criterion	15.92%
CS6 b	9.24%	Instance	15.33%
CS6 a	5.18%	Exploration Criterion	9.51%
Cooling Scheme	4.76%	**CS2** α	5.62%
CS2 α	4.47%	**NE4** k	3.95%
Acceptance Criterion	4.25%	**NE3** k	2.76%
CS5 b	2.65%	Temperature Length	2.43%
LAHC κ	2.46%	**AC5** r	2.37%
TL9 k	2.25%	Cooling Scheme	2.01%

Table 3.3: The ten most important components and numerical parameters on the four scenarios.

factor, with a contribution more than four times higher than on the structured QAP scenario; the temperature restart component also appears in the list. This is probably related to a more rugged landscape, where the lack of structure makes apparently the algorithms more sensitive to the numerical values of the instances. Also for the two PFSP scenarios, the instance factor has a high importance, which we here conjecture to be due to the rather different instance sizes in the training sets and the differences in the ratio jobs/machines.

Analogous tests on the data collected during the tuning for the anytime behaviour (for detailed results, see the supplementary material) show a higher importance of the instance factor. In particular, the more diverse the dataset, the more important is the instance factor (e.g. for the PFSP case). Another explanation for the stronger impact of the instance is that the maximization of the hypervolume, which is used to measure anytime performance, is more sensitive to parameter settings in the sense that it reduces the set of configurations that result in high performance across various

instances sizes or characteristics; hence it is more difficult for the configurator to reach configurations that generalize to the whole training set. Other components and parameters deemed important are mostly related to temperature initialization and update, which are relatively more important than in the configuration for optimizing the final solution quality.

A detailed analysis of the composition of the automatically generated algorithms is reported in the supplementary material. Here we highlight the most relevant results. The Metropolis criterion **AC1** is the most common choice in three out of four scenarios, appearing 8 out of 15 times on the structured QAP instances, 12 times on the random QAP ones (including the Bounded version twice), and 13 times on the PFSP-MS. On the PFSP-TCT, instead, LAHC is chosen 12 times, with the κ parameter ranging from 99 to 185 (with one outlier case being 5616). The other three algorithms for the PFSP-TCT use a Bounded Metropolis criterion, set to immediately discard solutions whose cost is 0.53% to 0.68% higher than the incumbent; these three algorithms are all paired with a Geometric cooling scheme. LAHC is chosen also in 4 cases in the structured QAP scenario, while 3 times Threshold Acceptance is chosen.

The sequential exploration criterion **NE2** is always chosen for the structured QAP instances; it is also chosen 8 times on the random QAP ones, with the **NE3** criterion selected the other 7 times. On both PFSP scenarios, instead, the random exploration **NE1** is chosen in all cases.

For the cooling scheme, the geometric one, **CS2**, is the only one chosen on the structured QAP scenario, and the one selected most often for the random QAP instances (7 times out of 15, along with five other cooling schemes) On the PFSP-MS the *Q8-7* and its original version Lundy-Mees are chosen respectively 8 and 3 times. For temperature length and restart, instead, we observe a wide range of choices, probably because the sequence of temperature values that yields a very good exploration/exploitation tradeoff can be given by different combinations of the components devoted to its update.

The outcome of the tuning for solution quality of the existing algorithms on the various scenarios is generally a strict acceptance of worsening moves, enforced by low values of the initial temperature and faster cooling than in the original settings. In other words, a well-performing SA algorithm quickly converges to good solutions, and then continues improving. This also explains the good performance of a simple scheme such as LAHC, that can never accept a solution worse than anything else encountered during the search, thus having limited though non-negligible exploration capabilities. The importance of a strong intensification is also reflected in the convergence profiles of the automatically generated algorithms (reported in the supple-

mentary material), that are rather similar to the profiles of the SA algorithms automatically generated for anytime behaviour.

In the supplementary material we include results obtained on the testset with longer runtimes, namely 30 and 100 seconds for the QAP scenarios and with 10× time for the PFSP scenarios; the tunings are the same ones of the previous experiments, performed over 10 seconds. It is interesting to observe that a longer runtime does not necessary entail better results. This happens for both the nontuned and the tuned versions of our set of algorithms. A temperature restart scheme is necessary to escape convergence and profit from additional runtime. If an algorithm not using temperature restarts has already converged to its best solution, more time will not be beneficial; in some cases (e.g. the nontuned versions of BR1 and especially JAJ for the QAP) the ten seconds limit was not sufficient for reaching convergence, and a longer runtime allows that, at least up to a certain extent. The automatically designed algorithms we obtained are able to improve with longer runtimes, in particular when designed for anytime behaviour. As observed especially for the PFSP scenarios, the improvement over a much longer runtime can anyway be limited; the practitioner should therefore consider carefully whether it is actually worth to wait longer, without any guarantee about the improvement of the final solution quality. Tuning over a longer runtime is likely to be beneficial, at the cost of much higher computational requirements.

Considering the configurations obtained from the various scenarios, we may conclude that there does not exist one single high-performance, "general" SA algorithm, but for different scenarios in part rather different settings turn out to be best. This clearly justifies the usage of an automatic algorithm configuration process. However, even for the same scenario, SA implementations may in part differ significantly as a good trade-off between search exploration and exploitation may be reached by different algorithm settings. This is particularly true when considering scenarios that only focus on the final solution quality. However, this is less the case when high anytime performance is required, apparently due to the more refined evaluation of the performance of a configuration.

Let us now mention also the discrepancy between our experimental results and the focus in many theoretical works about SA. As previously mentioned, SA theory has often focused on the aspects of the cooling process and gives conditions on the process to guarantee convergence of the algorithm to optimal solutions–typically achieved by very slow cooling schemes. This has probably contributed to a blind interpretation of the annealing metaphor, where a system is initially at a *high* temperature that *slowly* decreases over time. Consequently, many SA algorithms in the literature follow this folk-

lore, despite a strong convergence and a not-necessarily random neighbourhood exploration have been already discussed in literature as beneficial (see e.g. [218] for the former, and [257, 258] for the latter). Recent experimental works such as [17, 259] have instead studied several acceptance criteria for SA, and align with the present work both in methodology and findings.

Finally, though we stress that a comparison with other methods is neither in the scope of this work, nor entirely possible due to different experimental settings with other existing works, we discuss the efficacy of the automatically generated SA algorithms when compared to different existing methods. For the QAP, there is no single algorithm that can be considered state of the art [83]. The performance of different heuristics has been observed to be dependent on target quality [255], instance size [215] and instance characteristics [83], and several algorithm selection approaches have been proposed [260, 261, 262, 263]. In the supplementary material we report additional results obtained on the commonly used QAPlib [264] by the SA algorithms obtained for the random QAP instances scenario for different running times, where we obtain quite good solution qualities. For the PFSP scenarios we can consider the recent work of Pagnozzi and Stützle [153] as state of the art, with automatically generated hybrid algorithms that outperformed existing alternatives. In this case, the automatically generated SAs are not able to reach the same quality level.

3.8 Conclusions

In this chapter, we proposed to exploit the available knowledge on algorithmic components and parameter setting strategies for metaheuristics in the form of automatically configurable frameworks. As we have argued before, there are some inherent advantages associated to this such as (i) making these components and parameters explicitly available for further use, (ii) building a kind of collective memory of available algorithm options hardly any researcher alone may have readily available, (iii) allowing large-scale experimental analyses and the generation of knowledge under which circumstances which components will be most successful, and (iv) exploiting directly the recent advances in the automatic configuration of algorithms allowing to build potentially higher-performing algorithms than possible by pure manual intervention.

We have made our proposal concrete by building such a framework for Simulated Annealing (SA), one of the most widely studied metaheuristics. We have described the template from which we instantiate SA algorithms and detailed the set of algorithmic choices that are available in our current

framework. Interestingly, also a number of methods that fit the general outline of an SA algorithm can be instantiated from the same template, including methods such as threshold accepting [181], great deluge [209], record-to-record travel [209], or late acceptance hill-climbing [210, 211]. We have also experimentally studied various well-known SA variants for the quadratic assignment problem and two flow-shop scheduling problem variants both using default settings of the literature and different levels of automatic SA algorithm configuration. As maybe expected, the possibility of automatic configuration has shown to be very advantageous, allowing to derive SA algorithms that outperform the variants proposed in the literature even if the numerical parameters of these are also fine-tuned. The experimental data we obtained from the automatic configuration process have also shown to be useful to get insights into what makes a good SA algorithm. An importance analysis has identified the acceptance criterion and the neighbourhood exploration as key important components of SA. When configuring for best reachable solution quality, the cooling scheme seems relevant as it influences the behaviour of the acceptance criterion, but to a lesser degree than one would commonly expect.

Our work can be extended in a number of directions. A first one is to extend the experimental part to automatically generate SA algorithms for other problems and learn about possibly consistent choices and which algorithmic components are the most relevant ones; the knowledge obtained may in turn provide prior biases for the configuration process and lead to the development of new alternative algorithmic components. A second one is to extend our approach to other metaheuristics and to generate extended frameworks. Ideally, these extensions would be generated within a same framework so that possibly rich hybrids among these methods may be generated, as suggested in [265, 266]. This would enable us to compare automatically designed SAs against other automatically designed metaheuristics, to study the role, impact and composition of SA algorithms when combined with or used as part of other metaheuristics, and to understand when and how to move beyond the SA structure to automatically design bottom up new metaheuristics without constraining them to a predetermined form.

A landscape-based analysis of fixed temperature and simulated annealing 4

Since the introduction of Simulated Annealing (SA), researchers have considered variants that keep the same temperature value throughout the whole search and tried to determine whether this strategy can be more effective than the original cooling scheme. Several studies have tried to answer this question without a conclusive answer and without providing indications that could be useful for a practical implementation. In this work, we address this question following an experimental approach, relating the characteristics of the algorithms with the characteristics of the landscapes they encounter. We use problem-independent landscape features to study the algorithmic behaviour across different problems. We consider three different objective functions and various instance classes and determine the conditions under which the fixed-temperature variant of SA can outperform its original counterpart and when SA is instead a better choice.

4.1 Introduction

Stochastic Local Search (SLS) algorithms [12] are a popular class of non-exact methods to tackle optimization problems. They provide a set of guidelines to implement an effective heuristic algorithm for the problem under study. SLS algorithms are particularly useful in complex or black-box scenarios, where it is very difficult to develop a specialized algorithm, and large-

scale scenarios that are beyond the feasibility of exact methods. While they usually are not able to find optimal solutions or to recognize one in case they find it, very often they are able to return high quality solution in short time. The reason for a SLS to perform on an instance boils down to its success in balancing its two contrasting capabilities, the exploration of the search space and the exploitation of the search in promising areas of the search space.

Over the years, plenty of methods have been proposed [10, 11]; some of the simplest of such methods remain among the most popular choices, due to their simplicity in the idea and effectiveness in practice. Yet, it is still difficult to understand their behaviour [13]. In a nutshell, simple single-solution SLSs are built on top of pure intensification local search algorithms such as first- or best-improvement. To obtain better results, they have to introduce a certain degree of diversification and one of the main ways of achieving this is to allow for the acceptance of worsening moves. One of the main examples of such metaheuristics is Simulated Annealing (SA). It uses a parameter called *temperature*, which is progressively lowered during the search to transition from a diversification behaviour to an intensification one [199, 200]. One alternative is to use the same temperature value throughout the whole search [203]. This SA variant appears in the literature under different names [205, 206, 207, 234] and, for consistency, throughout this work we will refer to it as Fixed Temperature algorithm (FTA).

In this work we study how these two simple SLS methods perform on different problems and instances to understand the impact of their diversification strategies on different scenarios. A long-standing open question is to determine which of those two algorithms would obtain better results on a given instance, that is, whether it exists a temperature value for which FTA outperforms SA under any cooling scheme [234]. This question has been addressed in the literature mostly from a theoretical point of view, without a conclusive answer [206, 207, 234, 267, 268, 269]. We take instead a different approach, performing extensive experiments to compare these two algorithms on different problems and instance classes. For a meaningful comparison we obtain the best possible configuration for the algorithms in each scenario. Subsequently, we analyse the configurations obtained and determine the conditions for FTA to match or outperform SA. We focus in particular on some characteristics of the search space, as problem-independent features that we can study across the various scenarios, and to understand what characteristics a search space should exhibit for a FTA to perform as good as or better than SA. The specific research question addressed in this chapter serves therefore also as an example of how to perform problem-independent analyses of algorithms.

We report experiments on three problems, the quadratic assignment

problem and the makespan and total completion time objectives of the permutation flowshop problem. For each objective function we consider two instance classes of different characteristics. We use automatic algorithm configuration to generate the best possible FTA and SA algorithm to observe the potential of the algorithms in each scenario [148]. We analyze the results and configurations obtained, and relate them with information on the landscape of each instance, following an Exploratory Landscape Analysis approach [270]. We show that if the solution neighbourhoods for a given instance of a combinatorial optimization problem have the same structure in different areas of the solution space, then FTA works well. Conversely, if the neighbourhood structures vary for different areas of the search space, SA will find better solutions than FTA thanks to its higher adaptive capability.

In the next section, we review the literature comparing SA and FTA. In Section 4.3 we introduce the experimental settings uses for our experiments, reported in Sections 4.4 and 4.5. In Section 4.6 we discuss our results in relation with the solution landscape before concluding in Section 4.7.

4.2 Literature review

Simulated Annealing (SA) is a popular metaheuristic inspired by the annealing process in metallurgy [199, 200]. Starting from a given initial solution, SA iteratively evaluates its neighbourhood and accepts a move deterministically, if the solution either improves or is the same quality of the current incumbent, or with a probability that depends on the relative worsening of the solution quality with respect to the current incumbent and a parameter called temperature [198]. The temperature parameter is normally set at high values in the beginning of the search and slowly decreased as the search proceeds. This way, the algorithm slowly transitions from an initial exploratory behaviour, where many poor quality solutions are likely to be accepted, towards a final intensification one, where only good quality moves are likely to be accepted. The change of value of the temperature parameter is controlled by the so-called *cooling schedule*, a function that defines a usually non-increasing sequence of temperature values.

Soon after the introduction of SA, researchers began to study the behaviour of SA variants that keep the same temperature value throughout the entire search. This variant has been studied in seemingly independent lines of research, and appears in literature under several names such as Metropolis Algorithm [234, 271, 272], Static SA [206], Generalized Hill Climbing [205], or Fixed Temperature SA algorithm [207, 208] (FTA). For simplicity, we use FTA in the remainder of this chapter, and of this thesis.

The first work that mentioned a SA with a fixed temperature in the optimization literature is probably the one of Mitra *et al.* in 1985 [203]. In 1989, Hayek and Sasaki studied a SA for the polynomial-time matching problem and gave examples for which any monotone decreasing temperature sequence is not optimal [227]. They conjecture that no monotone decreasing temperature sequence is optimal for a broader set of cases. They also consider a (deterministic) threshold random search, prove that there is an optimal sequence of threshold values, and state that probably in many situations there is an optimal deterministic threshold sequence that outperforms any random threshold sequence; incidentally, this can be considered the first study of deterministic variants of SA, later also called Threshold Acceptance [180, 181]. However, they add that "the practical implication of these likelihoods is clouded, since it is unclear how to efficiently find an optimal temperature sequence or deterministic threshold sequence for a problem instance" [227], and thus SA is probably a good fallback solution. This is also an implicit statement about the necessity for an instance-based temperature schedule and about the adaptive capability of SA, something we are going to discuss more in detail in Section 4.6.

In 1996, Jerrum and Sinclair [234] study the Markov Chain Monte Carlo (MCMC) method and conclude with a comparison of SA and FTA (Metropolis Algorithm, MA, in their work) as an application of MCMC. They also consider the matching problem, and observe how MA either solves or finds good approximations on all the instances with high probability in polynomial time. Their method can prove the optimality of a temperature value for a given instance, but no constructive procedure is given to compute it. However, the analogous analysis for SA is much more complex; the outcome is that optimal coolings do exist, but they are so slow that are not competitive with exhaustive search. Hence, it remains an open question whether SA can beat MA on a natural problem.

Mühlenbein and Zimmermann offer an alternative perspective, stating that choosing a proper neighborhood is more important than a cooling scheme, hence their answer to the question whether "to cool or not" is negative, and they suggest instead to use a Variable Neighborhood Search approach [267].

Orosz and Jacobson call the fixed-temperature variant of SA the Static Simulated Annealing (S^2A) and introduce an estimator of the upper bound on the expected number of moves to reach a solution of a certain target quality [206]. They support the theoretical derivation with some experiments on the TSP, but they do not compare S^2A with SA or other algorithms.

Wegener proves instead that SA beats MA for a certain class of Minimum Spanning Tree instances, in terms of expected time and probability of

finding the optimal solution in a given bounded time [268]. Meer applies the same principle to the TSP and constructs some TSP instances for which MA is again outperformed by SA [269].

However, all these contributions are theoretical and focus on existence proofs, and in certain cases they address polynomially solvable problems. For a practicioner's perspective, they share the following drawbacks.

(i) There is no real indication on the right values for a fixed temperature scheme.

(ii) There is no guidance about which algorithm to choose for a certain given problem or problem instance.

(iii) The optimality of a value or a certain method is proven only a posteriori, that is, one should already be able to compute the optimal value in advance. This means, in practice, to solve the problem to optimality in order to be able to compute the optimal temperature value.

(iv) It is unclear how much these existence results generalize to other problems, how they apply to "natural" instances instead of artificially constructed ones, how they generalize when different components are used in the algorithm or, simply, how to consider them for a limited runtime instead of an infinite one.

However, as pointed out in [207]: "convergence is not relevant to the success of the algorithm". In fact, being all these analyses performed from a theoretical perspective, they deal with the "manageable" theory of Markov chains. From an algorithm design and implementation point of view this means that a vast amount of possible design choices that can make the algorithm perform well in practice are ignored. For example, only random moves in the neighbourhood are considered, while a different neighbourhood exploration scheme can be significantly better than a random one on certain problems [16]. Likewise, other effective components such as the temperature restart are ignored in the SA analyses.

Parallel to the theoretical works, some authors noticed that when a cooling strategy was reaching certain temperature values, good solutions were quickly obtained. Thus, the first practical indications on how to obtain these good temperature values began to appear in the literature. The first one we can find comes from Rothman, who according to [273] introduced SA in the Geophysics community in 1985. He noted that "the notion of a *critical temperature* is perhaps the most important, and less understood" open question in the development of a SA algorithm [274]. In a follow-up work,

he presents both a theoretical and experimental analysis on a specific problem, stating that the "solution is obtained by dropping immediately from a high T[emperature] to a low T, and then maintaining this low T [...]"; initial and final temperature values are determined experimentally via trial and error [275].

Connolly tackles the Quadratic Assignment Problem (QAP) in a similar fashion, using SA with a Lundy-Mees cooling scheme [201] and records the temperature at which the best solution is found; after a certain amount of discarded moves, this "optimal temperature" is set and kept fixed [217].

Basu and Frazer also determine experimentally a good temperature value for a geophysics problem they study [276], evaluating several short runs at different temperature values, and choosing the temperature with the best results. It is interesting to note that they justify their approach noting how theoretically optimal cooling schemes are unacceptable in practice.

Cohn and Fielding study several cooling schemes for SA, and find that a fixed temperature works well for some TSP instances, probably depending on the structure of the instance [207]. They estimate the number of moves required to get to a specified target quality, and the relative optimal temperature value, but still deriving those value from preliminary runs of a SA until the best solution is found. Hence, this work runs in the same issues of its theoretical counterparts by still requiring to know or estimate the best solution. They also discuss an example instance from [202], and they show that in that case the optimal schedule is what they call *boiling*, an infinite temperature schedule that accepts every move with probability 1. In other words, according to their analysis the optimal strategy for that instance is a random walk. In a subsequent work, Fielding gives the first closed formula for three problems, TSP, QAP, and Graph Partitioning, again based on the estimate of the number of moves required for a given instance [208]. The formulas are extrapolated from a very limited set of instances, so they do not appear to be particularly robust. In particular, the formula proposed for QAP does not correspond to what we observe in our experiments reported in Section 4.4.

Except for this last work, all the papers comparing SA and FTA study one single problem. There is, however, not clear univocal conclusion that can be drawn from the existing body of work on this subject. The discrepancy in the results reported suggests that which algorithm between SA or FTA performs better is to be determined on a case-by-case basis. In particular, since the optimal (local) search algorithm configuration for a problem and instance depends on the landscape it traverses, in Section 4.6 we will relate the performance of the algorithms with the characteristics of different landscapes.

Algorithm 4.1: Component-based formulation of a SA algorithm. In smallcaps the components we choose via automated tuning, in **boldface** the ones we keep fixed. FTA is obtained by omitting lines $12-14$.

Input: a problem instance Π, a **neighbourhood** $\mathcal{N}$ for the solutions, an **initial solution** s_0, control parameters
Output: the best solution s^* found during the search

```
1  best solution s* = incumbent solution ŝ = s_0;
2  i = 1;
3  t_1 = initialize temperature;
4  while stopping criterion is not met do
5  |  choose a solution s_{i+1} in the neighbourhood of ŝ
   |    according to search space exploration criterion;
6  |  if s_{i+1} meets acceptance criterion then
7  |  |  ŝ = s_{i+1};
8  |  end
9  |  if ŝ improves over s* then
10 |  |  s* = ŝ;
11 |  end
12 |  if temperature length is reached then
13 |  |  update temperature according to cooling scheme and
   |  |    temperature restart;
14 |  end
15 |  i = i + 1;
16 end
17 return s*;
```

4.3 Materials and methods

4.3.1 Algorithms and implementation

We can implement FTA as a fixed temperature SA, and therefore the components that differentiate the two algorithms are those that control the temperature during the execution. More precisely, we consider the traditional geometric cooling scheme for SA and a fixed temperature scheme for FTA; consequently, SA also employs a temperature length and a temperature restart component. The outline of the SA algorithm is given in Algorithm 4.1. FTA is obtained by simply omitting the temperature update components (cooling scheme and temperature length, lines $12-14$ in the outline).

4.3.1.1 Algorithmic components

The set of algorithm-specific components used is a subset of the ones considered in our previous work [16]. Here we list the options. For a more detailed description and original references we refer the reader to [16]. The common components between the two algorithms are the acceptance criterion, for which we use only the traditional Metropolis criterion [198], the initial temperature scheme, the neighbourhood exploration, the termination condition, and the problem-specific components (initial solution and neighbourhood). Importantly, the key parameters we tune are the initial temperature scheme and the neighbourhood exploration, as they do not discriminate between FTA and SA and have a strong impact on the algorithm performance [16].

We implement the set of components for the two algorithms in the EMILI framework [80]. For every experiment reported in this work, the algorithms are instantiated at runtime by selecting the desired combination of components and numerical parameters. The selection is done automatically using the irace configurator [148]. In each experiment, irace finds the best configuration on a training set of instances, and the final configuration is then evaluated on a separated test set. The duration of each experiment is also problem-dependent.

The problem specific components and experimental settings are described in the following sections relative to each problem.

Initial temperature We consider five options: a given initial temperature value, an initial value computed proportionally to a given initial solution value, and three schemes based on a preliminary random walk: (i) a value proportional to the highest gap between consecutive solutions observed, (ii) a value proportional to the average gap between consecutive solutions observed, and (iii) a value that yields a desired initial acceptance probability of worsening moves.

Neighbourhood exploration We consider four options to select one move to evaluate in the neighbourhood: the traditional random scheme, the sequential scheme that evaluates the solutions in a given order until one is accepted, and two partial evaluations of the neighbourhood. These two latter schemes randomly select k solutions in the neighbourhood, and choose either (i) the overall best one in the subset, or (ii) the first improving move found when evaluating the subset in a random order. If no such solution exists in the subset, the least worsening one is selected.

Temperature length The options selected for the temperature length are a constant value, a value proportional to the instance size, a value proportional to the neighbourhood size, and three schemes based on a maximum number of accepted moves: (i) a given constant value, (ii) either a given value or a maximum amount of moves evaluated, or (iii) either a given value or a maximum amount of moves evaluated proportional to the neighbourhood size.

Temperature restart The three options to reset the temperature to its initial value are: (i) no restart, (ii) restart when a given minimum temperature is reached, or (iii) restart when the rate of accepted moves falls below a given threshold.

Termination condition Each algorithmic run is executed for a given runtime. The specific runtime depends on the problem, and is specificied in the following sections.

4.3.2 Quadratic assignment problem setup

The quadratic assignment problem (QAP) is an NP-hard problem that models the assignment of n facilities to n locations [51, 52]. Between each pair of facilities i and j there is an associated flow f_{ij}, and between each pair of locations k and l there is an associated distance d_{kl}. A solution of a QAP instance can be represented as a permutation π, where each element $\pi(i)$ contains the location of facility i. The objective function is

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)}. \tag{4.1}$$

We use the same setup of Chapter 3. We use a random permutation as initial solution, and the neighbourhood function is the exchange neighbourhood

$$\mathcal{N}(\pi) = \{\pi' \mid \pi'(j) = \pi(h) \wedge \pi'(h) = \pi(j) \wedge \forall l \notin \{j,h\} : \pi'(l) = \pi(l)\} \tag{4.2}$$

that swaps two elements in positions i and j in a solution π. The size of the exchange neighbourhood is $n(n-1)/2$. The cost of a solution $s\prime$ neighbour of s can be computed in linear time starting from s.

We use a set of 600 instances in total, equally divided in two classes, that we refer to as random and structured instances. Instances in the first class have data matrices generated uniformly at random [215], while instances in the second class have data matrices generated randomly following

an Euclidean structure [248]. Each class has an equal number of instances of size 60, 80 and 100, which are beyond scale for current exact methods. As no certified optimal solution is available for our set of instances, we use previously computed best known solution values as target values for our evaluation. For each class and size, 50 instances are assigned to the training set for our tuning tasks, and 50 instances to the test set for the evaluation of the configurations obtained. The tuning and testing is done separately for each class and instance size, and the runtime of each algorithm on an instance is ten seconds.

4.3.3 Permutation flowshop problem setup

The permutation flowshop problem (PFSP) is another permutation problem that models several scheduling problems arising in real life [53, 54, 55]. It requires to sort n jobs to be executed on a set of m machines. The ordering of the jobs is the same on every machine, and each job i takes p_{ij} units of time to be completed on machine m. A solution is a permutation π of length n, where each element $\pi(i)$ contains the index of the i-th job to be executed. Several variants of the PFSP exist to model different situations arising in real production environments. These usually involve the completion time of the jobs; we denote with $C_{\pi(i),j}$ the completion time of the i-th job in the solution on machine j. Among the several possible objective functions to be optimized, we consider two of the most studied ones, the makespan objective

$$\min C_{\max} = C_{\pi(n),m}, \tag{4.3}$$

where $C_{\max}$ is the completion time of the last job on the m-th machine (PFSP-MS, [56]), and the total completion time objective

$$\min \sum_{i=1}^{n} C_{\pi(i),m}, \tag{4.4}$$

that minimizes the completion time of each job (PFSP-TCT, [57, 58]). In the variants we consider there is no concurrency nor pre-emption, and all the jobs are ready for execution at instant 0. The PFSP is NP-hard for both objectives.

As in the previous chapter, we use the NEH heuristic to obtain an initial solution for SA and FTA [249]. We consider the insertion neighbourhood that moves a job from position i to position $j \neq i$, resulting in a new permutation

$$\pi' = [\pi(1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(k), \pi(j), \pi(k+1), \ldots, \pi(n)] \tag{4.5}$$

if $j < k$ and

$$\pi' = [\pi(1), \ldots, \pi(k-1), \pi(j), \pi(k), \pi(k+1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(n)] \tag{4.6}$$

if $j > k$. The size of the insert neighbourhood is $n(n-1)$, and the time to evaluate each solution is $O(n)$. Additional experiments with the exchange neighbourhood as defined in the previous section and with a random permutation as initial solution are reported in the Supplementary Material [277].

This time, we use two different instance sets for both PFSP objectives. The first one is a set of randomly generated instances, where we use 40 instances with sizes from 50 jobs and 20 machines to 250 jobs and 50 machines [170]; the test set is instead the Taillard benchmark [250] with 120 instances equally divided in twelve groups with sizes from 20 jobs and 5 machines to 500 jobs and 20 machines. The second benchmark consists in job-correlated and instance-correlated instances from [278], each type of sizes 100×20 and 200×20. The training set is composed of 20 instances for each type and size (thus 80 in total), while for the test set we use 80 instances for each type and size (320 in total). The runtime limit for each algorithm execution is $(n \times m \times 0.015)/2$ seconds.

4.3.4 Experiments outline and computational environment

In Sections 4.4 and 4.5 we are presenting the results obtained on the QAP and on the PFSP (both objectives) benchmarks, respectively. For each objective function we have two classes of instances, each one containing instances of different size.

We configure a FTA and a SA for each instance class (for separate instance sizes for the QAP, considering all the instance sizes available for the two PFSP objectives). The choice of non-fixed components and numerical parameters is done automatically using the irace R package [148]. Each tuning is repeated 15 times, and each configuration obtained is tested 15 times. A fixed set of random seeds is used throughout all the experiments. For each tuning we use a budget of 5000 experiments, all run on Intel Xeon E5-2680 v3 CPUs running at 2.5GHz, with 16MB cache and 2.4GB of RAM available per algorithm execution. Each algorithm execution is single-thread. The complete list and range of parameters is reported in the Supplementary Material, where also the tuning setups, the instances used and some preliminary experiments are included [277].

For each problem/instance class scenario, we report the results obtained by the final configurations on the relative testbed in terms of percentage deviation from the best known or optimal solutions. We report also a sum-

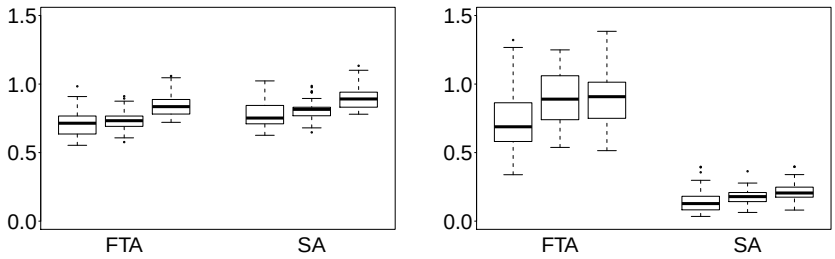

Figure 4.1: Results in terms of Average Relative Percentage Deviation (ARPD) obtained by the FTA and SA on the QAP random (left plot) and structured (right plot) instances. For each algorithm, the boxplots report the results obtained on instances of size 60, 80, 100.

mary of the configurations that we obtain, to understand what parameters are chosen by irace in each scenario.

In Section 4.6 we finally explain the results and configurations obtained by relating them with the landscape characteristics of each scenario.

4.4 Results on the quadratic assignment problem

We first consider the Quadratic Assignment Problem (QAP). We configure the FTA and SA algorithms separately on our training sets of instances, and compare the results that we obtain on the respective test set. Unless indicated otherwise, we consider separately both the different classes and the different instance sizes, having 25 training instances and 25 test instances for each algorithm. The results in terms of average percentage deviation from the best known solutions are reported in Figure 4.1 for both random and structured instances. We begin by discussing the results and configurations obtained on the random instances.

4.4.1 Random instances

As we can see in the left plot of Figure 4.1, the FTA obtains sligthly better results than SA on the random instances. For sizes 60 up to 100, the solutions found by FTA are on average 0.715, 0.734 and 0.844 worse than the best known solutions, while SA obtains solutions on average 0.781, 0.811 and 0.898 worse than the best known solutions. There is therefore a statistically significant difference in favour of FTA, with p-values of 1.42×10^{-7}, 1.3×10^{-8} and 5.44×10^{-7} for, respectively, instance sizes 60, 80, and 100.

Table 4.1: Occurrence of each component in the FTA configurations obtained out of 15 tunings on QAP random instance sizes 60, 80, 100, and on all the instance sizes.

Component	Size 60	Size 80	Size 100	All sizes
proportional to the average gap	8	10	4	11
proportional to the max gap	1	2	5	1
initial acceptance probability	3	2	6	2
fixed initial value	0	0	0	0
value based on initial solution	3	1	0	1
random exploration	0	0	0	0
sequential exploration	8	9	6	12
best of k	7	6	9	3
first best of k	0	0	0	0

In Table 4.1 we report the occurrence of each component, among the ones that appear in the final FTA configurations, for the tunings on the different instance sizes and for additional tunings that consider all the instance sizes. The best final FTA configurations for the general random scenario that we obtain are composed by (i) an initial temperature value obtained with an initial temperature scheme proportional by a factor of around 0.22 to the average gap between consecutive solutions in a random walk, and (ii) a sequential exploration. Thus, it looks like the initial temperature value is related to the instance size by means of the relative difference between solutions. With a scaling coefficient of 0.22, the acceptance probability of an average worsening move (that is, worse by the average gap found during the initial random walk) is around 1%. These configurations are observed more consistently in the tunings across all the instance sizes (last column of Table 4.1), because the homogeneity of the instances in each size makes it possible for other parameter combinations to obtain the same results. The initial temperature scaling coefficient is the key factor for the sequential exploration to make an impact. When this parameter combination is not found, the best alternative is a "best-of-k" exploration that selects the best solution in a randomly chosen subsample of the neighbourhood, with k consistently covering a circa 20% fraction of the neighbourhood. This configuration seems to be easy to find and overall quite robust. The reason for this is the "pure intensification" step of the exploration is able to balance the diversification entailed by a wide range of temperature values, even much higher

Table 4.2: Occurrence of each component in the SA configurations obtained out of 15 tunings on QAP random instance sizes 60, 80, 100.

Component	Size 60	Size 80	Size 100
proportional to the average gap	5	2	6
proportional to the max gap	3	3	2
initial acceptance probability	3	6	3
fixed initial value	3	3	1
value based on initial solution	1	1	3
random exploration	1	0	0
sequential exploration	0	2	2
best of k	4	8	6
first best of k	10	5	7
fixed temperature length	0	2	2
length proportional to instance size	2	4	4
length proportional to neighbourhood size	4	1	4
maximum number of accepted moves	2	1	2
maximum number of accepted moves, capped	2	4	2
max no. of accepted moves, cap cap $\propto$ neigh. size	3	1	0
no temperature length	2	2	1
temperature restart based on min. temperature	5	3	4
restart based on low acceptance rate	3	5	5
no temperature restart	7	7	6

than the best one.

We consider also additional experiments, reported in the Supplementary Material, that use the random exploration [277]. This is relevant because the random exploration is the common choice for SA algorithms and their variants, and random instances are often used in the literature. With these experiments we can therefore compare the results and configurations we obtain with the existing literature. Our experiments indicate that using the random exploration the best temperature value is consistently around 0.18, slightly lower than with the sequential exploration (0.22). However, even with the lower temperature the results are not as good as the results obtained with the sequential exploration, and, in fact, it was never selected by irace in our primary set of experiments. Solutions found using the random exploration are around 16% worse than those found using the sequential exploration, and are also slightly worse than those found by SA.

Table 4.3: Occurrence of each component in the FTA configurations obtained out of 15 tunings on QAP structured instance sizes 60, 80, 100, and on all the instance sizes.

Component	Size 60	Size 80	Size 100	All sizes
proportional to the average gap	13	14	12	14
proportional to the max gap	2	0	0	0
initial acceptance probability	0	0	1	0
fixed initial value	0	1	1	0
value based on initial solution	0	0	1	1
random exploration	1	3	7	5
sequential exploration	12	10	6	9
best of k	1	1	1	0
first best of k	1	1	1	1

The set of final configurations of the SA algorithms is reported in Table 4.2. As we can see it is very diverse, meaning that several cooling-based algorithms obtain the same results on our set of random instances, but none of them outperforms the best FTA configurations. We can, however, note how the "best-of-k" exploration (or its first improvement variant) with k covering roughly $1/4$ of the neighbourhood size appears in half of the final configurations, strengthening the suitability of alternative neighbourhood exploration on this scenario.

4.4.2 Structured instances

As seen in the right plot of Figure 4.1, the results on the structured instances are very different than the previous case, with SA clearly outperforming FTA. In fact, a SA tuned with a budget of only 500 experiments obtains better results than a FTA tuned with a budget of 5000 and less parameters. More precisely, FTA found solutions on average 0.764, 0.891 and 0.901 worse than the best known solutions for instances of size respectively 60, 80 and 100. For the same instance sizes SA obtains relative percentage deviations of 0.148, 177 and 0.213.

This is clearly a scenario where FTA is not a good choice. Nonetheless, there is a clear indication about what FTA configurations are better on this scenario, as we can see in Table 4.3. The initial temperature scheme based on the average gap between consecutive solutions in a random walk is chosen most consistently, with the other schemes appearing only sporadi-

cally. The difference with the random scenario is that the scaling coefficient is now dependent on the instance size. For instance size 60, the average coefficient is 0.095, for instance size 80 it is 0.078, for instance size 100 it is 0.067 when using the sequential exploration. This means an average acceptance probability of a random average solution of, respectively in these three cases, 0.0027%, 0.00028%, and 3.013×10^{-5}%. Essentially, as the instance size increases, the optimal temperature decreases. The reason for this are discussed in Section 4.6.

The sequential exploration is once again the most common choice. Conversely to the random instances case, however, the second most common option is the random exploration, while the remaining two schemes rarely appear.

The choice of the configurations, and, in particular, of the initial temperature values, is caused by the structure of the instances: if there is an "easy" valley in the fitness landscape, too much diversification will hamper the search convergence; however, the structure of the slope makes it impossible to find configurations that work fine for both the initial and the latter stages of the search.

The composition of the SA algorithms is again very diverse, as can be seen in Table 4.4, but in this case the results are consistently good. With respect to FTA, in fact, SA with its transition from "high" to "low" temperature values can better fit the fitness landscape. The temperature restart employed by most of the SA algorithms also makes it possible to cycle between exploration and exploitation several times, increasing the possibility of finding good temperature values for different sequences of neighbourhoods observed. This repeated transition is sufficient to adapt the behaviour of the algorithm, while the specific choice of the components is less relevant.

4.4.3 Discussion

On these two scenarios, we can observe very different results. On the random instances, no configuration for SA matches the results that are obtained by a proper temperature value of a FTA. This is caused by the shape of the landscape, which can be described as "uniformly smooth", in the sense that the proper temperature values for average-quality areas of the search space are almost the same around good quality solutions. The situation clearly differs on the structured instances, where SA can better adapt to the change of the landscape. In fact, the variety of SA configurations observed suggests that a progressive adaptation of the exploration/exploitation tradeoff matters rather than a well-defined SA structure. Section 4.6 is devoted to more detailed analysis of how the FTA and SA are affected by the landscape.

Table 4.4: Occurrence of each component in the SA configurations obtained out of 15 tunings on QAP structured instance sizes 60, 80, 100.

Component	Size 60	Size 80	Size 100
proportional to the average gap	4	5	4
proportional to the max gap	1	1	3
initial acceptance probability	3	2	3
fixed initial value	5	3	4
value based on initial solution	2	4	1
random exploration	12	10	5
sequential exploration	3	5	9
best of k	0	0	0
first best of k	0	0	1
fixed temperature length	1	5	3
length proportional to instance size	0	1	0
length proportional to neighbourhood size	3	1	3
maximum number of accepted moves	6	0	3
maximum number of accepted moves, capped	4	2	3
max no. of accepted moves, cap $\propto$ neigh. size	0	2	2
no temperature length	1	4	1
temperature restart based on min. temperature	12	7	5
restart based on low acceptance rate	2	3	6
no temperature restart	1	5	4

Additional experiments are included in the Supplementary Material [277]. We include experiments with various runtimes and tuning budgets. The best configurations we obtained do not change for different runtimes, where the solution quality increases monotonically with the runtime, nor when we rescale the instance values. This confirms that an initial temperature scheme based on the difference between neighbouring solutions is a proper and robust choice. We also observe how for increasing tuning budgets the results improve, but not as much as it can be expected; in fact, for the two instance classes, a tuning budget of 500 experiments is already sufficient to discriminate between good and bad performing algorithms. We also observe how starting with a good quality solution (computed with a hill climbing, which on our QAP instances can reach solutions between 4 and 5% of RPD) does not impact neither the final solution quality nor the configurations we obtain.

We can now comment on Fielding's temperature formula for the QAP $T = 1.5 \times f^\star / n^2$, where $f^\star$ is the optimal solution value and n the instance size. Fielding extrapolated his formula from results obtained on seven small QAPLIB random symmetric instances,[1] similar to those used in our experiments. The temperature values obtained are roughly inversely proportional to the neighbourhood size ($n(n-1)/2$). In our experiments, instead, the (near-)optimal temperature values are related to the relative difference between solutions. On our test instances, the effect of Fielding's formula is a roughly constant temperatue value for increasing instance sizes, while the initial temperature scheme chosen by the tunings in our experiments yields increasing temperature values for increasing instance sizes. The same happens when using the random exploration, the same scheme used by Fielding. Hence, we see that Fielding's formula is a clear overfit over a restricted set of small instances, whose outcome does not generalize.

We note instead how one of the findings of [16] is confirmed in these experiments, namely that the optimal diversification is the minimal one needed to "smoothen" the landscape. irace is able to find the best temperature value for such behaviour, and no configuration among the final ones has values lower than the best one [22].

4.5 Results on the permutation flowshop problem

In Figures 4.2 and 4.3 we show the results obtained by our set of algorithms on the Taillard benchmark, for the Makespan and Total Completion Time objectives, respectively, in terms of relative percentage deviation (RPD) with respect to the optimal or best known solutions. For each objective, we show separately the results obtained when testing the FTA and SA algorithms on (i) the 30 Taillard instance with a number of jobs and machines included in our training set (50×20, 100×20, 200×20), (ii) the 80 Taillard instances with a number of jobs included in our training set (with 50, 100 and 200 jobs), and (iii) the whole Taillard benchmark. For simplicity, we refer to these three sets of instances as TIJM, TIJ, and TIA. In Figure 4.4 we show the results obtained for the two objectives on the Watson benchmark.

All the algorithms use the insertion neighbourhood, and start from an initial solution computed using the NEH heuristic. The components selected by the tunings for the experiments reported in this section are listed in Tables 4.5 and 4.6 for FTA and SA respectively. In the Supplementary Material we report a full breakdown of the 12 instance classes of the Taillard benchmark, and additional experiments where the algorithms use

[1] Instance `sko100a` was left out of the regression as an outlier.

the exchange neighbourhood and a random permutation as initial solution [277].

4.5.1 Makespan objective

4.5.1.1 Taillard benchmark

Given the variety of the instance sizes and characteristics of this benchmark, where the instances are divided into 12 different groups, overall the results show a high variance. We can anyway distinguish a certain trend. For the TIJM and TIJ instances, whose characteristics are covered in the training set, FTA and SA obtain very similar results (p-values respectively of 0.0221 and 0.02553). On the whole benchmark, instead, SA outperforms FTA (p-value of 1.237×10^{-7}). In terms of solution quality, the results can be very different between different subclasses of instances. We observe also a general higher consistence of the results in each subclass of instances, except for instances of size 50×10, where while the average solution quality of the two algorithms is similar the variance is high.

Instances in TIJM have a low jobs/machine ratio, a case notoriously difficult in practice. On these thirty instances, FTA and SA obtain very similar results. The TIJ subset includes several instances with higher jobs/machine ratios, easier to tackle, and, in fact, the performance improves for both algorithms, remaining similar overall. SA can instead significantly outperform FTA on the remaining instance groups, which include the smaller instances and the largest group, 500×20. Perhaps surprisingly, on the largest instances both algorithms again perform similarly well. Where SA outperforms FTA is instead on the smallest instances, which have a number of jobs lower than any instance in our training set. On these instances, SA is often able to find the optimal solution, while FTA usually does not, and this explains the statistically significant difference in results observed in the third plot of Figure 4.2.

Regarding the FTA configurations, the majority of them use an initial temperature based on the average gap between consecutive solutions in a random walk, with a scaling factor for the initial temperature consistently around 0.066, and a random exploration. The remaining ones use a first or best improvement in a small random subsample of the neighbourhood as neighbourhood exploration. This scheme admits a greater variety of initial temperature schemes and values, which therefore appear albeit sporadically in the list of final FTA configurations.

Regarding the SA algorithms, instead, the set of options chosen for the initial temperature is much wider. Many of the algorithms feature instead

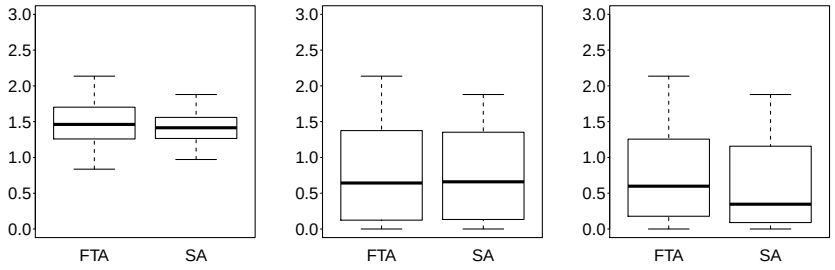

Figure 4.2: Results in terms of Average Relative Percentage Deviation (ARPD) obtained by the FTA and SA on the PFSP-MS Taillard benchmark. The three boxplots report the results for, left to right, (i) instances with a number of jobs and machines included in the training set (TIJM), (ii) the instances with a number of jobs included in the training set (TIJ), (iii) all instances (TIA).

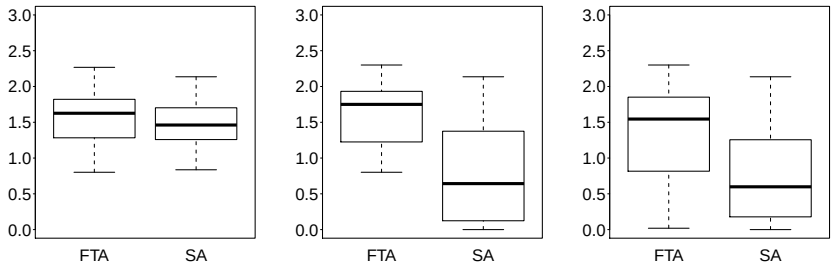

Figure 4.3: Results in terms of Average Relative Percentage Deviation (ARPD) obtained by the FTA and SA on the PFSP-TCT Taillard benchmark. The three boxplots report the results for, left to right, (i) instances with a number of jobs and machines included in the training set (TIJM), (ii) the instances with a number of jobs included in the training set (TIJ), (iii) all instances (TIA).

a cooling coefficient in the range 0.6 to 0.9 range, no temperature restart, a temperature length based on a maximum number of accepted moves, and random exploration of the neighbourhood.

4.5.1.2 Watson benchmark

The results on the Watson benchmark are very homogeneous, and the optimal value of many instances is met or the best known value improved. A

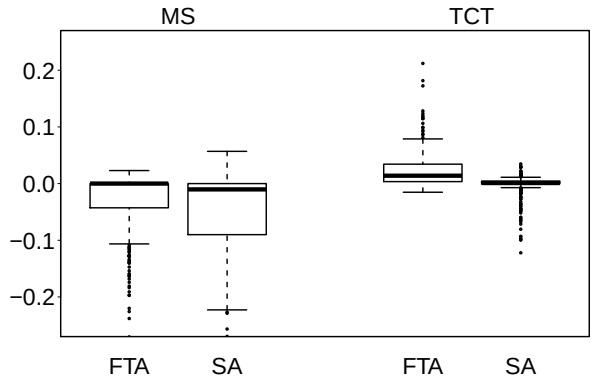

FIGURE 4.4: Results in terms of Average Relative Percentage Deviation (ARPD) obtained by the FTA and SA on the Watson benchmark under both the MS and TCT objectives.

Table 4.5: Occurrence of each component of FTA in the configurations obtained out of 15 tunings on the four PFSP scenarios (Makespan and Total Completion Time objectives, Taillard and Watson instances).

Component	MS T	MS W	TCT T	TCT W
proportional to the average gap	12	15	12	4
proportional to the max gap	1	0	0	3
initial acceptance probability	1	0	1	1
fixed initial value	0	0	1	2
value based on initial solution	1	0	1	5
random exploration	10	15	11	1
sequential exploration	0	0	0	0
best of k	2	0	0	10
first best of k	3	0	4	4

paired Wilcoxon test indicates that there is no statistical difference between the FTA and SA results, with a p-value of 0.5865.

The set of FTA configurations in this case is the most homogeneous among our experiments. All the fifteen FTAs use again an initial temperature based on a preliminary random walk, with an average scaling factor of 0.093, and a random exploration. The SA algorithms, instead, regardless of neighbourhood and initial solution scheme, exhibit a variety of choices

Table 4.6: Occurrence of each component of SA in the configurations obtained out of 15 tunings on the four PFSP scenarios (Makespan and Total Completion Time objectives, Taillard and Watson instances).

Component	MS T	MS W	TCT T	TCT W
proportional to the average gap	6	7	3	7
proportional to the max gap	1	2	4	3
initial acceptance probability	2	5	5	2
fixed initial value	2	0	1	2
value based on initial solution	4	1	2	1
random exploration	15	15	15	15
sequential exploration	0	0	0	0
best of k	0	0	0	0
first best of k	0	0	0	0
fixed temperature length	1	1	0	6
length proportional to instance size	0	0	0	0
length proportional to neigh. size	0	1	0	1
max number of accepted moves	13	10	15	2
max number of acc. moves, capped	1	3	0	6
max no. of acc. moves, cap $\propto$ neigh. size	0	0	0	0
no temperature length	0	0	0	0
temp. restart based on min.value	0	0	0	0
restart based on low acceptance rate	1	4	0	5
no temperature restart	14	11	15	10

and combinations, in particular, for the initial temperature. The only steady non-fixed choice is the random exploration of the neighbourhood, but also the temperature length scheme and the absence of a restart option appear often.

4.5.2 Total completion time objective

4.5.2.1 Taillard benchmark

When focusing on the TIJM sets of instances, FTA and SA obtain similar results: a pairwise Wilcoxon test gives a p-value of 0.08407. On the TIJ and TIA sets of instances, instead, FTA instead does not obtain results close to those obtained by SA.

The explanation we give is that the instances outside TIJM are easily

tackled by the progressive transition in the exploration/exploitation tradeoff of SA algorithms regardless of the specific components used, similarly to what we observed in the structured QAP instances (see Table 4.5). This interpretation is supported by the variety of SA configurations obtained, and by the landscape analysis of Section 4.6.

Eleven out of our 15 FTA algorithms have random exploration and average gap-based initial temperature with a scaling factor of 0.071. The remaining ones have a partial local search in the neighbourhood coupled with different initial temperature schemes yielding a higher initial temperature. Simulated annealing algorithms also have several different initial temperature options. However, they all employ the random exploration of the neighbourhood, no temperature restart scheme, and a temperature length based on a maximum number of accepted moves.

4.5.2.2 Watson benchmark

The results on the Watson benchmark are very homogeneous, and the optimal value of many instances is met or the best known value improved. Nonetheless, SA is still significantly better than FTA with a p-value of 2.2×10^{-16}.

The FTA configurations in these cases are very different; the only shared feature is a partial local search in the neighbourhood as exploration criterion. This is probably the factor that controls the exploration/exploitation tradeoff precisely enough to reach the best results. As we have seen with the QAP experiments in Section 4.4, this exploration scheme admits a wide range of temperature values, and, in fact, very different fixed initial temperatures appears. For the SA algorithms, instead, we again observe several options chosen for the initial temperature scheme. Most SA configuration also have no temperature restart scheme, and a variety of temperature length schemes. All the configurations, however, share the random exploration of the neighbourhood.

4.5.3 Discussion

We observe how, on our benchmarks, FTA can obtain results close to those of SA. It is however crucial to choose the right subsidiary components (neighbourhood and neighbourhood exploration) and to tune the temperature value on an instance distribution that matches the test set. When the test instances have different characteristics than the ones used for the training, SA is instead a better choice.

Also in this case, we report in the Supplementary Material additional experiments, where we use the exchange neighbourhood, a different initial solution, and an additional perturbation scheme [277]. In general we observe a regularization effect in the configurations introduced by the perturbation. The values of the initial temperature of the intensification FTA are more consistent when introducing the perturbation, and the same applies to the choice of components. This observation is true across different neighbourhoods and initial solutions, as it can be seen from the additional experiments included in the Supplementary Material. We explain this by the fact that the separation of the intensification and diversification phases makes it possible to have each step tailored for its specific task. There is instead no noticeable impact by the use of the NEH heuristic for the initial solution, as the results are not significantly different from the results obtained starting from a random initial solution for neither FTA nor SA.

We can also explain why the neighbourhood of choice for the PFSP is the insertion one rather than the exchange one as in the QAP. The exchange neighbourhood, in fact, generates a more rugged landscape than the insertion neighbourhood; exchange-based algorithms usually get stuck in a poor local optimum in a few moves. The sets of final solutions found by the algorithms that employ the exchange neighbourhood also have a higher variance than those found using the insertion neighbourhood.

The landscape generated by the Watson instances is extremely smooth, and a randomly generated permutation is expected to be 4% away (on some instances, less than 0.5% away) from the optimal or best known solution; in some cases a simple iterated improvement is sufficient to find the optimal solution. The variety of the configurations obtained and the lack of temperature restart schemes in many SA configurations are further indications of the easiness of the benchmark. The test set also comes from the same instance distributions of the training set. It is therefore not surprising that all the algorithms are, for both objectives, around or close to the optimal or best known solutions. The count for optimal or best solutions found or improved is reported in the Supplementary Material [277].

4.6 Exploratory analysis of landscape features

FTA and SA introduce a diversification mechanism over a pure intensification local search method. In this section we address the question of how the problem and instance characteristics impact over the effectiveness of this mechanism. Having tested three different objective functions on instances of various characteristics in Sections 4.4 and 4.5, in this section we report

the results of an exploratory analysis to understand under which conditions the diversification mechanism is more effective. In other words, we are interesting in finding out the conditions under which the constant rate of diversification employed by FTA suffices to obtain good results, and when the flexibility introduced by SA is necessary to outperform FTA.

To be able to study different problems, we need to use problem independent features. In discrete optimization, problem specific features are often considered, for example, in algorithm selection. We use instead an approach that follows the exploratory landscape analyses for continuous optimization problems and tries to characterize the solution landscape [270]. This is a valid approach also for algorithm selection [121]. In discrete optimization, however, it is an open question which kind of features should be considered to represent the landscape. Some recent works considered fitness landscape features for algorithm selection on the QAP, using global features such as fitness distance correlation and average distance to optimal solutions [261, 279]. In this work, however, we have a slightly different albeit related perspective. We do not seek to select different algorithms, but rather to understand how variants of a generic algorithmic template behave on different landscapes. Thus, under our perspective the landscape is generated not only by the objective function and the instance data, but also by the problem-specific components of the algorithm used, in particular in our case the neighbourhood function. By choosing a set of features that capture the behaviour of the algorithm on the landscape, we are then able to understand the relationship between a landscape and the performance of a specific algorithm used to traverse it. We choose therefore to use a set of *algorithmic-dependent* features to represent the landscape that is seen by the algorithms we use.

Since FTA and SA introduce a diversification mechanism on top of an iterative improvement, we probe the search space with a first-improvement and a best-improvement algorithm. By using these algorithms we want to understand what conditions are necessary to escape the local optima, and to relate them to the efficacy of each diversification mechanism tested. The features we compute include measures about the convergence of the search and statistics of the neighbourhoods traversed and are oriented towards observing whether a certain property holds constant throughout the fitness landscape or it changes in different areas of the search space. The full list of features is given in Table 4.7, and here we describe the most relevant ones, as resulting from the analysis, along with their rationale.

4.6.1 Feature analysis

For each instance of our test sets, we have run 30 first-improvement (FI) and 30 best-improvement (BI) algorithms, with different random seeds, from which we extract the landscape features used in our analysis. This means that each feature is the aggregated value (average or standard deviation) of 30 values, and every feature is generated twice, one time from the first-improvement runs (indicated as FI), and another from the best-improvement ones (BI). Each algorithm starts from a random initial solution. For the QAP we use the exchange neighbourhood, while for the two PFSP objectives we use the insertion neighbourhood.

The diversification mechanism of FTA accepts two equally worsening moves with the same probability at each step of the search, while SA, as usually defined, has a larger tolerance in the beginning and is much stricter at the end. We have empirically observed that a SA works well with overall low acceptance probabilities (less than 1% in the early stages), otherwise the algorithm will fail to converge to good solution areas. This means that, for FTA to be effective, the conditions for escaping a local optimum without failing to converge at all needs to be roughly the same at any stage of the search. For this to happen, the neighbourhoods traversed have to have the same properties regardless of whether they are centered around good or bad quality solutions. Conversely, the more flexible SA can exploit different settings in different stages of the search to adapt to different landscape profiles. One way of observing this difference is to observe whether the sequence of best solution values found can be better approximated by a linear model, indicating a flatter landscape where FTA can work well, or an exponential model, where SA can exploit its flexibility. We consider the R^2 error of the two models as features, along with their difference. The same model fitting idea can be applied also to the sequence of possible improving moves that appear in the neighbourhoods traversed.

Similarly, we consider that the Metropolis criterion works by rescaling the difference between the incumbent and the candidate solution values. For FTA to escape a local optimum or to deviate from a path that converges to a local optimum, the temperature value needs to be suitable to neighbourhoods of both good and bad solutions. As a proxy for this, first we record the difference between the incumbent solution value and the average solution value in its neighbourhood and then we fit a linear model on the sequence of such differences. A flat slope means that the effort it takes to escape the current path is the same throughout the whole search, while an upward slope means that the local optima are situated in deep valleys, more difficult to escape than other regions of the search space.

Other features that are apparently relevant in our analysis are relative to the average length of the path from a random initial solution to a local optimum. It is also interesting to consider values rescaled by instance or neighbourhood size.

In preliminary analyses we also considered features based on initial and final solution values and relative percentage deviations from the best known solutions (RPD). Several of these features turned out to be the ones with highest predictive power. However, this can be explained with the relatively diverse set of values and RPDs obtained on our set of instances, such that final solution values observed on problems and instance classes where SA and FTA perform similarly well do not overlap with solution values of problems and instances where SA clearly outperforms FTA. Hence, these measures appear to be too dependent on our specific examples. Because of this fact, and because these features are mostly available only a posteriori, we prefer to not include them in the analyses reported in the present section.

We have a set of 38 features observed across the three different objective functions used. For each objective function we have one subset of test instances on which the two algorithms obtain similar results, and another subset on which there is a significant difference in the results. We define a prediction task where, given an instance, the goal is to predict the difference between FTA and SA in terms of relative percentage deviation from the best known or optimal solutions. By training a random forest model for this task, we can estimate the contribution of each feature to the task, and we can select the most relevant features according to the root mean-squared error (RMSE), the standard deviation of the prediction error, computed on the resampling of a 10-fold cross-validation. We use the *caret* R package [280] to run our analyses.

We first report the analysis on the whole set of results, and then we further explore the analysis for each objective function considered, reporting the specific differences arising under each objective function. Since replications of the analyses reported may differ slightly in the number or the order of features selected, for each objective we run ten analyses and identify the features selected most consistently.

4.6.2 Analysis on the whole dataset

The analyses across the three different objective functions are the ones that give the most consistent results; in fact, the most important features and their order is largely the same throughout the ten replications of the analysis.

The most important feature is the difference between the R^2 errors of a linear and an exponential model fit on the sequence of the normalized so-

Table 4.7: Set of features used in the analysis. Each feature is computed with both a first-improvement and a best-improvement local search. The aggregated features are computed from 30 independent runs.

1	Average number of moves to local optimum
2	Standard Deviation of number of moves to local optimum
3	Average number of moves to local optimum, rescaled by instance size
4	Standard Deviation of number of moves to local optimum, rescaled by instance size
5	Average number of moves to local optimum, rescaled by neighbourhood size
6	Standard Deviation of number of moves to local optimum, rescaled by neighbourhood size
7	Average R^2 of a linear model fit on the solution values
8	Average R^2 of a linear model fit on the solution values normalized in $[0, 1]$
9	Average R^2 of an exponential model fit on the solution values
10	Average R^2 of an exponential model fit on the solution values normalized in $[0, 1]$
11	Average R^2 of a linear model fit on the number of improving moves in the sequence of neighbourhoods traversed
12	Average proportion of neutral moves in the neighbourhoods traversed
13	Number of neutral last moves
14	Difference of features 7 and 9
15	Difference of features 8 and 10
16	Average slope of the sequence of differences between best and average solution in a neighbourhood
17	Standard deviation of the slopes of the sequence of differences between best and average solution in a neighbourhood
18	Average of the standard deviation of the sequence of differences between best and average solution in a neighbourhood
19	Standard deviation of the standard deviation of the sequence of differences between best and average solution in a neighbourhood

lution values, followed by the average number of moves needed to converge to a local optimum, rescaled by the neighbourhood size. The third most important feature is again the difference between the R^2 errors of the two models, but this time on the actual solution values and not on their rescaled

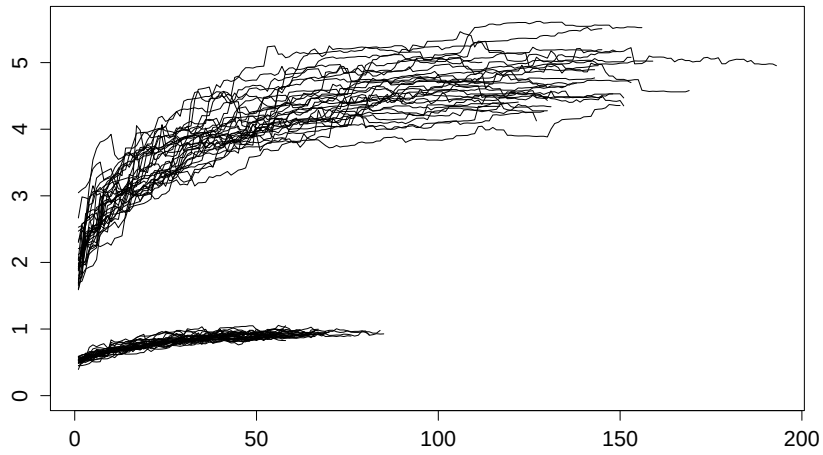

FIGURE 4.5: Convergence of thirty BI algorithms on a QAP structured instance (top) and thirty BI algorithms on a random instance (bottom). The x axis reports the moves accepted, and on the y axis we report the ratio (in percentage) between the average gap in the neighbourhood (of the corresponding move) and the value of the solution around which the neighbourhood is centered.

values. The fourth feature selected as most relevant is the average ratio of neutral moves in the neighbourhood, followed by the average slope of a linear model fit on the sequence of the differences between the best and average solutions in the neighbourhood traversed, and the standard deviation of the number of moves to reach a local optimum, when rescaled by the neighbourhood size. Except for the average slope, which is computed using a FI, all the other features are computed after BI runs.

In general, on our set of problems and instances FTA works well (as good as SA, or even slightly better) on instances whose landscape converges to locally optimal solutions quickly (that is, in less moves than on the instances where SA outperforms FTA), smoothly and regularly (the solution values fit nicely a linear model).

In Figure 4.5 we show the main difference in the landscape that is seen by a best-improvement on two QAP instances. On the x-axis we have the moves accepted and on the y-axis we report the percentage ratio between av-

erage gap in the neighbourhood and the value of the solution around which the neighbourhood is centered. This ratio represents the average effort necessary to accept one random worsening move and deviate from the current BI path. Therefore a horizontal line would indicate that the effort is perfectly constant, and the proper temperature value one should choose is the same for every neighbourhood traversed. In the plot we report ratios obtained along the search for thirty BI algorithms run on two size 100 instances, a structured one (on top) and a random one (on the bottom). The three main differences between the landscapes are immediately clear. First of all, the BIs on the random instance converge much faster than on the structured instance. The structured instance exhibits a high difference between the initial stages of the search and the final stages, where the search stops in a local optimum. Such an increase is not surprising, since for an average solution we can expect half of its neighbours to be better than it; as the search continues, the BI algorithm will end up in a path that is difficult to escape,[2] until it terminates in a local optimum. The third characteristic is a high variability of these paths, that indicate a rough landscape. The curves for the random instance, instead, have a flatter profile, which denotes a higher similarity in the neighbourhoods of both average and high quality solutions, and exhibits a lower variability, an indication of a smoother landscape.

4.6.3 Quadratic assignment problem

For the QAP we have one set of random instances where FTA and SA perform similarly, and a set of structured instances where SA clearly outperforms FTA. Each subset of instances includes three different instance sizes, and for each instance size the results are rather homogeneous.

As in the general case, the average number of moves to converge to a local optimum (rescaled by instance size, feature 3 for BI and FI) appears, along with the difference between the R^2 error of the linear and exponential models on the original solution values (feature 14, for FI), and of the normalized ones (feature 15 for BI and FI). Feature 2, the standard deviation of the number of moves to reach a local optimum is selected nine times. Other related features appear seven times (the average R^2 errors for linear and exponential models, and the standard deviation of the number of moves to converge to a local optimum rescaled by instance and neighbourhood size), and other features less frequently.

On our QAP instances FTA works well on the random instances, whose landscape matches the description given in the general case. SA outper-

[2] More precisely, that it would be more difficult to escape, if the algorithm had this capability.

forms FTA on the structured instances, whose landscape changes shape as the solution quality increases (from steeper to flatter), and using different temperature values has an advantage over the fixed settings of FTA. On the structured instances it is very easy for any algorithm to improve over average solutions. However, at a certain point during the solution traversal towards good quality solutions, the landscape will change, and temperature-based diversification will allow to continue improving whereas a FI or BI stops. A good FTA for this sort of landscape has its temperature set to a value that allows to continue the progress beyond the possibility of a FI. However, as the change in the landscape continues, also this temperature value will eventually become ineffective. On the other hand, higher temperature values, which would be more suitable for the latter stages of the search, would also allow a greater diversification in the beginning, rendering the FTA unable to even converge to the solutions found by the FI.

4.6.4 Permutation flowshop problem, makespan

For the PFSP problem we consider the instances for which we have a corresponding number of jobs in the training set. This ensures that the fixed temperature value chosen is properly chosen, given that the higher diversity of the instances in the Taillard benchmark makes with respect to the QAP scenario.[3] We also take the same number (80) of instances of the Watson benchmark, 20 for each size available.

The following features always appear: the standard deviation of the number of moves to reach a local optimum, rescaled by instance and neighbourhood size (features 2 and 4 for BI), the average number of moves to reach a local optimum and the average ratio of neutral moves in a neighbourhood (features 1 and 12, both FI). The average number of moves to reach a local optimum, rescaled by instance and neighbourhood size, appears nine times, alongside with the average ratio of neutral moves in a neighbourhood, and the average slope of gaps between the best and the average solution value in the neighbourhood (features 1, 2, 5, 12 and 16, all BI).

Our PFSP instances form a more fragmented set than the QAP instances, so in this case it is more difficult to pinpoint a clear distinction between instance groups. It appears, however, that the instances on which FTA is able to find solutions of quality similar to SA, are those where BI and FI are able to converge in a shorter amount of moves, and with a larger proportion of neutral moves in the neighbourhoods. On instances for which BI and FI take a longer time to converge to local optima, or where there is a

[3] However, we observed little difference in the analysis when considering the whole instance set.

lower number or neutral moves in the solution neighbourhoods, SA is still the better choice.

4.6.5 Permutation flowshop problem, total completion time

For the TCT objective we consider the same set of instances used in the makespan case. In this case we have a set of eight features always selected with a consistent order of importance, with only a handful of other features appearing less often. The most important features are the difference between the R^2 errors of a linear and an exponential model (feature 14 for both FI and BI – again for the latter also with normalized solution values, feature 15), the average ratio of neutral moves in a neighbourood as seen by BI (feature 12), the average number of moves to converge to a local optimum (also when rescaled by instance or neighbourhood size, features 1 and 3, for BI), the average R^2 error of a linear model fit on the sequence of solution values (feature 7, BI), and the standard deviation of the standard deviations of the sequence of gaps between best and average solutions in the neighbourhood traversed (feature 19, BI).

In this case, FTA performs well on instances where BI or FI can converge regularly towards local optima, and in a shorter amount of moves. SA instead performs better than FTA when the landscape around average quality solutions differs from the landscape observed in high quality areas of the search space. From this perspective, the TCT objective function generates landscapes that are more similar to those generated by the QAP than to the Makespan objective function.

4.6.6 Discussion

In this section we have analyzed the landscapes generated by the instances in several different scenarios, and we have been able to relate the characteristics of the landscape to the results obtained by the algorithms we considered. We have therefore determined the conditions necessary for understanding which algorithm between FTA and SA can perform better on a given instance.

The diversification mechanism introduced by the Metropolis acceptance criterion works by allowing to accept worsening moves with a probability that is dependent on both the relative worsening of the move with respect to the incumbent solution and the temperature parameter. Therefore, the temperature enables the transition from the incumbent solution towards a portion of its neighbourhood that increases along with the temperature. In particular, the optimal fixed temperature value would be the one that allows to reach the optimal solution; in practice, we search for a temperature value

that obtains the best possible solutions. For this condition to be verified, the same temperature value has to be optimal, or close to optimal, in every neighbourhood traversed. This can happen only if the neighbourhood structure is the same across all the solution space, in the sense that the distribution of differences between neighbouring solutions must be the same for all the solutions. This is indeed the "ideal" case considered in several theoretic works. The more the actual distributions of differences look the same, the better a FTA will perform. In this case, it is therefore possible to determine an optimal temperature value. Lower values for the temperature enforce stricter acceptance conditions, making the search stop in local optima, or suboptimal regions of the solution space. Higher values, instead, will accept too many worsening moves, making the algorithm unlikely to converge towards good solutions.

When, instead, different neighbourhoods will have different distributions of differences, there is no single temperature value that can allow the same optimal traversal throughout the entire search space: values good for certain neighbourhoods will be either too strict or too high for other neighbourhoods. In this case, the adaptive capability of SA will make it possible to eventually find good temperature values in different areas of the search space. This does not mean that SA is able to determine a good temperature value for the incumbent neighbourhood, but rather that there is a good chance that at a certain point the variation of temperature in SA will end up at a value that allows progress, until either the neighbourhood structures or the temperature change.

We have thus offered an alternative perspective to the theoretical studies reported in Section 4.2. We emphasize that, differently to many of the existing studies on this subject, our approach explicitely relates algorithmic characteristics with instance characteristics and the performance obtaines, on a plurality of problems and instance classes. Contrarily to those studies, moreover, we did not focus on the probability of finding an optimal solution, but on the effective performance under realistic conditions, thus providing an indication about which algorithm to use in practice.

4.7 Conclusions

Simulated annealing and its variants are a popular and effective class of stochastic local search algorithms. They are widely used in practice, but there is not yet a complete characterization of their behaviour. One of the long standing open questions in the theory of stochastic local search algorithms has been to determine whether it is possible to design a fixed-temperature

variant of SA that can outperform its original counterpart. This question has been addressed in several works, but no definitive answer has been provided so far. Researchers have, in fact, for the most part focused on single problems, and on one or very few instances, trying to model the ideal convergence behaviour of the algorithm. However, no work so far has included specific characteristics of the instances in the modeling, essentially detaching the algorithm from the conditions it operates in, and this is the reason for the variety of outcomes of the different analyses.

In this chapter, we have instead followed an experimental approach, using configuration tools to automatically instantiate the best possible algorithms for a variety of scenarios, and relating them to the solution landscapes traversed. This way, we have been able to answer the question whether "to cool or not" by identifying the conditions that have to be present in order for the fixed temperature variant of SA to obtain solutions as good as or even better than SA. The answer to the question is therefore "it depends", being dependent on the landscape generated by the instance. FTA works well for instances whose distribution of differences between neighbouring solutions is approximately constant. If, instead, neighbourhoods have a different structure in different areas of the search space, for example they differ for average quality solutions and good quality ones, SA is able to find better solutions.

Beyond the particular research question addressed in this work, we have shown how to combine automatic algorithm configuration and design techniques with problem-independent landscape analysis to infer knowledge about the general behaviour of stochastic local search algorithms.

This work can be extended in several ways. One direction is to include additional problem-independent features in the analysis, and determine the minimal set necessary to make per-instance algorithm selection and configuration feasible at runtime. Likewise, including different problems and instance classes will make it possible to expand our analysis and observe the behaviour of FTA and SA under different conditions. Finally, the same approach can be applied to other classes of algorithms, both metaheuristics such as genetic algorithms or exact methods like MIP solvers, to gain further understanding about how they work. The final goal of this direction is to move away from predefined algoritmic structures to be able to automatically instantiate algorithms whose components and parameters are tailored to a specific problem and instance.

A causal framework for optimization algorithms 5

Despite the countless algorithms available in the literature and the many approaches that study them, understanding the behaviour of an optimization algorithm and explaning its results are fundamental open questions in Operations Research and Artificial Intelligence. We argue that the body of literature available is already very rich, and the main obstacle to the advancements towards an answer to those questions is its fragmentation. In this work we propose a causal framework that relates the entities involved in the solution of an optimization problem. We show how this conceptual framework can be used to relate many approaches aimed at understanding how algorithms work, and how it can be used to address open problems. In particular, we use it to show how to make use of data collected in past experiments to automatically infer plausible algorithms for unseen problems, and provide a proof-of-concept using a simple stochastic local search.

5.1 Introduction

In the last decades, plenty of exact and non-exact methods have been proposed to tackle optimization problems. However, if algorithms for polynomially solvable problems are supported by mathematical proofs that both guarantee and explain the results, it is far more challenging to understand how effectively algorithms for NP-hard problems obtain their results. The branch-and-bound algorithm, which is the base of many state-of-the-art

methods in exact optimization, exhibits a chaotic internal behaviour [76, 77]. Hence it is almost impossible to know a priori the performance of a solver on an instance in practice. Theoretical analyses of SLS behaviour have often been limited to specific cases, idealized models, or unrealistic assumptions such as infinite running time. In particular, despite the hundreds of metaheuristic algorithms and the thousands of variants proposed in the literature, understanding how a stochastic local search (SLS) algorithm works in general is still an open research question [13, 14]. Yet, these algorithms are routinely used with successes in countless applications. Thus, the gap between theoretical analyses and practical results is still very large.

In recent years, however, the adoption of rigorous experimental practices led many researchers to develop mathematical and statistical tools to gain insights in algorithmic behaviour [12]. For example, various analyses have tried to determine bounds on either solution quality or runtime of heuristic algorithms in various scenarios, or the optimal parameter values for an algorithm on a certain instance [281, 282, 283]. Alongside the theoretical approaches, the increased availability of computational power made empirical analyses possible. After early examples [218, 219], the increased availability of cheap computational power makes it possible to perform extensive comparison between algorithms and study the impact of particular problem features thanks to careful experimental design and statistical analyses [284]. Efficient algorithms can be instantiated with reduced human efforts thanks to automatic selection and configuration tools [116, 117, 148, 193]. Selection and configuration tasks generate also huge amounts of data, that can be analyzed to gain insights on both the algorithms and the scenarios considered [285, 286, 287]. Fitness landscape analyses study properties of problems and instances, and can be used to explain algorithmic results [121, 288].

In the literature, however, little work has been done to formally connect these related areas of research. The main issue lies in the difficulty of generalizing the many results obtained for specific problems, instances and algorithms to other scenarios. In turn, this difficulty arises from the huge number of options we can choose from, on the stochasticity of the algorithms and instances, and on the computational burden of generating usable information. To understand the behaviour of an algorithm, a practitioner still needs manually inspect the results obtained, and relate them with the information available on the problem and the instance. Moreover, the knowledge acquired both with practice and from the relevant literature is often difficult to apply to the development of a new algorithm or a new application.

In this work, we show how the works in literature coming from these

research areas already contain a huge amount of knowledge and methods to understand how an optimization algorithm works, and to explain its results. We start from simple assumptions about optimization algorithms to develop an unified framework that relates the elements involved in the solution of an optimization problem. We use a causality-based approach, to make use of well-established properties of causal models to show (i) how this general framework represents the relationships between the high-level entities at play when solving a problem, (ii) how several existing approaches considered in various areas of research related to the understanding of optimization algorithms from an empirical perspective, and (iii) how it can be used in practice to bridge the knowledge gap between different problems and algorithms. We argue that the various approaches proposed to understand algorithmic behaviour are inference tasks on our causal framework, essentially offering different perspectives on the same subject.

In particular, using our conceptual framework and the properties of causal models, we show how to make use of available methods and past experiments to perform transfer learning across different optimization problems. We focus on trajectory-based SLS algorithms for combinatorial optimization problems; the framework is however sufficiently general to be also applicable to other heuristic and exact algorithms or continuous problems.

In Section 5.2, we review the preliminary concepts of causal models before outlining our working hypotheses and presenting the causal framework in Section 5.3. In Section 5.4, we review several works approaches aimed at understanding the behaviour of optimization algorithms and explaining their results, and relate them to our causal framework. In Section 5.5 we then show how our framework is useful to guide the development of a practical application, namely transfer learning. Finally, we conclude in Section 5.6 outlining several possible research directions.

5.2 Preliminaries

A causal model is a model that expresses the causality relationships between entities in a system [289]. We can use it to formalize a *theory* we have about the system under study in the form of a graph. Intuitively, each entity considered is a node, and two nodes are directly connected if the value taken by one of the two entities directly affects the value taken by the other one. When changing the value of the first variable, we then expect the value of the second variable to change accordingly. One common assumption is that this relationship is not bidirectional, and therefore only directed edges are used.

Table 5.1: Summary of flow of influence along a path between X and Y: when can X influence Y given evidence about a set of variables $\mathbf{Z}$?

	$W \notin \mathbf{Z}$	$W \in \mathbf{Z}$
$X \to Y$	✓	✓
$X \leftarrow Y$	✓	✓
$X \to W \to Y$	✓	
$X \leftarrow W \leftarrow Y$	✓	
$X \leftarrow W \to Y$	✓	
$X \to W \leftarrow Y$		✓

A causal model can be formally defined as a quadruple $\langle \mathcal{U}, \mathcal{V}, \mathcal{F}, \mathcal{P} \rangle$, where $\mathcal{U}$ is the set of *exogenous* variables whose values are determined by reasons external to the model (e.g., values we choose), $\mathcal{V}$ is the set of *endogenous* variables whose values are determined by a subset of the other variables in the model, $\mathcal{F}$ is a set of functions that we can use to compute the values of the $\mathcal{V}$ variables from the values of their parent variables, and $\mathcal{P}$ specifies a joint probability distribution over $\mathcal{U}$. Each variable can therefore be considered to model our subjective belief or knowledge about it.

The causal structure of the system can therefore be represented as a Directed Acyclic Graph (DAG) whose nodes are $\{\mathcal{U} \bigcup \mathcal{V}\}$ and the $\mathcal{V}$ nodes have incoming arcs from their parent variables. The values each node can take is defined probabilistically, for nodes in $\mathcal{U}$ by $\mathcal{P}$, and for nodes in $\mathcal{V}$ as probabilities conditional on their parent nodes, as defined using $\mathcal{F}$. In other words, a causal model can be described as a Bayesian network where the relationship between the variables is assumed causal – contrarily to classical Bayesian networks where the relationships represent the weaker state of conditional independence.

Causal models are particularly useful because they define how to reason about certain variables in a model, taking into account the other variables. There is in fact a codified set of rules that define how to update the value of an otherwise unobserved variable having observed a set of other variables, depending on how the variables involved are connected. This set of rules answers, case by case, the general question "how does new information about observed variables change our knowledge about the unobserved variables?".

Queries on causal models belong to three categories: inference, intervention, and counterfactual analysis. These three actions are also referred to as the three rungs of the causality ladder, because each one is an abstraction

level more powerful than the previous one: seeing the data, act on the system to obtain new data, imagine how some observations we don't have would be [290].

Inference is the task of observing the values a subset of variables $\mathbf{X}$ take when other variables $\mathbf{Y}$ take certain values. When there are only two variables involved, in a scenario $X \rightarrow Y$ with no other variables connected, it suffices to apply the Bayes theorem for both cases (i) updating Y having observed X, and (ii) updating X having observed Y. For more than two variables the patterns of reasoning can be reduced to cases with three variables, summarized in Table 5.1. For a chain structure $X \rightarrow W \rightarrow Y$ there is a causal path from X to Y if W is not observed. Conversely, when W is observed, that causal path is interrupted, since no matter the state of X, the value of Y depends only on W. The same applies for the inferential path from Y up-stream to X. For a common-effect relationship $X \rightarrow W \leftarrow Y$, or v–structure, X and Y are conditionally independent when W is unobserved, but a causal path opens when W is observed. The common-cause relationship $X \leftarrow W \rightarrow Y$, or fork, is the opposite case, with X and Y being related when W is unobserved, and independent when W is observed. All these relationship can be combined with each other, generalize to the case of more than three variables, and apply to sets of variables.

When performing *intervention* we instead actively modify the values of some variables $\mathbf{Y}$, regardless of whether there are existing causes for them. This is done using the $\texttt{do}(\cdot)$ operator, which performs a removal of the incoming nodes to $\mathbf{Y}$. Thus, $\texttt{do}(Y = y)$ means to ignore the existing causes for Y, arbitrarily assign to it a value y of interest, and observe how this impacts the network downstream. For root-level nodes, it is therefore equivalent to inference.

Finally, *counterfactual analysis* aims to infer the values that some variables $\mathbf{X}$ would have taken, had other variables $\mathbf{Y}$ taken certain values. This level of analysis estimates the outcome of experiments that are not performed, and is particularly useful to provide explanations.

Causal models are particularly important because they are *falsifiable*: if the observations do not match the model, we can conclude the model (and with it, our theory about the system we represented in it) is wrong and it has to be fixed or discarded. They also enable the *integration* of observations from partially differing sources, thus being particularly useful in transferring knowledge across scenarios. Causal models have been increasingly adopted in disciplines such as medicine, biology, psychology, economy, in all cases when the goal is to understand causes and effects in a system.

5.3 A causal framework for stochastic local search algorithms

In this section we define a theory that encodes the hypotheses, and a subsequent causal framework to model the relationships between the entities involved in the solution of an optimization problem.[1] We define high-level entities, such as "algorithm" or "problem", and connect them according to their causal relationships. Later we will see how to instantiate the framework into an effective causal model for a given scenario, or set of scenarios, and the practical limitations we encounter in this instantiation.

5.3.1 Working hypotheses

In this work, our goal is to propose a theory that relates the entities involved in the solution of an optimization problem. In building the theory and thus the framework, we start from the following four working hypotheses.

(H1) An algorithm can be divided into basic components and parameters. We take a component-based view of SLS algorithms [152, 171, 176], that is, we consider an algorithm as a set of basic building blocks, each one possibly coming with a set of parameters, combined together in a certain way.

(H2) Separation between algorithm- and problem-specific components. Following (H1), algorithmic blocks are divided in problem-specific and algorithm-specific. Problem-specific components are the parts of a SLS that require specific knowledge of the problem under study, such as the generation of an initial solution, the perturbation of a solution, or the evaluation of a solution. Algorithm-specific components are all the components that define what a SLS is. They can be used across different unrelated problems, such as the tabu list in a Tabu Search, or the cooling scheme in a Simulated Annealing[2].

(H3) An algorithm operates a traversal of the search space. An SLS works by traversing solutions in a search space; the problem-specific components of a SLS are needed to (i) select one starting point of this traversal, and (ii) evaluate another solution in the search space, relative to the current one.

[1] The framework differs slightly from a previous preliminary version [19], and is the result of further analyses and discussions.

[2] We include in the algorithm-specific components also those components that require problem-specific knowledge, but still operate at the algorithm level, such as the initial temperature defined for a simulated annealing for the PFSP proposed in [214].

(H4) Identification of optimal algorithm behaviour. For obtaining optimal results on a search space, an algorithm needs to reach an optimal tradeoff and alternance of diversification and intensification behaviour.

For any practical application, we can assume that our set of blocks is wide enough to cover all the needs for our case, that is, we can instantiate at least one fully working algorithm for any given problem. This is a reasonable assumption, since in case we miss the components we need to solve a certain problem, we can develop them and add them to our existing set of blocks.

Thus, an SLS is defined by its algorithmic-specific components and their combination, and an instantiation of a SLS for a problem is the combination of the algorithm-specific components with a set of problem-specific components. This separation also means that, if we have the problem-specific components for a problem of interest, we can use them to instantiate several algorithms for that problem, that will differ at the algorithm-specific level. In other words, with (H1) and (H2) we can apply the Programming by Optimization paradigm. We will also use interchangeably the terms "parameters" and "set of components", since an algorithm will be instantiated by choosing a set of blocks that, together, form a valid procedure for the task. Likewise, in this work the term "algorithm" can be considered synonym with "configuration".

Hypothesis (H3) formalizes the intuition that if we assume to have complete knowledge of the entire search space (or to have an oracle that gives us the desired information about a solution when polled), then any optimization problem becomes a search problem, and we do not need additional knowledge on the problem anymore. This is of course a far-fetched assumption when solving a problem instance in practice, but we show that, for the purpose of understanding algorithm behaviour, this lets us bridge the insights we obtain across different problems. The separation of the components of (H2) is an effective simulation of the oracle, because the search part of a SLS is devoted to traverse the search space, using the problem-specific components to retrieve information about the value of each solution considered.

Two different landscapes define, in principle, a different optimal behaviour, but several different algorithm can potentially reach this desired behaviour [16]. Building upon (H1) and (H2), with (H4) we fully realize the PbO paradigm, by assuming we can identify a subset of sufficiently powerful components that can reach this behaviour, regardless of the form (and name) of the resulting algorithm. Often, in practice, we do not have this subset, or we are not able to identify it among the components we have. So

we can relax this hypothesis by assuming we can identify and configure an algorithm that obtains results that are "good enough" for our purpose.

5.3.2 The causal framework

The theory. From the working hypotheses (H1)–(H4), we consider problems and instances as given inputs to tackle, using an algorithm built starting from the basic components; the efficiency of the algorithm depends on the computational environment such as machine power or running time. These factors are the *causes* for the results obtained, which therefore are the *effects*. What relates the inputs to the results, is how (efficiently) the algorithm can traverse the search space, that is, how it can efficiently oscillate between diversification and intensification, based on the characteristics of the instance. These characteristics can be represented by the features.

This theory is rather straightforward and many if not all the works cited in this paper implicitely already assume this model. However, by making it explicit, we construct a falsifiable theory that we can follow in our goal of explaining how and why an algorithm works.

The general causal framework representing the theory is shown in Figure 5.1. In the general framework each node is actually a macro-node representing a set of nodes, which are grouped together by their function.

High-level entities. P represents the variables relative to the problems, such as the objective function. D is the data necessary to instantiate one specific occurrence of a problem, e.g., the instance file. While it may be argued that an instance exists for a given problem (e.g. a TSP instance), we consider them separately to emphasize the separation between the generic problem definition and its specific realizations. Following (H2), PA is the set of problem-specific components and algorithms (e.g. initial solutions, neighbourhoods, heuristics), and SA is the set of algorithm-specific components that compose the search part of the algorithm. Components in PA and SA include both algorithmic functions such as neighbourhoods or acceptance criteria, but also numerical parameters.

C represents the set of computational factors that impact an actual implementation of any algorithm instantiated, such as the computational power, running time, random seed, but also quality of the implementation, and any other possible factor affecting the computation.

A is the set of algorithms that we can use for solving an instance $i_p \in D$ of problem $p \in P$; differently from PA and SA, an algorithm $a \in A$ is an instantiated and fully configured algorithm. The separation of A from its components collected in PA and SA follows from (H1).

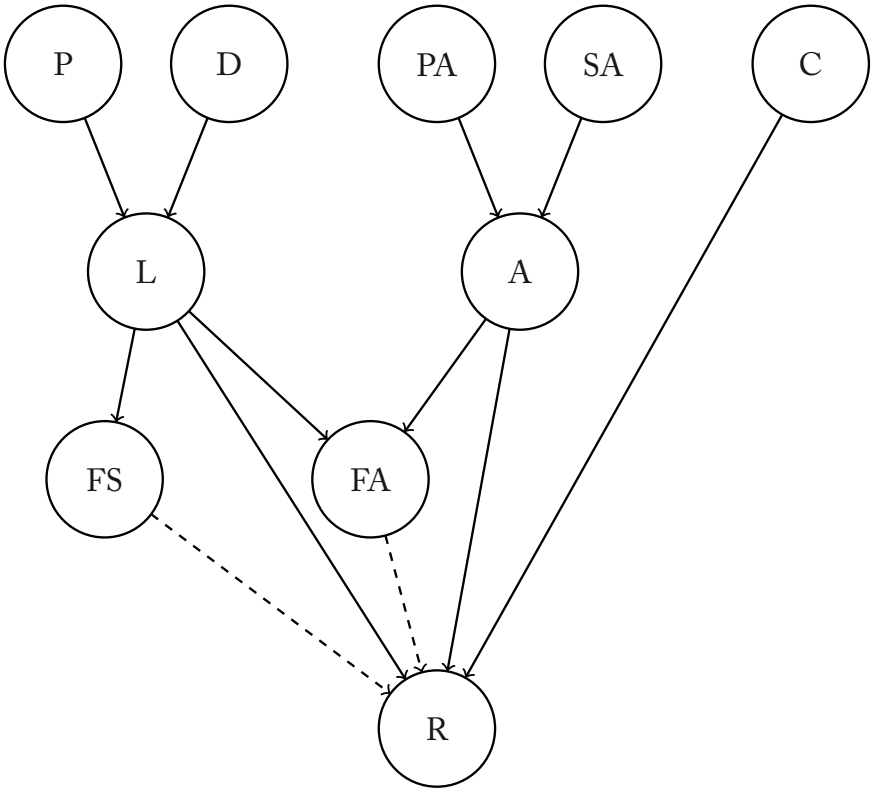

FIGURE 5.1: A generic causal framework representing the interaction between problems, instances, algorithms and results. Each node in the DAG is a macro-node modeling a set of entities: P for the problem, D for the data, PA and SA for respectively the problem-specific and algorithm-specific components, A for the search algorithm, C for the computational environment, L for the solution landscape, FS for the features of the instance, FA for the features that represent the landscape as seen by a search algorithm, and R for the results. Arcs represent the causal relationships between the high-level nodes. In practical applications, not all the nodes might be observed, and not all the nodes might be present, depending on the specific instantiation of the problem under study.

L is the abstraction of the landscape that is generated when a problem in P is instantiated with data from D. For any practical purposes, we can consider the landscape as not directly observed, but we can observe some characteristics that represent it. We classify these features in FS (static), features that arise from the instantiation of a problem, and in FA, features that represent the landscape as seen by an algorithm in A. For example, FS contains the set of instance features, such as the instance size, the problem-specific features, such as angle features for the Travelling Salesman Problem, or properties of the solution landscape such as the ruggedness. FA instead contains features observed with some search algorithm, such as the number of iterations a Hill Climbing takes to converge to a local optimum.

Finally, R models the results that can be obtained by an algorithm $a \in A$ on an instance i_p for a problem $p \in P$, under the computational environment specified in C.

Causal relationships. $\{I, P, \text{PA}, \text{SA}, C\}$ form the set of exogenous variables, because they are the ones we set when we tackle an optimization problem; $\{L, A, \text{FS}, \text{FA}, R\}$ are the endogenous variables, whose value depends on other variables, either deterministically or in a probabilistic way.

An algorithm in A can be instantiated combining components from PA and SA, according to (H1) and (H2), hence we have arcs from PA and SA to A. The relationship between PA, SA and A is entirely deterministic: an algorithm is either composed by a certain set of components or not. We remark that the PA and SA components used to compose the algorithms in A are not necessarily the same ones used to compute the landscape features in L. When it is not possible to decompose an algorithm $a \in A$ into basic building blocks from PA and SA, because we do not have or recognize them, we treat it as an intervention operation: we assume it is composed by unobservable building blocks and we manually instantiate it as $\texttt{do}(A = a)$.

The landscape L is defined solely by the instantiation of a problem, so we have incoming arcs from P and D.

The features in FS arise only from the contribution of L, which mediates P and D. The features in FA are instead observed when an algorithm in A traverses the landscape L. Note, however, that the features generated using an algorithm in E are considered mere properties of the landscape, and not an evaluation of the algorithm, which is a different task. It is also important to remark that as FS and FA represent properties, at this stage we do not impose any limit (from C) on our ability of observing them; clearly this cannot always happen in practice and we might need to settle for some approximation.

Finally, the results R for an instantiation of a problem as defined in P and D depend on the performance of an algorithm in A, under the computational environment represented in C; however, following our working hypothesis (H_3), once an algorithm from A has been instantiated, it operates on a search space here represented by L and can be considered, at that point, unaware of the specific problem that generated it.

Despite L already containing (at the conceptual level) all the information about the problem and the instance, we include additional edges from FS and FA to R (dashed in Figure 5.1). We include these edges because FS and FA provide an observable intermediate (mediator) level of "reasons" that can help us understand the performance R of A for P, D given C, a level that we can use, conversely to the (usually unobservable) landscape L.

For the moment we do not assume any inter-node connection, that is, the nodes composing the high-level entities are not directly connected to each other, since they are defined either as exogenous variables, or as determined by other entities. This simplification may be not fully true, for example for nodes in PA and SA and their connections with A, where components can be combined in some hierarchical fashion [80], or the relationships between algorithm parameters can be represented using another DAG [146, 291], but for the sake of simplicity we choose to not investigate further this issue and to demand it to future work.

From this generic framework we can instantiate an effective causal model, depending on the specific case. For example, in the PbO paradigm, A will be the set of components in which we can decompose an algorithm. In the context of algorithm selection, instead, A is a variable taking values $a_1, a_2, \ldots, a_n$ for the n algorithms in the portfolio. Likewise, P contains at least the objective function, but if necessary it is possible to represent the constraints.

One of the advantages of the general framework is to abstract the connections between high level concepts, being therefore valid across several scenarios (in particular, problems). An instantiation of the framework for a specific problem may contain elements that do not apply to other problems (e.g. the number of clauses in SAT). It is anyway possible to select a tradeoff between generality and specificity and operate at the desired level of precision. Another consequence of this is that we do not need to worry about being too precise in the classification of the components and features, since their relationship with the other entities of the framework is far more important.

Notation and reasoning on the framework. The notation we use for this framework is somewhat an abuse of the notation used in causal models, and we use it to model the high-level interactions. For two high-level nodes X and Y, the$Y|X$ indicates the choice of Y depending on X. For example, we have that $A|$PA, SA, because the algorithms in A are generated starting from a specific instantiation of the components and parameters in PA and SA.

Causal and inferential reasoning is as in any causal model. Since there is no edge inside each macro-node, all the flows of information we define on the framework apply also when we "open" the macro-nodes into their constituting nodes.

Limitations of the framework. There are several difficulties we encounter when constructing and instantiating a specific model from the high-level framework. The first one lies in the amount of choices we need to make to properly define the model variables. How to represent the results? Do we consider absolute solution values, or relative deviations from optimal or best known solutions for the instances considered? In the latter case we can use best known values as bounds, but how do we choose to compute them? How to represent the problem or the constraints, when present? Which features to include, and how to represent them numerically? Many features will be correlated, so some feature engineering can be useful: how and to which extent can we do it? How to represent in a meaningful way the computational environment in C? Some of these questions have obvious answers when implementing an algorithm to perform experiments and collect data, but require reflection when integrating the relative information in a single model instantiation.

The next issue is related to the computational task defined by a model. A full model will contain several variables, and in particular for nodes in A and R, with many parent nodes the resulting conditional probability tables will be huge. One common assumption in many practical causal models e.g. in biology is that continuous variables appear only as terminal nodes, and causes and intermediate nodes are discrete. In this context, this assumption does not hold, because our framework can have continuous causes, for example continuous parameters.

The third issue lies in the difficulty of fully instantiating a model; aside from the entities that may not be considered in some of the tasks, single tasks or studies usually ignore or underestimate some of the variables involved. For example, algorithm comparisons often restrict the analysis or the experiments to a fixed set of design and parameter choices. This makes

it impossible to generalize the findings, since, for example, a proper configuration can improve the algorithms and subvert the results [16].

Nonetheless, in the next section we see how working under this conceptual framework is useful to relate existing topics and works from the literature, and how the theory behind the framework provides a support to existing analysis approaches.

5.4 Some examples from the literature

In this section, we revise research lines aimed at understanding the behaviour of optimization algorithms, and relate them to our framework, showing how they can be represented as inference tasks on the framework. The goal of this section is to show how the framework can be used to define the research questions, highlighting the different perspectives and elements considered in each work. While a complete literature review is beyond the scope of this work, we choose to select some papers to cover a wide range of methods and applications.

5.4.1 Theoretical and experimental analysis of algorithms

The analysis of the results obtained by optimization algorithms is the most common kind of analysis for SLS algorithms and it appears in the literature in countless different flavours. Theoretical works aim to establish statements that are certain and true given a certain set of assumptions. However, concessions usually have to be made in the assumptions to be able to derive general statements. Nonetheless, several sophisticated mathematical tools have been developed to analyze the behaviour of an algorithm, from schema theory for genetic algorithms to methods based on Markov chains to statistical analyses [88, 292, 293, 294]. Experimental analyses are instead the only viable choice when moving from ideal assumptions to problems meant to model real-life applications or from basic algorithmic templates to complex ones. Over the years, the research community has developed best practices and guidelines about performing computational experiments [284, 295, 296, 297, 298]. Theory and experiments offer therefore complementary views on the same research questions. Under our framework, we can focus on the specific question asked in each work, and consider theory and experiments as tools to provide answers.

Historically, the theoretical analysis of SLS has focused on convergence results, to show how a particular algorithm could guarantee the discovery of an optimal solution under a set of assumptions. This means to prove that,

for an algorithm $a \in A$,

$$R = r^* | A = a, P = p, D = d, C = c, \tag{5.1}$$

where r^* is one optimal solution, for some assumptions encoded in p, d, c. Notable examples are the convergence analyses of cooling schemes in the simulated annealing literature [203, 299, 300], where it is shown how some cooling schemes are guaranteed to find an optimal solution under the assumption than infinite runtime is available to solve an instance sampled from an uniformly random distribution. Noting that, in this case, the cooling scheme is one of the components of simulated annealing we can use (H1) and (H2) to describe the goal of these works as proving[3]

$$\lim_{\text{runtime} \in c \to \infty} Prob(R = r^* | \text{SA} = sa, \text{PA} = pa, P = p, D = d, C = c) = 1, \tag{5.2}$$

that is, given infinite runtime, the probability that an algorithm composed with a selected set of pa, sa that include the specific cooling scheme considered finds the optimal solution r^* for an instance of p, d approaches 1. These analyses have since then developed also for other algorithms. For example, Gutjahr studied the convergence to the optimal solution of ant colony optimization algorithms [301, 302]. Rudolph discussed conditions that allow a non-elitist $(1, \lambda)$-ES to converge to the optimal solution in infinite time [303]. He also proved that a canonical genetic algorithm will never converge to the optimal solution for any combinatorial optimization problem, but its variant that keeps the best solution after selection can, in infinite time [304]. Reviews of this area can be found in [294, 305, 306, 307].

On the other hand, runtime analyses aim to determine (bounds on) the runtime that an algorithm $a \in A$ takes to find the optimal solution of an instance $d \in D$ of a certain problem $p \in P$. Hence, the problem can be framed as

$$\text{runtime} \in C | A = a, P = p, D = d. \tag{5.3}$$

Examples from the theoretical community include analyses of generalized hill climbing (another name for the fixed temperature variant of simulated annealing) [205], or ant colony optimization variants [308, 309].

Works such as [203, 227, 268, 269], instead, compare alternative cooling strategies, notably the traditional geometrical cooling scheme and the fixed temperature one. A statement such as "algorithm $a_1 \in A$ is superior to algorithm $a_2 \in A$" (or, alternatively, a component sa_1 is superior to another

[3] We use $\text{Prob}(x)$ to denote the probability of x, to avoid possible confusion with the variable for the problem in our framework.

component sa_2) can be interpreted in several ways, even when considering one specific scenario. This can make it difficult to relate the findings of two different works, because of the different perspectives. Our causal framework can help disambiguate the precise question asked. For example, we could mean that a_1 converges faster than a_2 to an optimal solution, thus having a comparative runtime analysis

$$\begin{aligned} &\text{runtime}_1 \in C_1 | A = a_1, R = r^*, P = p, D = d < \\ &\text{runtime}_2 \in C_2 | A = a_2, R = r^*, P = p, D = d \end{aligned} \tag{5.4}$$

where C_1 and C_2 are the respective experimental conditions, including the runtime, under which a_1 and a_2 can discover an optimal solution r^* of an instance of a problem p and data d. Alternatively, the statement could be understood as that under the same conditions, a_1 is expected to find a better solution than a_2, which can be formalized as

$$\begin{aligned} &Prob(r_1 > r_2 | P = p, D = d, C = c) > \\ &Prob(r_2 > r_1 | P = p, D = d, C = c) \end{aligned} \tag{5.5}$$

where r_1 is the solution found by a_1 for an instance of P and D under constraints C, r_2 the solution found by a_2 in the same scenario, and the "$>$" sign in this case indicates "better quality". We could also mean that the probability of a_1 finding an optimal solution is higher than the probability for a_2, as in

$$\begin{aligned} &Prob(r = r^* | A = a_1, P = p, D = d, C = c) > \\ &Prob(r = r^* | A = a_2, P = p, D = d, C = c) \end{aligned} \tag{5.6}$$

for the same instance of P and D and under the same constraints C. Finally, another different interpretation is that a_1 can operate under more relaxed assumptions than a_2, that is

$$\begin{aligned} &\exists\, p \in P, d \in D, c \in C\ : \\ &Prob(R = r | A = a_1, p, d, c) > 0, \\ &Prob(R = r | A = a_2, p, d, c) = 0 \end{aligned} \tag{5.7}$$

that is, a_1 can potentially reach a target solution r under assumptions on the problem, the data distribution or the experimental conditions that prevent a_2 from doing so. Again, our hypotheses (H1) and (H2) make it possible to apply Equations 5.3 to 5.7 both at the algorithm and the component level.

In the experimental community, the most immediate example of analysis of algorithms is probably the comparison between metaheuristics on a certain problem. For instance, many works have compared simulated annealing

and tabu search on different problems such as the QAP [215, 254], facility location [310] or constraint solving [311], or the many heuristic algorithms proposed for the flowshop problem [57, 312]. Usually this corresponds to the formulation of Equation 5.5, choosing the algorithm $a \in A$ that gives the best results for a problem and a set of instances, under a certain computational environment

$$\arg\max_{a \in A} Prob(R|P, D, a, C). \tag{5.8}$$

Some works have studied in detail the impact of each component of an algorithm, such as simulated annealing [16, 218, 219], tabu search [313] and ant colony optimization variants [105, 314]. An a priori understanding on the functioning of some component in SA or PA can be an essential part in the explanation of the results in addition to information at the mediator level, and the only way of explaining the results when no features are computed. For example, the late acceptance hill climbing cannot accept a solution that is worse than anything the algorithm has encountered in the past. Thus, it has a limited diversification potential and cannot accept arbitrarily bad solutions, contrarily to the Metropolis criterion [17, 211]. These insights at the component level make it possible to understand the behaviour of the entire algorithm. Subsequent analyses have then focused on the particular impact of several options for one specific component or parameter of an algorithm, such as the tabu list in tabu search [315], the acceptance criterion in large neighbourhood search [316] and simulated annealing [17], or the initial population strategies [317] and crossover operators [318] in genetic algorithms. Other studies have instead evaluated the results obtained by algorithms of different complexity, such as iterated local search algorithms on flowshop scheduling problems [81], and evolutionary algorithms on knapsack, Ising model and MaxSAT [319]. From the perspective of this work, all these works aim to answer the same questions formulated in Equations 5.3 to 5.7, either at the A or at the SA level, simply using different tools.

On top of the different angles explored in the various works, theoretical analyses have for the most part focused on one specific problem and one particular instance distribution. This is another explanation for which the results can be contrasting. In [227], Hajek and Sasaki proved that there is no monotonically decreasing cooling scheme that is optimal for specific instance of the matching problem. In our framework, we have $p = \mathsf{MATCHING} \in P$ and a particular $d \in D$ that includes a specifically constructed distribution and a worst-case initial solution. The authors consider sa_d and $\text{sa}_{nd} \in$ SA, respectively a decreasing and non-decreasing cooling scheme defined

as components of the search part of the algorithm[4] that obtain results r_d, $r_{nd} \in R$. The particular implementation of the decreasing cooling scheme is not important in this work. In fact, Hajek and Sasaki prove that it exists a non-decreasing cooling scheme sa_{nd} such that for every monotonically decreasing cooling scheme sa_d, and using the Metropolis acceptance [198] to evaluate a randomly selected candidate solution (also in SA), we have $r_{nd} > r_d$ and for average running time in C polynomial in the size of the instance.

Wegener proved instead that on natural instances of the Minimum Spanning Tree, simulated annealing, defined as one specific decreasing cooling scheme $\text{sa}_d \in \text{SA}$, outperforms sa_{nd}, in that work called the Metropolis algorithm [268]. That is, $Prob(r_d = r^*) > Prob(r_{nd} = r^*)$ within a bounded number of steps. Later Meer proves the same result for a specifically constructed TSP instance [269].

These results can, of course, be easily reconciliated by noting that the results depend on the specific problems and problem instances, which generate different landscapes. Thus, the results have to be explained by using some variables at the mediator level, as a surrogate for the unobservable landscape:

$$R|P, D, \text{SA}, \text{PA}, \text{FA}, \text{FS}, C. \tag{5.9}$$

Here we also mention a parallel and complementary line of research to the quantitative analyses we focus on in this chapter, with recent works such as [178, 179, 320, 321] that qualitatively study and compare algorithms to show functional similarities and differences between them. This trend arose as a reaction to the proliferation of metaphor-based metaheuristics that did not include any actual algorithmic novelty [182]. The first mathematical proof of the equivalence of a supposedly novel metaheuristic came from a discussion between Weyland and the authors of the harmony search, that was demonstrated to be perfectly equivalent to, and its performance always bounded by, the $(\mu + 1)$ evolutionary strategy [178, 190]. Camacho, Dorigo and Stützle first showed that the intelligent water drops algorithm is a special case of ant colony optimization [179, 320], and subsequently that grey wolf, firefly and bat algorithms are variants of particle swarm optimization [322] In these works it is essentially shown how $\text{sa}_1 \in \text{SA} \equiv \text{sa}_2 \in \text{SA}$, and therefore $Prob(R|P, D, \text{sa}_1, C) = Prob(R|P, D, \text{sa}_2, C)$.

Qualitative analyses have been applied also to the experiments reported in some works. In notable cases such as the harmony search proposed to

[4] Hajek and Sasaki reason in terms of *temperature schedule*, that is, sequences of temperature values. However, in practice we construct a function that computes such values from the instance.

solve the sudoku [178] or a modification of the Clarke-Wright heuristic for the capacitated vehicle routing problem [321], it is proved both theoretically and experimentally that

$$Prob(R|P, D, A, C) = 0, \tag{5.10}$$

that is, the original papers report results that are impossible to obtain. This is therefore an indication of possible mistakes or misconducts.

5.4.2 Instance and landscape analysis

As we mentioned before, understanding the conditions encountered by an algorithm during its search is a necessary step to ultimately understand how an algorithm works.

It is very common for an algorithm that performs well on a certain set of problem instances to perform poorly on a set with different characteristics, and possibly viceversa. In practice, it is commonly observed that some problem instances and some problem classes are intrinsically harder than other ones. In our causal framework, we can first of all disambiguate what it is meant with "difficult". One possibility is that for two data distributions $d_1, d_2 \in D$ of a problem $p \in P$, a randomly chosen algorithm $a \in A$ is expected to obtain better results on the landscapes generated by (p, d_1) than on those generated by (p, d_2), for some computational constraints $c \in C$. Or, alternatively, for $a \in A$ to obtain results $r_1, r_2 \in R$ of the same quality on $d_1, d_2 \in D$ the experimental conditions $c_2 \in C$ on d_2 need to be more relaxed (e.g. a longer runtime) than $c_1 \in C$ for d_1. Still another possibility is that, given $d_1, d_2 \in D$, the set of algorithms $a_1 \in A$ that can obtain a target solution quality $r \in R$ for d_1 is greater than the set of algorithms $a_2 \in A$ that can obtain the same solution quality $r \in R$ for d_2, under the same constraints in $c \in C$. In all these cases, we normally consider d_2 to be a more difficult scenario than d_1.

One approach to understand the conditions for an instance to be "difficult" is to figure out some characteristics that the instance should have, in order to make it harder for a search algorithm to converge to the optimal solution. In our framework, this means to distinguish the results in R by conditioning them on some characteristics $d \in D$. For example, for the TSP when the ratio of the number of clusters to the number of cities is between 0.1 and 0.11 the Lin-Kernighan algorithm takes nearly six times longer to converge to a "good" solution than on a randomly generated instance [323]. Those values are said to yield a *phase transition* between practically "easy" and "difficult" instances [324]. Instances can be generated "difficult" for an algo-

rithm, to study the relationship between instance properties and algorithm performance [325].

To have a better chance of explaining the results, we can move to the landscape and features level. This is equivalent to expanding the relationship between P, D and R to include also a selection of variables in FA and FS to represent the landscape L, thus having

$$R|P, D, \text{FA}, \text{SA}, C. \tag{5.11}$$

In many cases we can, however, omit the experimental conditions C. Some properties of the fitness landscape have been related to the difficulty of a problem instance. Several measures such as ruggedness, autocorrelation, fitness distance correlation, are used as proxies for higher-level properties of the search space like convexity or multimodality [270, 288, 326, 327].

Local Optima Network analyses study the relationships between solutions entailed by the neighbourhood function, observing local optima and the paths to escape from them [328, 329]. This kind of analysis aims at understanding global properties of the landscape L by computing some of its features

$$L|P, D, \text{FA}. \tag{5.12}$$

In case of complete exploration of the search space, we are forced to not impose any computational constraint or limitation.

Fitness Landscape Analysis can also be used for automatic selection and configuration of algorithms [121, 167, 261, 330]. In [331] the landscape is translated into a set of constraints, to understand how complex a local search needs to be on different landscapes.

A smaller number of works have combined algorithmic analysis with problem and instance features. The impact of search space features for the Job Shop Problem on the performance of tabu search was explored in [332]. In [333] six metaheuristics form the portfolio for an algorithm selection problem for the PFSP based on both problem and landscape features. Three different objective functions are instead considered in [18], where a cross-problem analysis identifies the landscape conditions necessary for two variants of simulated annealing to perform well.

Some other works have instead applied problem-specific knowledge knowledge to the design of algorithms. In [334] high quality solutions of Vehicle Routing Problems are analyzed to understand what makes them good, in order to then develop a Guided Local Search that is biased towards solutions that "look like" the good ones [335]. Optimal or high quality CVRP solutions have compact routes and "narrow", with few intersections and short edges that connect to first and last nodes of the routes to the depot. A

knowledge-based algorithm can therefore penalize solutions that have, for example, overlapping routes, or nodes in the route that are spread apart. In [336] an analysis of efficient schedules for small instances of the no-idle Permutation Flow Shop Problem lead to the definition of *superjobs*, that is, subsequences of jobs likely to appear in high-quality solutions, that are treated as a single job to reduce and smoothen the search space. Two algorithms based on Iterated Greedy are then augmented with a learning subprocedure aimed at discovering these superjobs.

5.4.3 Modeling the algorithm behaviour

For further analysis and applications, it is necessary to represent in a mathematical form the behaviour of an algorithm.

To completely separate the general algorithmic template from its implementation for a problem, it is often assumed that the algorithm has no access to any information about the instance, and therefore the algorithms treats the problem as a black-box function [337, 338, 339]. This corresponds to ignoring any prior observation of FA, FS or L, and to learn the distribution of some target variables, being in R, L, in A, or at the component level, during the progress of the search. The use of artificial problems, whose characteristics are perfectly known in advance, make it possible also to evaluate how close the behaviour of the algorithm in the black-box scenario gets to the optimal one, computed analytically, and in some simple cases to generate optimal algorithms [340, 341].

Surrogate models can model knowledge to approximate the outcome of experiments [342, 343, 344, 345]. A surrogate model can be considered an alternative model $(P, D, \mathrm{PA}, \mathrm{SA}, C, A, L', \mathrm{FS}', \mathrm{FA}', R')$ that approximates the real model $(P, D, \mathrm{PA}, \mathrm{SA}, C, A, L, \mathrm{FS}, \mathrm{FA}, R)$. This is useful not only in case of expensive objective functions, but also to filter candidate solutions and guide the search by focusing on the most promising ones [346, 347, 348]. A notable case is algorithm configuration, where the expensive evaluation of an algorithm is replaced by a query on an Empirical Performance Model (EPM) that predicts the solution quality of a configuration on an instance, and only the most promising configurations are actually evaluated [138, 140, 349].

The data collected in the selection and configuration tasks, either on actual experiments or using surrogate models and EPM, is particularly useful to analyze the performance of algorithms. For example, several techniques can be used to estimate the importance of features or algorithmic components and parameters. Determining the most important feature or subset of features means to find the subset $\mathrm{FA}', \mathrm{FS}' \subseteq$ FA,FS such that the propor-

tion of influence mediated from P, D to R remains approximately the same than using FA,FS. Finding instead the most important parameter means to estimate the subset $\mathrm{PA}', \mathrm{SA}' \subseteq \mathrm{PA}, \mathrm{SA}$ that has the greatest impact on the results, that is,

$$\underset{\mathrm{PA}', \mathrm{SA}' \subseteq \mathrm{PA}, \mathrm{SA}}{\arg\max} \quad \Delta R | P, D, \mathrm{do}(\mathrm{PA}'), \mathrm{do}(\mathrm{SA}'), C \tag{5.13}$$

with a slight abuse of the $\mathrm{do}(\cdot)$ notation to indicate a generic intervention on the variable rather than a precise value assignment. These are again different questions, but they share the innately causal goal of estimating the effect of an intervention. Hence, the following methods can be applied to both tasks.

Functional ANOVA [350] is one such approach to quantify the importance of the variables of a function and their interactions according to the proportion of variance explained. One possible alternative is to apply forward selection to the features observed in the EPM [285]. Random forests [244] and surrogate models based on them also provide a native way of estimating the variable importance, that has been used to compute importance of parameters [16, 23, 286, 351]. Ablation analysis [352] is another technique proposed to determine the subset of parameters having the highest impact on the results. It is a path relinking procedure that connects two configurations flipping one parameter a time, choosing the one that yields the maximum gain (or the minimum loss) from the previous configuration. Data obtained in the configuration phase can be analyzed also to find similarities in the final configurations [353], to infer what makes a configuration perform well on a scenario.

The multilevel regression proposed in [287] uses a two-level regression to relate the algorithms components with the problem characteristics. In the first level, the algorithm components and parameters are used to predict the objective function value; in the second level, the problem features are used to predict the coefficients of the first regression. This is equivalent to performing a series of tasks, the first one being

$$R | \mathrm{PA}, \mathrm{SA}, C \tag{5.14}$$

followed by

$$x | \mathrm{FA}, \mathrm{FS}, P, D \; \forall x \in \mathrm{PA}, \mathrm{SA}. \tag{5.15}$$

The use of regression in this order is aimed at explaining first the results in terms of the algorithm used, and then the algorithm in function of the landscape, represented by the features. We note that this approach is justified by the implicit assumption of a set of causes and effects that prescinds from the mathematical method itself, which alone is not sufficient to determine the

causal relationship between the entities. It provides a more detailed analysis that the functional ANOVA, relating single instances and configurations to their results, instead of their average values [354].

Studies on the scaling behaviour of algorithms such as [355, 356] instead aim to predict the results obtained by an algorithm on instances larger than those that have been observed. This corresponds to inferring $R|P, D, A, C_2$, using data from a model (R, P, D, A, C_1) with C_2 in this case being a longer runtime than C_1.

5.4.4 Discussion

In this partial literature review, we have seen how virtually any method proposed to explain the behaviour of an optimization algorithm can be represented as an inference task on a subset of entities of our causal framework. By outlining the research question of different lines of works we have also shown how they can be related to each other. This is very important to navigate the research literature on this subject, and understanding precisely why two works that deal with the same algorithms on the same problems can offer different conclusions.

Of course, how to perform the inference task in practice depends on the observable variables, the tools available, and the computational environment that can be used. Nonetheless, we argue that in order to properly understand and explain the behavior of an algorithm and its results, it is required to collect information at the mediator level, that is, the features that represent the landscape. This is also a necessary step to furthen the generalization and automation of algorithm instantiation, an example of which we will discuss in the next section. This perspective is shared by other works such as [357], who effectively outline a sequence of operations that can be considered an effective implementation of one such inference tasks.

5.5 Applications

In this section we show how under our framework the algorithm selection and configuration problem can be conceptually related to each other, and the difference between them is not in the characteristics of the tasks but in their practical application. We then show how to exploit the relationship between the various entities involved to make use of data collections to automatically find configuration that work on new problems.

5.5.1 A unified view of selection and configuration

Given a set of instances arisen from a problem P and data D, a set of algorithms A and a cost function $c(\cdot)$ to measure the quality of each algorithm $a \in A$ on P, D, the Algorithm Selection Problem (ASP) [116, 117, 118] aims to find a mapping $\mathcal{S} : P, D \mapsto A$ such that $\forall d \in D$, $c(a(d))$ is minimized, or, in other words, to find the algorithm in A that obtains the best results on each instance.

ASP is usually tackled by computing a set of proxy problem features that represent the problem instance "well enough" and thus allow to find a mapping $\mathcal{S}$ that works "well enough" in practice. Features can be of various kinds. For continuous optimization problems, usually fitness landscape features are considered [121, 122, 124, 270]. For combinatorial optimization problems, very often features are problem-specific (or even class-specific) characteristics, such as statistics of the cities in TSP-related problems, or of clauses in SAT, thus requiring a certain degree of problem-specific knowledge. Efforts to use fitness landscape features to select algorithms for combinatorial optimization problems include [333] for flowshop scheduling, or [279] for the QAP. The selection of the set of features is crucial for the performance of the mapping. Problem-specific features can provide deep insights on the specific instance, at the cost of sacrifying generality of the mapping, while for more general features it is more difficult to pinpoint the ones that can give a reliable mapping.

Some authors have instead explored the possibility of using the automatic feature extraction of deep neural networks to bypass the manual definition of features in AS, on the bin packing problem and the TSP [358, 359]. The drawback is, of course, the difficulty or impossibility to a posteriori understand and explain the selection.

Given a set of instances that arise from a problem P and data D, an algorithm $a \in A$ with a set of parameters θ_a and a cost function $c(\cdot)$ to measure the quality of a on D, the Algorithm Configuration Problem (ACP) aims to find the set of parameters $\theta_a^\star$ such that $c(a)$ is minimized on P, D. That is, the goal is to find the best configuration of parameters for a on the instances. This problem is also known as *parameter tuning* or, in machine learning, *hyperparameter tuning*. Contrary to ASP, in ACP features are not necessarily considered. The problem is often modeled as a black-box stochastic optimization problem, so the best configuration $\theta^\star$ is the one which obtained the best results on a set of training instances, but no insights are given. However, by comparing different outcomes of ACP across different scenarios, we can infer explanations for the different results. The inclusion of features in the configuration process gives instead the Per-instance Algorithm Config-

uration Problem [360, 361, 362], which will be the focus of the rest of this section.

Automatic approaches for the ACP can be naturally applied to the task of automatically design algorithms, following the Programming by Optimization paradigm [193]. This makes it possible to tailor an algorithm for a specific scenario, performing extensive experiments and improving the final performance while reducing the burden for the user

The Combined Algorithm Selection and Hyperparameter optimization (CASH) problem [125, 126] is the problem of selecting an algorithm and its parameter configuration that best solve a problem, or $\arg\min c(a, \theta_a^\star)$, for the $a \in A$. It differs from ASP because in the latter the configuration task is omitted, and from ACP because it consider a set of algorithms rather than a single one, and finally it differs also from separately running a selection phase and a subsequent configuration of the outcome of the selection. It appeared more recently in the literature with respect to ASP and ACP, mostly due to the higher computational cost entailed.

Nonetheless, here we argue that these three problems can be conceptually represented as an inference task on the same set of variables. Informally, this can be understood by considering the portfolio A for the ASP as a categorical parameter of a super-algorithm, whose values are the different algorithms in the portfolio. With (H1) and (H2) we assume that we can instantiate at least one, and possibly more, algorithms from A. The mapping $\mathcal{S}$ searched for in Algorithm Selection as presented above can thus be described as $A|P, D,$ FA, FS, C. Since FA, FS already depend on observing P, D, and the selection is made by observing the results R for the alternatives evaluated, we can define ASP as the inference task $A|R,$ FA, FS, C. The emphasis on the mediators FA, FS emphasizes the "white-box" characteristic of ASP.

With (H1) and (H4) we also claim we can configure an algorithm using PA and SA. The usual black box formulation of the Algorithm Configuration Problem is PA, SA$|D, C$. Again, the final decision is based on results R observed in the tuning phase. Taking into account that the fact that the algorithm already works for a given problem P is usually assumed, we have $PA, SA|P, D, R, C$. For a proper definition of PA, SA, the same formulation applies to the PbO paradigm for Algorithm Design.

Therefore, we see the relationship between ASP and ACP. They are similar problems, in the sense that they are both inference problems based on problem and instances, and the outcome depends on the results (in fact, the racing algorithm [363] proposed in 1997 for model selection was subsequently used for configuring algorithms [146]). They instead differ in (i) whether we use fully configured algorithms (in A, for ASP) or building

blocks (in PA and SA, for ACP), and (ii) if we can observe (and use in the selection or configuration process) some features in F or L.

The separation is, however, neither crisp nor immutable. By observing any feature in F or L in a configuration or PbO task we open the black box and move towards a per-instance configuration approach. By allowing the configuration of numerical parameters of the algorithms in a portfolio for a selection task we have the CASH problem, which can also be instantiated as PA, SA$|R, F, L, C$. In practice, therefore, the question is what tools should we use for our application, and the answer depends on the possibility of observing features, the extent to which our algorithm or set of algorithms can be parameterized, and the computational constraints we have.

5.5.2 Per-instance configuration

Transfer learning is the application of knowledge obtained on one domain onto a different domain [364, 365]. It is an important task in machine learning, where it aims to reuse of past learning tasks as starting point for new tasks. Transfer learning is an inherently causal task, because the conditions that determine whether some information is transferrable between two tasks can be determined by observing the causal structure of the tasks [366, 367]. In particular, we have *direct transportability* between two tasks that share the same set of causal relationships [366].

In this section, we show how our causal framework can be used to transfer algorithm configurations obtained for some problems and instances into new scenarios. This is an open problem of great interest, because one of the limitations of many existing configuration approaches is the difficulty of making use of past experiments for new configuration tasks. Transferring configurations onto new scenarios would make it possible not only to speed up new configuration tasks, but also to exploit the vast amount of data collected in the past. This would also be extremely useful in scenarios where a configuration is not possible or not convenient, such as when the evaluation of the objective function is extremely expensive, when few instances are available, or in case of a problem that have to be tackled only once.

In a typical transfer learning task in machine learning, we typically consider one model and different distributions. The question is therefore how to translate the typical optimization scenario, consisting in algorithms, problems and instances into distributions. The starting point to answer this is to note that transfer learning and per-instance configuration are the same task. This can easily be seen with a derivation analogous to the one between AC and AS in Section 5.4. From our framework in Figure 5.1 we see that {FA, FS} is a set of mediator variables between the problem instances

$\{P, D\}$ and the results R (since we assume the landscape L is unobserved, an assumption derived from (H3)). Thus, for a specific problem $p \in P$, the goal is to obtain the best configuration for each instance, that is

$$\{\mathrm{PA}_d, \mathrm{SA}_d\} | \{p, d, \mathrm{FA}_d, \mathrm{FS}_d, R, C\} \; \forall d \in D_p \tag{5.16}$$

where FA_d and FS_d are the features computed for every instance d of problem p, to distinguish them from the sets of features FA and FS computed over all the instances, for example, the instances we use in the training of the model.

Equation 5.16 explicitly includes instantiation of a problem with its data. Following our argument in Section 5.4, we can remove this explicit dependency by observing a suitable set of features in $\{\mathrm{FA}, \mathrm{FS}\}$ that are representative enough of the information contained in $\{P, D\}$. We thus obtain

$$\{\mathrm{PA}_d, \mathrm{SA}_d\} | \{\mathrm{FA}_d, \mathrm{FS}_d, R, C\} \; \forall d \in D_p. \tag{5.17}$$

Hence, we see that the observation of this suitably representative set of features makes the starting objective function irrelevant, allowing us to characterize scenarios only at the feature level. Of course, this requires also the features to be problem-independent for the procedure to work across different objective functions, which is why we are going to focus only on landscape features, as they can be more easily defined to suit our goals.

The inference process is reported in Figure 5.2. The training data is composed of data collected on the training instances, pairing the features computed on each training instance with the configurations tested on it, and the results observed. For each instance, only the configuration that obtains the best results is kept. This data is fed to an inference engine; in principle, any engine can be used for the tasks. The inference engine will then take the features computed on the test instances and compute suitable configurations to use on the test instances.

5.5.3 Transfer of configurations of a fixed temperature simulated annealing

We follow our procedure to find good configurations for a fixed-temperature variant of simulated annealing (FTA). This is a very simple SLS that always accepts improving moves and accepts worsening solutions probabilistically, weighting the relative worsening in solution quality by a scaling parameter called temperature. Contrarily to the classic simulated annealing, which alters the temperature during the search, FTA always uses the same temperature value. Following our earlier work [18], we define FTA as a two-parameter SLS, the fixed temperature value (real-valued factor in $[0, 1]$ that

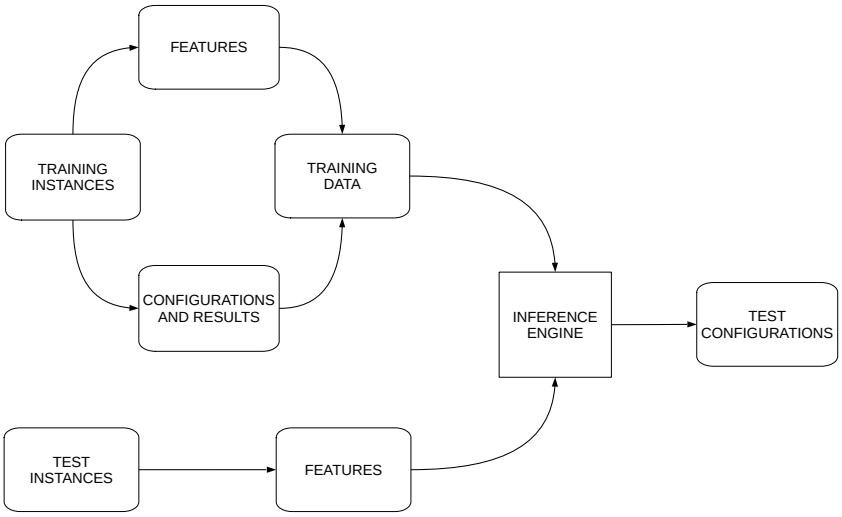

Figure 5.2: Inference process for the transfer of configurations.

rescales the average gap between consecutive solutions observed in a preliminary random walk), and the neighbourhood exploration, a binary parameter that determines whether the next solution to be evaluated in the neighbourhood of the incumbent has to be selected randomly or following some ordering.

We consider four objective functions: the Quadratic Assignment Problem (QAP), the Permutation Flowshop Problem under the makespan objective (PFSP-MS), the Permutation Flowshop Problem under the total completion time objective (PFSP-TCT), and the Traveling Salesperson Problem (TSP). For each objective function we have two instance classes. These are all permutation problems where all the possible solutions are feasible, and we can expect a similar algorithmic behaviour. In fact, in [18] we have characterized the behaviour of FTA and simulated annealing with respect to the landscape, analyzed using problem-independent features. The goal is to make use of collection of experiments, either performed in the past or simulated, for example via a surrogate model, to infer good configurations for each new test instance, both for the same problem and across different problems.

Following the causal framework, problem instances and algorithmic components become conditionally dependent if one or more common descendants are observed. In our case, the observations of features in FA is the set

Table 5.2: Set of landscape features used in the transfer learning. Each feature is computed with both a first-improvement and a best-improvement local search.

1	Number of moves to local optimum
2	Number of moves to local optimum, rescaled by instance size
3	Number of moves to local optimum, rescaled by neighbourhood size
4	R^2 of a linear model fit on the solution values
5	R^2 of a linear model fit on the solution values normalized in $[0, 1]$
6	R^2 of an exponential model fit on the solution values
7	R^2 of an exponential model fit on the solution values normalized in $[0, 1]$
8	R^2 of a linear model fit on the number of improving moves in the sequence of neighbourhoods traversed
9	Average proportion of neutral moves in the neighbourhoods traversed
10	Number of neutral last moves
11	Difference of features 4 and 6
12	Difference of features 5 and 7
13	Slope of the sequence of differences between best and average solution in a neighbourhood

of descendants that makes this possible. In particular, the landscape features we consider, listed in Table 5.2, are computed with a first improvement and a best improvement. Since the features in FA descend from one specific algorithm $a \in A$ instantiated from $\{\text{PA}, \text{SA}\}$, for which there are virtually countless valid combinations, we cannot expect for two algorithms a_1 and a_2 to observe similar values for the the respective set of features in FA. We choose therefore to use a probing algorithm a_0 that can approximate the behaviour of all or at least many FTA combinations, assuming that the set of features FA_0 observed can be considered an approximation of the set of features observed by the set of actual FTAs tested, and a first improvement can be seen as a FTA with temperature 0. Following [16], the "right" temperature value is the minimal one that guarantees the correct diversification.

The FTA algorithm is implemented in a component-based fashion in the emili framework, and an algorithm is instantiated at runtime by a desired parameter configuration [80]. We find good configurations for the various scenarios using the irace configurator, a state-of-the-art-offline configurator [148]. Each tuning is repeated 15 times, and the best configuration of each tuning is used on the entire test set. The configurations evaluated by irace

during the tuning phase, and the relative results observed, are also paired with the features computed on the training instances to be used as training data for the inference task. We use MERCS to infer the test configurations [368]. Also with MERCS we generate 15 configurations for each test instance, for statistical purposes. QAP and TSP experiments run for 10 seconds, while the runtime for the PFSP experiments is $0.015 \times n \times m/2$, where n is the number of jobs and m is the number of machines of an instance. All the experiments are performed on a machine with Intel Xeon E5-2680 v3CPUs running at 2.5GHz, with 16MB cache and 2.4GB of RAM available for each algorithmic run, in single thread mode.

The structure of the experiments is the same in every scenario presented in the following; they differ only in the set of problems and instance considered. First we perform one tuning with irace on the training set, then we couple the training data with the instance features to infer a new configuration for every test instances. We therefore compare the results obtained on the whole test set considered by the final configuration obtained using irace, with the results obtained by the set of test configurations obtained using MERCS, each one on a single test instance. The goal of this experiment is not to compare the tuning methods, but to show that a general purpose inference engine can be a valid alternative practice for automatic algorithm configuration, using collections of past experiments. This use of problem-independent features makes it possible to apply this process to any collection of problems, with meaningful outcomes.

In the first two scenarios we perform transfer learning between QAP instances. First we consider separately two different instance classes, performing separate tunings and evaluating separately the configurations found on random and structured instances. Inference is also applied separately for the two instance classes, so configurations for the test random instances are inferred only from data obtained during the tuning on random instances, and the same is done on the structured instances. The results are reported in Figure 5.3, separated by instance size, comparing the solution qualities obtained by the configurations found with both irace and MERCS in terms of percentage deviation from the best known solutions. The results are very similar. Only in the case of structured instances of size 60 there is a statistical difference in favour of the per-instance approach with MERCS, with a p-value $< 3 \times 10^{-5}$, where all the other cases have a p-value greater than 0.05. This happens because for each instance class and size there is a relatively narrow interval of good temperature values that allow FTA to reach those solutions in the given runtime. irace is able to identify it, and MERCS can exploit the information collected during the tuning.

In the second experiment we consider the two instance classes together,

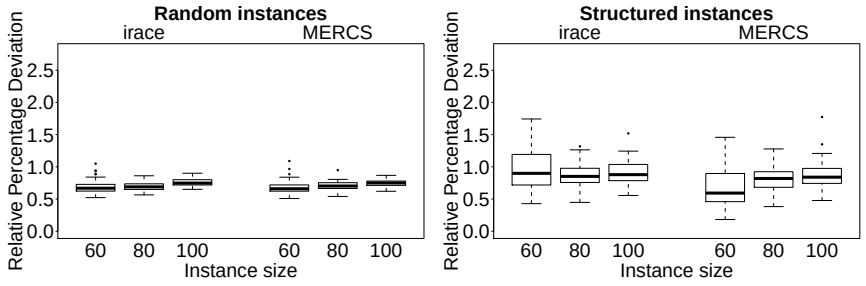

Figure 5.3: Results obtained in terms of relative percentage deviation from the best known solutions on the random and structured QAP instances by irace (separate tunings for each instance class) and MERCS; lower boxplots represent better results.

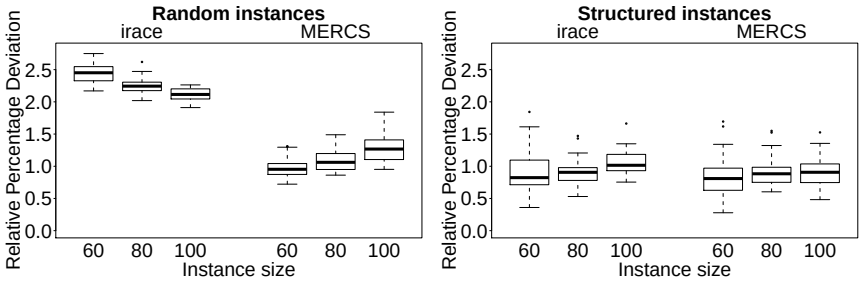

Figure 5.4: Results obtained in terms of relative percentage deviation from the best known solutions on the random and structured QAP instances by irace (separate tunings for each instance class) and MERCS; lower boxplots represent better results.

and tune FTA with irace using a training set of both random and structured instances, obtaining one configuration to test on the entire test set. The results are worse than in the previous case, especially on the random instances, because the two instance classes are better tackled using different configurations. During the training phase, however, irace evaluated also configurations that are good for the random instances, but were eventually discarded because of their poor performance on the structured instances. The feature-based approach with MERCS can exploit those configurations, obtaining better results, albeit not as good as in the previous experiment. In this case, in fact, the configuration space suitable for the random instances was not properly explored during the tuning phase by irace. The results are statistically equivalent only on the structured instances of sizes 60 and 80.

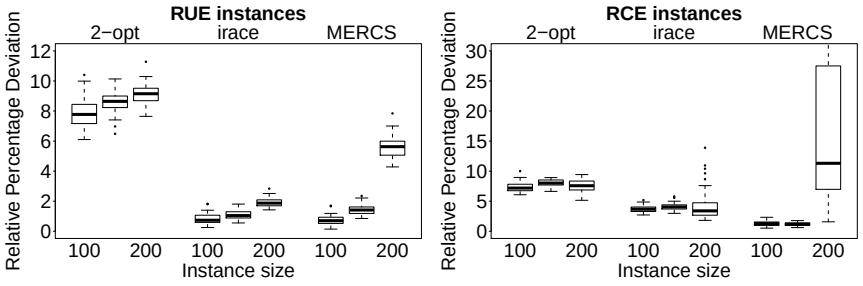

FIGURE 5.5: Results obtained in terms of relative percentage deviation from the optimal solutions on the RUE and RCE TSP instances by a 2-opt, irace (tuning on a training set of TSP instances, across all sizes) and MERCS (inference on QAP and PFSP data); lower boxplots represent better results.

5.5.4 Transferring configurations across different objective functions

In the third experiment, we use the feature-based approach to infer configurations for the TSP, starting from the tuning data generated by irace on the QAP and the two PFSP objectives. That is, MERCS uses data for these three problems to suggest configurations to test on each TSP instance. We consider two TSP instance classes, uniformly random (RUE) and clustered (RCE), with sizes 100, 150 and 200 cities. We created the instances using the generator from [369], with coordinates on a 1000×1000 matrix.[5] The results obtained by the configurations inferred by MERCS are compared against the results obtained by irace when tuned on a training set of TSP instances, considering the two instance classes separately but including the three instance sizes. In addition, we include the results obtained with a 2−opt heuristic as a reference. The results are reported in Figure 5.5. A summary on the temperature values obtained is reported in Table 5.3.

In both instance classes the results obtained by the configurations found by MERCS are comparable to those found by irace for instance sizes 100 and 150, while they differ greatly, in both solution quality and variance, for size 200. On the RUE instances the results are not significantly different for size 100 (p-value of 0.2059), with an average solution quality around 0.7% worse than the optimal solutions. For size 150, the solutions found by the configurations inferred by MERCS have an average of 1.42% RPD,

[5] On the RCE instances the coordinates may exceed those boundaries, due to the sampling procedure of the generator. This does not cause any numerical instability, so we do not need to correct a posteriori the coordinate list.

Table 5.3: Average temperature values obtained for each instance class, with the relative standard deviation. For irace the averages are computed over fifteen tunings, for MERCS across all the temperature values for the relative class and size. T_i is the average temperature obtained by irace on the two separate TSP training sets, and is the temperature value used in the experiments reported in Figure 5.5. T_i' is the temperature obtained by separate tunings on different instance class and size. T_M is the average of the temperatures identified by MERCS for the instances of each size.

Class	Size	T_i	T_i'	T_M
RUE	100	$(0.0349, 3.615 \times 10^{-3})$	$(0.0489, 3.014 \times 10^{-3})$	$(0.0372, 6.192 \times 10^{-3})$
	150		$(0.0373, 2.029 \times 10^{-3})$	$(0.0437, 9.002 \times 10^{-3})$
	200		$(0.0313, 2.156 \times 10^{-3})$	$(0.0569, 6.811 \times 10^{-3})$
RCE	100	$(0.018, 5.738 \times 10^{-3})$	$(0.054, 4.439 \times 10^{-3})$	$(0.0337, 6.247 \times 10^{-3})$
	150		$(0.0443, 3.908 \times 10^{-3})$	$(0.0409, 6.217 \times 10^{-3})$
	200		$(0.0134, 1.603 \times 10^{-3})$	$(0.0399, 8.95 \times 10^{-3})$

compared to a 1.1% of the solutions found by the configurations obtained by irace. For size 200, the results are considerably worse, with an average RPD of 5.6%, but still better than those obtained with a $2-$opt heuristic.

On the RCE instances, for sizes 100 and 150 the solutions found by the configurations obtained with irace are around $3-4$% worse than the optimal solutions, while those found by the configurations inferred using MERCS are on average 1.2% worse than the optimal ones. On size 200, instead, the quality of the solutions obtained using MERCS is drastically worse, even with respect to the results obtained with a $2-$opt.

The explanation can be found by observing the instances generated. Representative instances for each class and size are reported in Figure 5.6. TSP instances, overall, have a landscape that is globally very different from the landscape generated by the QAP and the two PFSP problems of Section 5.5.3. In particular, a random TSP solution can be expected to have an RPD of 120% from the optimal solution, while on the QAP the initial RPD rarely exceeds 50%. A local search, however, can exploit only local information at the neighbourhood level. A fixed temperature algorithm in particular will work well if the structure of the neighbourhoods is similar in different areas of the solution space [18].

RUE instances exhibit a "regular" behaviour, and a properly configured FTA can traverse a large portion of the search space and find good quality

Table 5.4: Distances (rounded to the nearest integer) for selected neighbours in each instance class. For each test instance we compute the average and standard deviation of the distance to the nearest neighbour, the distance to the 50%-th neighbour when ranked by distance, and the distance to the farthest point. The values reported are the average across each instance class and size.

	1-NN		50%-NN		Farthest point	
Class	Average	St.Dev.	Average	St.Dev.	Average	St.Dev.
RUE 100	52	29	528	46	987	144
RUE 150	42	23	529	43	1004	144
RUE 200	36	20	530	40	1013	144
RCE 100	23	19	171	34	400	63
RCE 150	15	14	139	27	338	52
RCE 200	16	13	328	27	725	70

solutions. This is a similar situation to the random QAP landscape of our previous two experiments, so MERCS can exploit these similarities up to a certain extent and find reasonably good configurations. The basic landscape differences, and possibly the set of features considered, make it however not possible to fully distinguish the details between the instance sizes, so the temperature values are suboptimal, and in an opposite trend with respect to the correct one. On our instance sets, good temperature values decrease as the instance size grows, as the increased density of the points makes the average relative difference between neighbouring solutions smaller.

On the RCE instances we observe instead a different situation. The generator we used creates instances with $\lfloor n/100 \rfloor$ clusters, which, in our case, results in 1, 1, 2 clusters for sizes 100, 150 and 200 respectively. These are very different instances. With a single cluster, the distances between each node are actually more uniform than on the RUE instances. As we see in Table 5.4, on RCE instances of sizes 100 and 150 not only the average distance between nearest neighbours is smaller than on the RUE instances, but also more distant points are anyway comparatively closer. This generates neighbourhoods that are quite uniform, a situation well suited for a fixed temperature algorithm, which in fact obtains better results than on the RUE instances.

Two clusters, on the other hand, will partition the distances into small intra-cluster and much larger inter-cluster distances, as can be inferred from

Table 5.4, making this a challenging scenario for a FTA. A careful inspection of the results on each instance shows how the solution quality found by the FTA worsens as the distance between the clusters increases. This correlation between cluster distance and results explains the huge variability obtained by MERCS. Conversely, overlapping clusters generate a landscape no different from the landscape generated by a single cluster, which results in good final solution qualities. A similar situation arises when the two clusters are adjacent, as also in this case it is possible for a FTA to perform well.

Some visual examples of different instance classes are given in Figure 5.6. In Figure 5.7 we show instead three different RCE instances of size 200, with overlapping, adjacent and distant clusters. It is interesting in particular to note the average solution quality found by the 15 configurations found by MERCS in these latter instances. On `rce200-85`, with two overlapping clusters, the average RPD is 1.69%. On `rce200-96` the two clusters are instead adjacent, and the average RPD obtained is 3.19%. Finally, on `rce200-95`, where the two clusters are very distant, the RPD is 50%. In Figure 5.8 we report the correlation between cluster proximity and final RPD obtained. For MERCS, that relies on features whose value is in some cases dependent on the size of the instance or of the neighbourhood, the correlation is very close to 1. Also for irace, when tuned across all the RCE instance sizes, the correlation is quite strong (0.787). When instead the tuning is done only on the RCE instances of size 200, the correlation on the test set is negligible; in this case the results are better for instances with average distance between clusters, and worse for overlapping or extremely distant clusters.

These results reflect the fact that different point distributions generate different landscapes, and in some cases they may resemble very closely landscapes observed in different situations, making it possible to obtain meaningful configurations from past experiments. On the other hand, on a completely new landscape our method fails to produce a valid configuration, and the too high temperature value prevents the FTA to converge to any decent solution.

5.5.5 Discussion

In this section, we have described the per-instance configuration task in terms of our causal framework, and have shown how to relate it to the transfer learning more commonly encountered in the machine learning literature. We have also reported a proof-of-concept that shows how existing tools can be used to successfully perform this task on various scenarios, including transferring configurations across different objective functions. In particular, this is made possible by the the observation of problem-independent

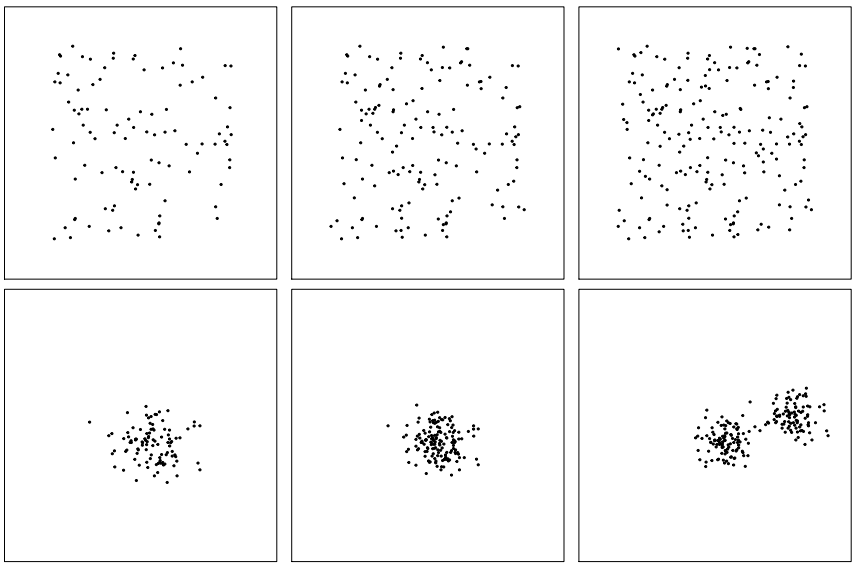

FIGURE 5.6: Sample TSP instances from our test set. Top row, from left to right: RUE 100, RUE 150, RUE 200. Bottom row, from left to right: RCE 100, RCE 150, RCE 200. These instances have been generated using the same random seed.

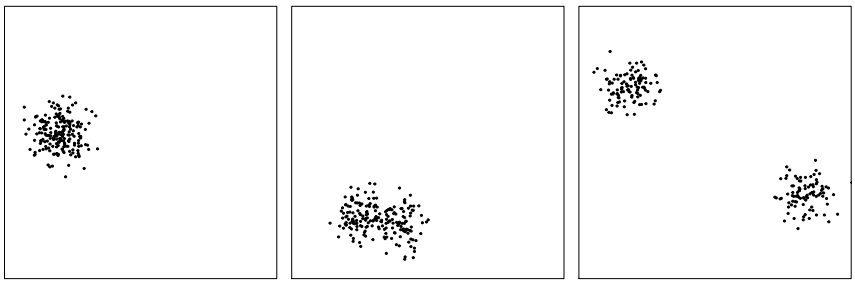

FIGURE 5.7: Different cases from the RCE TSP test instances with two clusters. From left to right: `rce200-85` with overlapping clusters, `rce200-96` with adjacent clusters, `rce200-95` with distant clusters.

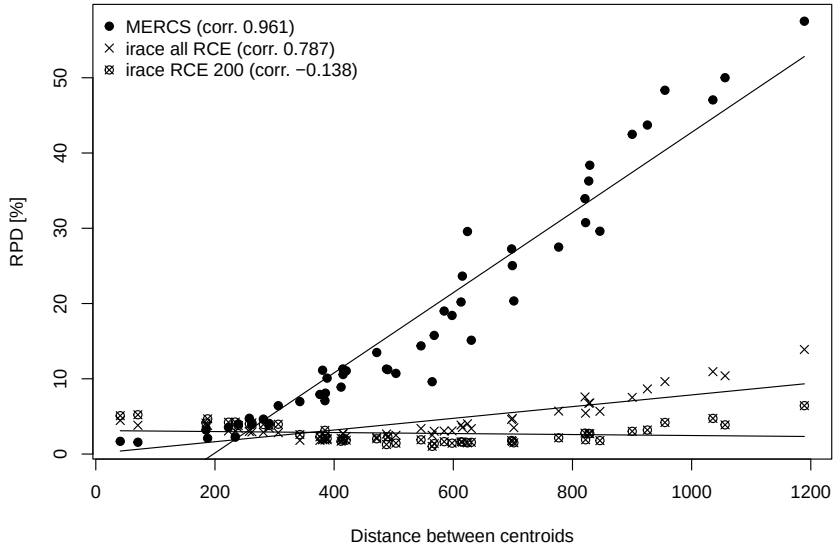

Figure 5.8: Correlation between cluster distance and RPD obtained on the RCE 200 test instances by MERCS, irace tuned across all the RCE instance sizes, and irace tuned only on RCE 200 instances.

features that represent the conditions encountered by an algorithm during the search. This is of particular interest especially in situations where past experiments are available, but it is not possible or not convenient to perform a focused configuration task. This may be in case of extremely expensive function evaluations, a limited budget, or the impossibility to obtain a homogeneous or representative training set of instances.

In the QAP experiments we have seen how our approach can make us of past experiments to match or even outperform a state-of-the-art offline batch configurator, when our starting data is sufficiently representative of the testing scenario.

We have then obtained configurations for TSP instances, starting only from QAP and PFSP observations. The results obtained clearly demonstrate the viability of our approach. On the RUE instances, and even more so on the RCE ones with one cluster, the results can be considered very good. In these cases, MERCS was able to make use of data from similar landscapes, that were generated by different objective functions. This strenghtens the validity of our working hypothesis (H_3), in that only the landscape suffices

to characterize the behaviour of an algorithm, and the observation of enough information about it can "cut off" the information flow from the variables in P and D. On the clustered instances of size 200, instead, the results were drastically worse, because either the landscape generated was too different from anything observed in the training data, or the features measured were not sufficient to properly represent the information needed.

Our results can of course be improved in several ways. First of all, a larger collection of problems, instances, features and algorithms increases the likelihood of observing new situations to use in future tasks. It is in fact unlikely that the current training base would allow for the same quality of results on other kinds of problems, in particular problems where the solution is not, in principle, a complete permutation. A more focused choice of features, achieved by feature selection or by the inclusion of additional relevant ones, can also improve the results. We also note how the inference engine we used, MERCS, is not a causal tool. It is possible that an alternative procedure that considers several configurations for every test instance, for example by means of a surrogate model, could obtain better results. The use of a proper causal inference engine could also be better suited for this task. Finally, the failure of our configurations on the TSP RCE instances of size 200 strongly indicates how our procedure needs to be augmented with some mechanism capable of evaluating the suitability of the configuration for the test scenario. The starting point to build such a mechanism would be to analyze the features computed and to relate them to the results obtained, to define a distance metric to not trespass in order for the procedure to output a test configuration. Another possible direction is to determine whether, based on the features computed, the test instance can be considered an anomaly with respect to the starting data, in which case the procedure should alert the user, rather than returning a configuration almost guaranteed to fail.

5.6 Conclusions

Understanding the behaviour of a stochastic local search (SLS) algorithm is an open question of fundamental importance in operations research and artificial intelligence. There is a huge corpus of literature aimed at this goal, but it is difficult to navigate it because of the different perspectives offered by each work, making it more difficult to narrow bridge the gap between theory and practice. In this work we have proposed a causal framework to model the relationships between the elements involved in the solution of an optimization problem, in order to relate all these approaches in a unified

perspective. We have also demonstrated how our approach could be useful to automatically configure an algorithm for an unseen problem, using tools that are already available. Albeit a proof-of-concept, our results prove that the systematic organization of knowledge that already exists is useful to both understand how algorithms work, and to advance the development of algorithms.

The framework is a formalization of our current understanding of the behaviour of optimization algorithms. It is therefore a representation of our subjective interpretation, and thus open for debate in case of disagreements. However, one of the main advantages of using such a model is the possibility of pointing out precisely what the subject of the disagreement is, making it more likely to have a productive discussion about the subject. This applies both at the general level, that is, on our general understanding of how algorithms work, and when considering specific research works.

While we focused on unconstrained combinatorial problems and SLS algorithms, the framework can be extended to include other classes of optimization problems, such as constrained or continuous ones, and other classes of exact and heuristic algorithms. The conceptual structure of the causal framework makes it possible to relate different works, and include information from different sources. This makes it possible to design a knowledge-based system, an extension of the currently available algorithmic frameworks that can include more and more algorithmic components and problem data, to eventually be able to automatically instantiate a valid algorithm for any unseen scenario. We have already discussed several possible directions to improve the performance of our transfer learning procedure. Additionally, the framework can also be used in a dynamic fashion to perform online configuration, selection and design based on local information, to tailor the resulting behaviour in an optimal way for different shapes of the landscape.

Having explicitly outlined the relationships between the elements involved in the solution of an optimization problem, we can easily define new research questions. A natural one is quantifying how well the set of observed features represents the landscape. While in Section 5.5 we ignored this issue, it is a fundamental step for applying our transfer learning procedure in a reliable way. This questions can be answered by performing *mediator analysis*, to estimate the proportion of the information from P and D to R that is effectively mediated by FA and FS. In fact, our working hypothesis (H_3) that says that a search algorithm traverses a landscape, regardless of how it was generated, in practice can be applied only if the information we collect about the landscape is truly representative of the real one.

Another possible direction is the estimation of the equivalence between algorithms, in order to establish how an algorithm a has to be composed

for it to obtain statistically equivalent results to another algorithm b, or to understand when it cannot reach the same level of performance. The theoretical equivalence or non-equivalence of SLS algorithms is a long-standing research direction in the theory of optimization algorithms and the systematic collection of data would make this direction both grounded on real experiments and scalable, thanks to the use of automated tools.

A systematic comparison of artificial and natural landscapes based on experiments with automatically generated state-of-the-art algorithms can further our understanding about the applicability of theoretical results to the development of algorithms for real-world problems. Similarly, it is possible to define an experimental protocol to answer the question of whether and how different classes of problems generate different landscape. The collection of data from different NP-hard problems may also be used in conjunction with the polynomial-time reduction between them, to understand how this transformation alters the landscape, and consequently, how this transformation reflects onto the development of algorithms.

Conclusions 6

Stochastic local search (SLS) algorithms are a class of methods routinely used in applications that require to solve optimization problems. They are particularly useful in contexts where rapidly obtaining a good solution is preferred to obtinaing a proof of optimality, or where a proof of optimality is actually infeasible. Despite their widespread use, their behaviour is still an open question in artificial intelligence and operations research. Consequently, the development of a SLS for a given scenario has always been more of a craft than a science, relying on experience and limited preliminary experiments. The introduction of automatic methods makes it possible to tailor the algorithm to the scenario and improve the results in an objective manner, but offers no explanations on how the algorithms were actually obtained, and little insights on the algorithm behaviour.

6.1 Developments of stochastic local search

In this thesis, I have summarized my work aimed at understanding how SLS algorithms actually work. I conducted extensive experiments, to be able to compare the best version of various algorithms on different scenarios. This empirical approach makes it possible to avoid the necessity of prior assumptions on the problems and instances considered, observing the behaviour of an algorithm under realistic settings.

I studied one of the oldest and most popular algorithms, simulated annealing (SA), and the class of SLSs that can be represented in its structure. Following a component-based view of SLS algorithms, SA can in fact be

considered a general template that contains several other algorithms that appear in the literature under different names. The precise innovations introduced in many variants of SA have been collected into an algorithmic framework. We made use of automatic algorithm configurators to efficiently improve SA algorithms proposed in the literature, and to obtain the best possible SA algorithms for different problems and instance classes. We have noted how ideas proposed thirty or more years ago are still valid and perfectly usable today. However, they need to be adapted to the specific scenario considered, and automatic algorithm configurators are the best way to take care of it. The ensemble of these ideas can also be efficiently exploited to easily generate new state-of-the-art algorithms. These observations are quite elementary, but nonetheless important, as they concretely support the opinion, shared by many but not all in the optimization community, that it is not necessary to continue inventing new algorithms, but rather that our efforts should be directed towards a deeper understanding and a more efficient use of the many tools we already have.

We have also observed how different scenarios may require different algorithms, but the converse is not necessarily true: a scenario can be efficiently tackled by different algorithms. This is again a basic observation with relevant implications, because it proves how the actual behaviour of an algorithm, that is, its balance between exploration and exploitation, is more important than its structure – that is, what mechanism is employed to control the exploration/exploitation tradeoff. In fact, an analysis of the most important components of a simulated annealing algorithm showed how the key part of a SA is its acceptance criterion, a component that is shared by all SLS algorithms. Conversely, the cooling scheme, the SA component that received the most attention in the literature, is relatively less important.

These observations raise the question of what influences the performance of an algorithm and how. In other words, given a particular scenario, we want to know what an algorithm should look like to obtain good results on it. We addressed this question by comparing two variants of simulated annealing on various problems and instance classes. For each scenario, we obtained the best configuration for both algorithms, for a fair and meaningful comparison, and we observed that for some scenario the traditional simulated annealing was outperforming its fixed temperature variant, while in other ones it was the fixed temperature variant to match or outperform its alternative. To explain the results, we investigated the conditions encountered by the algorithms during the search. The idea is to measure the same characteristics of the landscapes in all the scenarios, independently of the specific problems that generated them. We collected algorithmic landscape features, computed using simple local search algorithms to approximate the

real one observed by the two algorithms, and related them to the results obtained. We observed that when the structure of the neighbourhood is similar in both good and bad areas of the search space, a fixed temperature algorithm works well. Conversely, when different areas of the search space have neighbourhoods of different structure, SA has the capability of adapting, albeit not in a guided way, to the changes of the landscape. Beyond the specific observations, we have also outlined a methodology that supports the possibility of problem-independent, landscape-based approaches to algorithm selection and configuration.

In an attempt to formalize and generalize the intuitions behind this analysis, I have subsequently proposed a conceptual framework to relate the elements involved in the solution of an optimization problem. This approach has several advantages, including the fact that it requires to make our assumptions explicit, making it possible to pinpoint exactly the subject of works and discussions about optimization algorithms. I have, in fact, shown how this causal formulation can be used to relate various approaches aimed at understanding how algorithms work, by defining the precisely the research questions addressed. For example, this helps disambiguating seemingly constrasting results in the literature, identifying the conditions under which they have been obtained. Moreover, this approach makes it easier to identify the scope of research questions. We have demonstrated this by showing how algorithm selection and configuration can be seen as inference tasks on our framework, and identifying the entities required to transfer algorithm configurations across scenarios, making use of existing tools. To the best of my knowledge, this was the first instance of transfer learning across different problems.

The same relationships exploited to perform the transfer learning have broader implications. They show that to really understand the behaviour of an optimization algorithm a black-box approach is not sufficient – unless some properties of the landscape can be determined analytically, a scenario unlikely to happen in practice. It is in fact the observation of how the algorithms traverses the search space, as measured by the landscape features, that explains the results obtained by an algorithm.

6.2 Future work

This thesis can be considered a starting point for many future research directions. Here we outline some of them.

The set of variants proposed to the original simulated annealing formulation is wider than what we considered. It is one natural prosecution to

continue including these ideas, to hopefully widen the set of scenarios that can be efficiently tackled by our SA container. Likewise, in our analysis we did not focus on problem-specific components and their impact. They are of course a fundamental part of SLS algorithms, their efficiency is paramount, and including them in the automatic design of algorithms is of course going to both improve the final algorithm performance and our knowledge of its behaviour.

The analysis performed for SA and its acceptance criterion can also be performed for other components, and other algorithms. However, ideas that may seem different at first may effectively be equivalent from the point of view of the resulting algorithmic behaviour, and identifying them will make their collection and use more efficient. One important research direction is therefore to establish the equivalence between algorithms, and algorithm components. For example, by studying how a certain algorithm a has to be composed to perform well on different scenarios, we can try to determine whether it is possible to use this information to design a different algorithm b to perform well on the same scenarios. This sort of "mapping" could guide us in understanding whether there are algorithmic components that perform the same function for different algorithms, and measure the extent to which they enable the discovery of solutions of (statistically) equivalent quality, or if one algorithm is consistently better than the second one for certain scenarios. The theoretical equivalence or non-equivalence of SLS algorithms is a long-standing research direction in the theory of optimization algorithms, and the systematic collection of data would make this direction both grounded on real experiments and scalable, thanks to the use of automated tools.

As we have seen, the collection of information about how an algorithm traverses the landscape is the only way of improving our understanding about algorithm behaviour. Therefore, the identification of relevant sets of features for different algorithms, such as genetic algorithms or even mixed integer programming solvers, is the main way to improve our understanding of their behaviour. The causal perspective makes it also possible to address one fundamental question, that we did not address in Chapter 5 and that, to the best of our knowledge, has never been addressed in the optimization and automated machine learning literature: how representative of the actual landscape our set of features really is. In our opinion, this question was never asked in this field, precisely because of the lack of a coherent causal outlook on the subject. In fields where the use of causal reasoning is more widespread, such as psychology, this estimation is known as *mediation analysis*. The objective of this task is, given a cause and an effect, to compute the quantity of information that is mediated by a third variable, through which part of the causal information flows. The case we intend to address

is more complex than what is available in the literature, especially in terms of off-the-shelf software packages, so ready-made solutions are not available yet. It is, however, a promising research direction and a fundamental step in determining what characteristics of the landscape we need to observe to understand the key properties of a landscape and, consequently, what we need to pay attention to when designing effcient algorithms and algoritmic components.

The mediation analysis is important also to improve the performance of transfer learning. The transfer of configurations across problems relies on the working hypothesis asserting the independence of the search on a landscape from the process that generated it, which in practice can only be approximated. From a mediation perspective, we can say that the working hypothesis is fully realized only when the proportion of information from the causes to the results that is observed at the landscape level is close to 100%. Estimating the actual proportion of information mediated is therefore a key step in developing a reliable transfer procedure.

Many works study the behaviour of optimization algorithms on artificial problems, of which we can know and control the landscape. It is however not clear the extent to which such scenarios are representative of "natural" problems, problems that model real-world problems. Systematic experiments with automatically generated state-of-the-art algorithms may help in determining the applicability of theoretical results to the development of algorithms for real-world problems. More generally, we can study how different classes of problems and instances generate different landscape. Experimental analyses may be complemented with theoretical results in many more ways. For example, the polynomial time reduction between NP-complete problems may be used in conjunction with landscape studies to observe how the transformations of the problem formulations alter the landscape and impact the structure of good algorithms for them.

On the practical side, from the work here presented, we can also outline a procedure to move beyond transfer learning towards a knowledge-based automatic design of algorithms. Having formalized the relationship between problems, instances, features and algorithms, we can in fact systematize the collection and classification of information about experiments, to use it in future applications by means of automatic tools. This organization is, in my opinion, what makes it possible to apply automatic landscape-based algorithm design at a large scale. Furthermore, causal models natively provide methods to integrate data from different sources, and to impute missing observations. The application of causal reasoning such as counterfactuals can also be useful in both explaining the results obtained, and improving the automatic procedures employed to design the algorithms.

Finally, we have seen that the behaviour of SLS algorithms is determined by the local properties of the search space. Hence, the framework can be deployed in a cascading fashion, on the model of dynamic Bayesian networks, to contain local information. For example, we could construct a submodel containing information regarding improving phases of the search, another submodel containing information about uphill paths, and alternate between the two to select the best configurations for different parts of the search. Under this extended dynamic framework, is would be possible to perform landscape-based dynamic algorithm configuration and design and, by embedding the framework in the search loop, also landscape-based online tuning.

Comparison of acceptance criteria in randomized local searches A

One key component of stochastic local search algorithms is the acceptance criterion that determines whether a solution is accepted as the new current solution or it is discarded. One of the most studied local search algorithms is simulated annealing. It often uses the Metropolis condition as acceptance criterion, which always accepts equal or better quality solutions and worse ones with a probability that depends on the amount of worsening and a parameter called temperature. After the introduction of simulated annealing several other acceptance criteria have been introduced to replace the Metropolis condition, some being claimed to be simpler and better performing. In this article, we evaluate various such acceptance criteria from an experimental perspective. We first tune the numerical parameters of the algorithms using automatic algorithm configuration techniques for two test problems, the quadratic assignment problem and a permutation flowshop problem. Our experimental results show that, while results may differ depending on the specific problem, the Metropolis condition and the late acceptance hill climbing rule are among the choices that obtain the best results.

A.1 Introduction

Stochastic local search (SLS) methods are generic procedures commonly used to tackle hard optimization problems [12]. They are composed of a set of general rules of how to design effective heuristics for specific optimization

problems; hence, an alternative name for these methods is *meta*-heuristics. Often, the sometimes rather problem-specific heuristic algorithms derived from these rules are very effective in finding high quality solutions in short computation time, and for many problems such algorithms define the state of the art.

To achieve good solutions, SLS methods balance the intensification of the search in narrow regions, often needed to find the best solutions in promising search space areas, with the exploration of different areas of the search space. One mechanism that many trajectory-based SLS methods use to promote diversification is the acceptance of solutions that are worse than the current incumbent solution. In this article, we call *acceptance criterion* the function devoted to determining whether a newly proposed candidate solution should replace the current one. A first metaheuristic that proposed a probabilistic acceptance criterion for acceptinf a worse candidate solution is simulated annealing (SA) [199, 200]. It uses the so called Metropolis condition from statistical mechanics, which relies on a parameter called temperature, as acceptance criterion [198]. The name of the parameter mimics the temperature of a physical system, which was used in a Monte-Carlo simulation of physical systems proposed by Metropolis *et al.* [198]. According to the Metropolis condition, an improving or equal quality candidate solution is always accepted, while a worsening candidate solution is accepted with a probability that depends both on the quality difference between the current solution and the newly proposed one and the temperature parameter. To create a transition from search diversification to intensification of the search, in a typical SA algorithm the temperature is initially set to some high value (corresponding to a rather likely acceptance of worsening candidate solutions) and then subsequently lowered to make the acceptance of worsening candidate solutions less likely.

Over the years, various new ideas have been conceived with the motivation to improve over this usual acceptance criterion of SA algorithms. These new acceptance criteria have been compared in individual papers often directly to basic SA algorithms and in various such papers potential improvements have been reported. These new ideas include refinements of the Metropolis condition, such as generalized SA [225] and the bounded Metropolis condition [224]; a criterion where the acceptance probability of worsening solutions decreases geometrically [226]; and deterministic criteria such as threshold acceptance [180, 181], the great deluge and record-to-record travel algorithms [209], and the late acceptance hill climbing [211]. The latter four methods all accept a solution with probability one when it meets the specific, deterministic acceptance conditions. In our experiments, we also include a basic hill climbing acceptance criterion [228], which ac-

cepts a solution if and only if it improves over the incumbent, as a baseline the other criteria need to outperform.

The original articles proposing these acceptance criteria often report experimental results on few problem instances or on very small instance sizes. One reason is that many of these acceptance criteria were introduced when experimental conditions available were quite different from today. Hence, there is limited indication in the original works on how to apply the various methods to different problems. To just cite one example, in the original paper on threshold acceptance, the authors present a sequence of values for the "threshold" parameter, stating that "*We have the feeling (really only the feeling, not, for instance, the impression) that the sequence above is somewhat better* [than another sequence mentioned]" [181]. The comparisons in these papers are also usually performed against the Metropolis condition and a limited set of the other criteria.

In this work, we compare well-known acceptance criteria on common benchmark sets, derived from two classical, NP-hard problems, namely the quadratic assignment problem (QAP) and the permutation flow-shop problem with the total completion time objective (PFSP-TCT). To obtain unbiased results we tune the numerical parameters of the algorithms, using the automatic algorithm configuration tool irace [148]. We evaluate the impact of the nine different criteria we study in terms of the quality of the final solutions and the robustness of the criterion. Our experiments show that the results may change according to different problems, instance classes, or experimental condition. Overall, the Metropolis condition, its generalized version and, in particular, the rather recent late acceptance hill climbing are the criteria that gave the best results.

A.2 Literature review

We first introduce the notation used in the remainder of this work. We consider NP-hard combinatorial optimization problems, in which for a given problem instance π a globally optimal solution $s^* \in S$, where S is the search space of candidate solutions, is to be found. The quality of solutions is evaluated according to an objective function $f : S \mapsto \mathbb{R}$ and $f(s)$ is the objective function value for a generic solution s. Without loss of generality, we consider minimization problems, that is, for a globally optimal solution it holds that $f(s^*) \leq f(s), \forall s \in S$. Each algorithm we consider uses an iteration counter of the search process, which is denoted by i, and s_i is the new candidate solution evaluated in that iteration. The difference in terms of objective function value between two solutions s_i and s_j is denoted with $\Delta(i, j)$, or

Algorithm A.1: Outline of a generic randomized search algorithm.

Input: problem instance Π, $\mathcal{N}$, initial solution s_0, control parameters
Output: best solution s^*

```
1  best solution s* = incumbent solution ŝ = s_0;
2  parameter initialization, i := 0;
3  while stopping criterion is not met do
4  |  while parameters settings fixed do
5  |  |  i := i + 1;
6  |  |  generate a random solution s_i ∈ N(ŝ);
7  |  |  ŝ := accept (ŝ, s_i);
8  |  |  s* := best (s*, ŝ);
9  |  end
10 |  update parameters;
11 end
12 return s*;
```

simply Δ when no confusion may arise. With $\hat{s}$ we indicate the incumbent solution. The neighbourhood of $\hat{s}$ is denoted by $\mathcal{N}(\hat{s})$ and comprises all candidate solutions that can be reached from s by one application of the neighborhood operator.

All SLS methods that we consider can be interpreted as instantiations of the generic algorithm outlined in Algorithm A.1. It starts from a given initial solution as incumbent (line 1), and iteratively generates one new candidate solution in the neighbourhood of the incumbent uniformly at random (line 6); at iteration i the new candidate solution s_i can be chosen to replace the current incumbent $\hat{s}$ if it meets some criteria (e.g. it is an improving solution, line 7), otherwise it is discarded. Periodically, the parameter(s) that control the search may get updated (line 10). All the algorithms we examine here fit in this generic template. They only differ in the acceptance criterion. However, some of these algorithms may not use all the components of the algorithm; for example, the late acceptance hill climbing relies only on one parameter that is held constant during the run of the algorithm and, hence, does not need to be updated in the outer loop (lines 3 to 11). In the following, the counter k refers to the number of times the outer loop has been invoked. Conversely, SA and others evaluate solutions using the same parameter values in the inner loop (controlled by the temperature length, lines 4 to 9), and update the parameters in the outer loop.

SA, proposed independently in [199] and [200], is inspired by work in statistical physics [198]. In the usual, basic variants, SA iteratively generates and evaluates one random solution $s \in \mathcal{N}(\hat{s})$; if the new solution is better or equal to the incumbent in terms of objective function value, it replaces the incumbent one; otherwise it gets accepted with a probability that depends on the relative difference in terms of objective function values, $\Delta(s, \hat{s})$, and on the temperature parameter, denoted as T. The acceptance criterion of SA can be written as

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s, \hat{s}) \leq 0 \\ \exp\left(-\Delta(s, \hat{s})/T\right) & \text{otherwise.} \end{cases} \tag{A.1}$$

This probabilistic criterion is known as Metropolis acceptance criterion or Metropolis condition, and it is the distinctive feature of SA. We refer to this criterion simply as SA in the rest of this chapter.

More recently, in [224] the authors argue that solutions that are worse with respect to the incumbent by a quantity that exceeds a certain threshold ϕ_{BM} are not worth considering at all. This *bounded* Metropolis criterion (BSA) accepts a solution s with a probability

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s, \hat{s}) \leq 0 \\ \exp\left(-\Delta(s, \hat{s})/T\right) & \text{if} \quad 0 < \Delta(s, \hat{s}) \leq \phi_{BM} \\ 0 & \text{if} \quad \Delta(s, \hat{s}) > \phi_{BM}, \end{cases} \tag{A.2}$$

where ϕ_{BM} is a parameter.

Soon after the introduction of SA, the Metropolis acceptance criterion has been generalized in [225], where a variant of SA called generalized simulated annealing (GSA) was introduced. The GSA acceptance criterion is defined as

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s, \hat{s}) \leq 0 \\ \exp\left(-\beta f(\hat{s})^{\gamma} \Delta(s, \hat{s})\right) & \text{otherwise,} \end{cases} \tag{A.3}$$

where β and γ are control parameters. Even if the temperature parameter is not explicitly considered in GSA, it is possible to recreate the original Metropolis condition by defining $\beta = 1/T$.

In [226], the authors propose a criterion in what is the first occurrence of a SA variant that does not consider the temperature value in the acceptance of solutions. They propose to accept a solution with probability

$$p^k = \begin{cases} 1 & \text{if} \quad \Delta(s, \hat{s}) \leq 0 \\ p_0 \times \rho^{k-1} & \text{otherwise.} \end{cases} \tag{A.4}$$

where p_0 is the initial acceptance probability, $0 < \rho < 1$ is a reducing factor, and k is the number of times the probability has been updated. In this *geometric* acceptance criterion, the temperature value is (possibly) related only to the initial acceptance probability; during the search, the updating process of the probability matters, rather than the actual value of a temperature.

The actual need of stochasticity in the Metropolis acceptance criterion is questioned independently in [181] and [180]. In both works, the authors propose a criterion that accepts any move that is either improving or worsening by at most a given threshold $\phi^k > 0$:

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s, \hat{s}) \leq \phi^k \\ 0 & \text{otherwise,} \end{cases} \tag{A.5}$$

where ϕ^k is the value at step k of the threshold, which gets updated periodically. In [181], the authors consider a sequence of thresholds, without giving any indication on how to set its initial value or how to update it. Our implementation follows [180], maintaining the SA terminology: the initial value of ϕ is the initial temperature of SA, and the updating process of the threshold is called cooling. This threshold acceptance (TA) is a deterministic version of SA. At the time of its introduction, it was argued that using TA is faster than evaluating the Metropolis condition as it does not require the generation of a random number and the computation of an exponential. This advantage may be important when the computation of the objective function value of a neighboring candidate solution is very fast. However, for problems that benefit little from incremental update schemes or where the computation of the objective function value of neighboring candidate solutions is expensive (as is the case in the problems we study here), the advantage of a faster computation of the acceptance test diminishes.

Two acceptance criteria have been derived from TA and proposed in [209] as new algorithms. The first algorithm and criterion proposed in [209] is called record-to-record travel (RTR), and accepts solutions that do not deviate from the best solution found so far plus a given threshold ϕ:

$$p_{RTR} = \begin{cases} 1 & \text{if} \quad f(s) \leq f(s^*) + \phi \\ 0 & \text{otherwise,} \end{cases} \tag{A.6}$$

RTR is therefore a stricter version of TA, which compares the newly proposed candidate solution with the current incumbent; moreover, in the RTR algorithm ϕ does not get updated.

The second algorithm proposed in [209] is called great deluge algorithm (GDA) and is a radical change in terms of solution evaluation, as it moves

away from the idea of comparing solutions. The acceptance criterion of GDA accepts every move whose objective function value is lower than a certain threshold that gets progressively lowered during the search

$$p^k = \begin{cases} 1 & \text{if} \quad f(s) \leq \phi^k \\ 0 & \text{otherwise,} \end{cases} \tag{A.7}$$

with $\bar{\phi}^{k+1} = \phi^k - \lambda$, λ being a fixed parameter. The consequence of a lowering bound is that GDA becomes increasingly strict for accepting solutions.

A more recent work proposes another simple deterministic acceptance criterion, called late acceptance hill climbing (LAHC) [210, 211]. This algorithm makes no use of a temperature-like parameter, but maintains limited knowledge about the history of the search. It accepts every solution s that is improving either with respect to the current incumbent $\hat{s}$ or with respect to a solution visited κ iterations before, for a fixed κ:

$$p = \begin{cases} 1 & \text{if} \quad f(s) \leq \min\{f(\hat{s}), f(s_{i-\kappa})\} \\ 0 & \text{otherwise.} \end{cases} \tag{A.8}$$

Finally, we consider as baseline for the comparison a simple hill climbing (HC) algorithm [228] that accepts a solution if and only if it improves over the incumbent. Obviously, we expect the other criteria to obtain better results with respect to HC. While in practice one would implement HC using a systematic enumeration of the neighbourhood, we implemented it inside the framework of Algorithm A.1 for convenience.

A.3 Experimental setup

The nine acceptance criteria presented in Section A.2 are evaluated as candidate acceptance criteria for a generic algorithm outlined in Algorithm A.1. The common components of the nine implementations are: (i) a random exploration of the neighbourhood, (ii) no parameter restarting rule (e.g. temperature restart in SA), and (iii) a termination condition based on runtime. The runtime differs for each problem, so the actual value is given below.

For the criteria that need initial values for their parameters (such as the temperature for the SA family of algorithms, or the threshold ϕ in TA), we use a value proportional by a coefficient ϵ to the maximum gap between consecutive solutions observed during an initial random walk of length 10000 in the solution space. The parameters that need to be modified during the algorithm run time (e.g. temperature in SA or threshold in TA) are updated using a geometric decreasing; e.g., the temperature T in SA is updated according

Table A.1: Parameter values for the algorithms.

	Metro	BMetro	GSA	Geom	TA	GDA	RTR	LAHC
ϵ	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	$[0,10]$	–	–
α	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[0,1]$	$[1,100]$	–	–
τ	$[1,100]$	$[1,100]$	$[1,100]$	$[1,100]$	$[1,100]$	$[1,100]$	–	–
ϕ, ϕ_{BM}	–	$[0,1]$	–	–	–	–	$[0,1]$	–
β	–	–	$[10^{-4},10]$	–	–	–	–	–
γ	–	–	$[0,10]$	–	–	–	–	–
ρ	–	–	–	$[0,1]$	–	–	–	–
κ	–	–	–	–	–	–	–	$[1,10^4]$

to the formula $T_{k+1} = \alpha \times T_k$, where α is a parameter. The inner loop of Algorithm A.1 evaluates a number of solutions that is given by $\tau \cdot |\mathcal{N}(s)|$, where τ is a parameter and $|\mathcal{N}(s)|$ is the size of the neighbourhood of a solution s.

We choose to not use parameter restart schemes to better observe the impact of the main algorithm component that is studied, the acceptance criterion. The periodic reset or increase of parameters such as the temperature or the threshold is often beneficial to obtain better results, as it facilitates search space exploration, but it also has the side effect of smoothening the difference in terms of impact of the other components.

The parameter values, and their presence for each algorithm, are given in Table A.1. Parameters equivalent in scope and values are grouped together. The only algorithm that does not use the components described above is GDA. During the experimental phase, we have observed very poor results when using the GDA acceptance criterion with the choices above, indicating a lack of flexibility of the method. We thus consider the GDA algorithm in its original settings, which are anyway valid components that fit in the template of Algorithm A.1. The initial threshold value ϕ is computed proportional to the objective function value of the initial solution, using a coefficient $\epsilon \in [0,10]$; ϕ is updated according to the formula $\phi_{k+1} = \phi_k - \alpha$, where α is an integer in the interval $[1,100]$; we bound this decrease to 0. The other components are as described above.

Our setup considers as test problems the quadratic assignment problem (QAP) [52] and the permutation flow-shop problem with the total completion time objective (PFSP-TCT) [57, 58]. The QAP models the location of a set of facilities, with the goal of minimizing the overall distance between facilities taking into account also the flow between them. PFSP instead is a

scheduling problem where a set of jobs have to be ordered to be executed on a set of machines.

For the QAP we use a randomly generated initial candidate solution and the exchange neighbourhood, which is defined as

$$\mathcal{N}(s) = \{s' \mid s'(j) = s(h) \wedge s'(h) = s(j) \wedge \forall : l \notin \{j,h\}\, s'(l) = s(l)\}, \tag{A.9}$$

where $s(j)$ is the solution vector at position j. The neighbourhood size is $n(n-1)/2$, where n is the instance size. The running time considered for termination is 10 seconds. We consider two different instance sets of size 100, one where all QAP instance data are generated uniformly at random, and one randomly generated in analogy to structured real-world like QAP instances. Each instance set is divided into a training set of 25 instances and a test set of 25 instances. From here onwards, we refer to these two scenarios as random instances and structured instances, respectively. The two scenarios are not mixed, that is, the configurations obtained for the random instances are evaluated on the random instances and not on the structured ones, and viceversa.

For the PFSP-TCT we use the `NEH` heuristic [249] for the initial solution generation. For an instance of size $n \times m$, where n is the number of jobs and m the number of machines, the neighbourhood is the insert neighbourhood that randomly picks one element $s(j)$ in position j of the permutation $s = [s(1),\ldots,s(n)]$ and inserts it in position $k \neq j$, obtaining

$$s' = [s(1),\ldots,s(j-1),s(j+1),\ldots,s(k),s(j),s(k+1),\ldots,s(n)] \tag{A.10}$$

if $j < k$ and

$$s' = [s(1),\ldots,s(k-1),s(j),s(k),s(k+1),\ldots,s(j-1),s(j+1),\ldots,s(n)] \tag{A.11}$$

if $j > k$. The neighbourhood size is $n(n-1)$. In this case, we use an instance-based maximum runtime of $n \times m \times 0.015$ seconds. The training set consists of 40 randomly generated instances of size ranging from 50 jobs and 20 machines to 250 jobs and 50 machines [170] and the test set is composed by the instances `Tai31-110` of the Taillard benchmark [250]. We will, however, discuss separately the instances whose size is smaller than those covered by the training set (those with $n = 20$), covered by the training set (`Tai31-110`), and larger ($n = 500$).

We tune the numerical parameters using irace [148] with a budget of 2000 experiments per tuning on an Intel Xeon E5-2680 v3 CPU, with a speed of 2.5GHz, 16MB cache and 2.4GB of RAM available for each job.

For each algorithm we run nine tunings, evaluate the best configuration obtained from each tuning on the test set, and average the final solution quality obtained on each instance by the nine configurations. The real valued parameters have a precision of 4 decimal digits.

A.4 Experiments on the quadratic assignment problem

In Figure A.1 we show the results obtained by the nine algorithms after the tuning on the random instances and on the structured instances respectively. Each boxplot reports the results obtained on the test instances in terms of the average relative percentage deviation (ARPD) from the best known solutions. In Table A.2, we report the results of the Friedman rank sum test, obtained for the nine algorithms on the two QAP instance classes. The algorithms are ordered according to the sum of their ranks, and the difference in terms of rank sum with the best ranked algorithm is computed along with a statistical significance threshold. Algorithms whose rank sum differs from the best ranked one by a value larger than the significance threshold are statistically significantly worse than the best one.

On the random instances, RTR obtains the best results, with a mean ARPD slightly lower than 1%. The criteria based on the Metropolis condition (SA, BSA, GSA) and the LAHC algorithm obtain similar results, with mean ARPDs around 1.2 to 1.3%. Though the results are similar, BSA is consistently slightly better than the other ones. TA, GDA and the geometric criteria are worse, but still within the 2% average deviation, while HC stands around 3%. On the structured instances it is instead TA, LAHC and the family of the Metropolis criteria that obtain the best results, with average ARPDs all around 0.3%. The ARPDs among these five criteria are not statistically significantly different. The geometric criterion also obtains reasonably good results when considering the ARPD values, though from the rank-based analysis it is already clearly worse than the top-ranking group of acceptance criteria. RTR and GDA obtain solutions around 1% and 2% worse than the best known ones and, thus perform clearly worse than the other acceptance criteria. HC, as expected, is overall the worst, with ARPDs around 3 to 4%.

The difference of the results on the two scenarios can be explained by the different landscape of the instances [248, 370]. The random instances present a relatively flat landscape, where it is easy to discover local optima and move through them, but difficult to converge to very good solutions. On the other hand, the landscape of the structured instances is less flat,

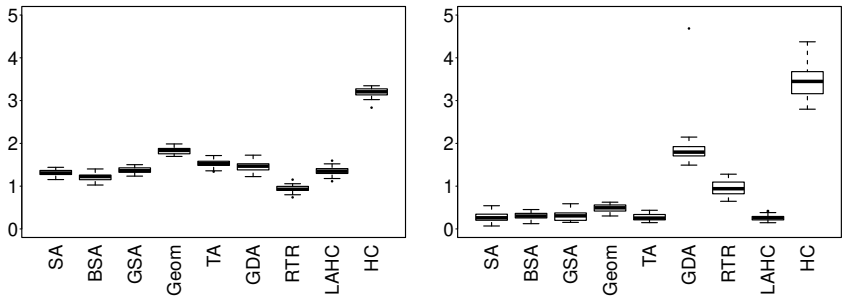

FIGURE A.1: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained on random (left plot) and structured instances (right plot).

Table A.2: Results of the Friedman rank sum test for the nine algorithms on the QAP instances. Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Instance class	Δ_R	Acceptance criteria ranking
Random	13.15	**RTR** (0), BSA (33), SA (70), LAHC (80), GSA (88), GDA (115), TA (140), Geom (174), HC (200)
Structured	16.08	**TA** (0), **LAHC** (0), **SA** (11), **BSA** (16), GSA (21), Geom (81), RTR (109), GDA (135), HC (158)

with "deeper" local optima than in the random instances. The criteria that strengthen the intensification along the search process are the ones that apparently benefit from this landscape. RTR compares candidate solutions to the global best, making it therefore more difficult to accept a worsening solution; additionally, using a same parameter settings across all instances may make it less robust.

A.5 Experiments on the permutation flowshop problem

In Figure A.2 we report the results obtained on the PFSP-TCT on the 80 instances of the Taillard benchmark with 50 to 200 jobs. The results of the

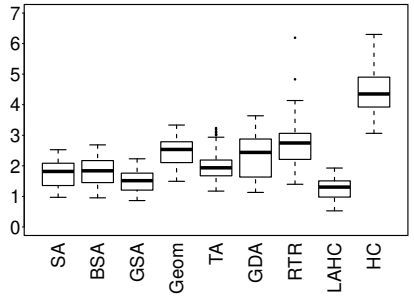

Figure A.2: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained on the instances `Ta031-110` of the Taillard benchmark.

Table A.3: Results of the Friedman rank sum test for the nine algorithms on the Taillard Benchmark. Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Instance class	Δ_R	Acceptance criteria ranking
`Ta001-030`	13.57	**LAHC** (0), GSA (30), TA (74), SA (82), BSA (87), Geom (142), RTR (193), GDA (206), HC (212)
`Ta031-110`	27.58	**LAHC** (0), GSA (94), SA (229), TA (267), BSA (279), GDA (412), Geom (455), RTR (490), HC (636)
`Ta111-120`	2.93	**LAHC** (0), GSA (11), RTR (19), GDA (31), HC (39), Geom (50), TA (63), SA (67), BSA (80)

Friedman rank sum test for the nine algorithms are reported in Table A.3, separated for the three sets of instance subclasses considered (smaller than in the training set, size covered by the training set, and larger).

The results in Figure A.2 for the PFSP-TCT exhibit higher variance than on the QAP, because they report results obtained on 8 subclasses of instances, with a different number of jobs and machines. Inside each instance subclass, the variance is much lower, indicating consistent results for each instance size.

Late acceptance hill climbing is the criterion that obtains clearly the best results, with an average ARPD of 1.2%. It is also more robust than the others: its worst results are below 2% of ARPD. GSA comes second best,

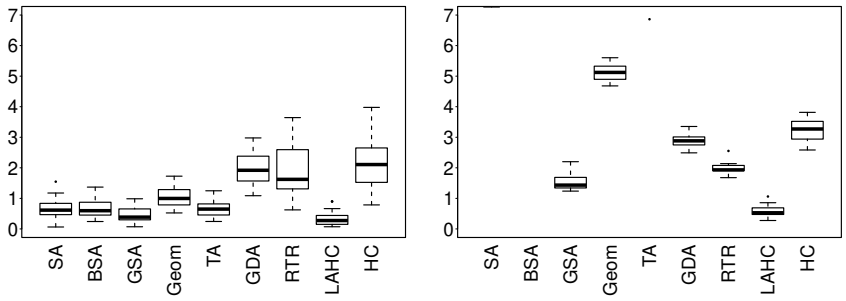

FIGURE A.3: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained on the `Ta001-030` (left plot) and on the `Tai111-120` instances of the Taillard benchmark set (right plot, SA, BSA, and in part TA results are not shown as they were very poor).

with an average deviation of 1.5%, followed by SA and BSA (respectively 1.7% and 1.8% on average; a Wilcoxon test shows no statistically significant difference between them). TA obtains results comparable to BSA. The other criteria obtain results between 2% and 3% of ARDP, still significantly better than HC.

The different instance sizes in both the training and test sets favour the more robust solutions. GSA appears to be more robust than the original SA, thanks to the increased flexibility given by the additional parameters. Looking at the different instance subclasses, however, LAHC consistently outperforms all other acceptance criteria.

We focus now on the instance subclasses not covered by the training set, either because they are too small (`Ta001-030`) or because they are too big `Tai111-120`. We can observe in Figure A.3 and in Table A.3 that LAHC is consistently the best performing one, followed by GSA. Overall, on the small instances all algorithms obtain results that are according to the ARPD values at least as good as on medium size instances of `Ta031-110`, with GDA and RTR being the only ones for which this is not true.

On the large instances, LAHC and GSA remain the top-performing algorithms with an average ARPD of 0.59% and 1.57%, respectively. SA and BSA instead obtain good results on the small instances, but perform very poorly on the larger ones, with ARPDs ranging around $9 - 10\%$, much worse than even HC. This effect is due to the parameters selected by the tuning phase, which are calibrated for instance sizes occurring in the training set and the given running time. The convergence behaviour of SA and BSA does not scale to large instance sizes for which the evaluation per so-

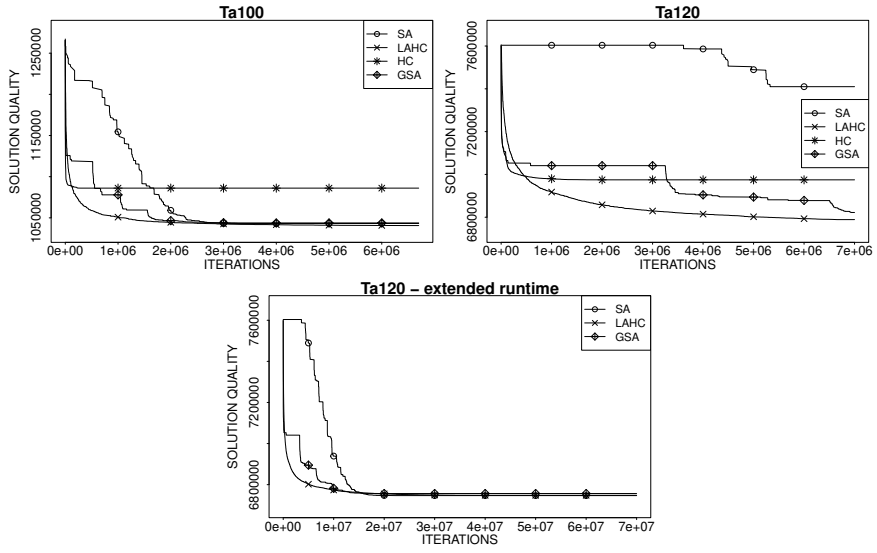

Figure A.4: Convergence behaviour of the SA, GSA, LAHC and HC algorithms for the `Ta100` (top left plot) and `Ta120` (top right plot) instances; in the bottom plot, the results for SA, GSA and LAHC on `Ta120` with 10× the original running time.

lution is much more costly (the evaluation scales quadratically with instance size while the computation time only increases linearly). This is illustrated in Figure A.4, where we compare the development of the solution quality over the number of iterations for SA, GSA, LAHC and HC on two instances: `Ta100`, whose size is 200×10 and is covered by the training set, and `Ta120`, whose size is 500×20. On `Ta100`, the four algorithms quickly discover good solutions; still, the convergence of SA is slower with respect to the other three. On `Ta120`, SA is clearly unable to converge within the originally allocated computation time. In the right plot of Figure A.4 we observe the convergence of SA, GSA and LAHC on `Ta120` with a runtime ten times higher (1500s instead of 150s on that instance – HC not included in the plot): the convergence is more similar to the one observed for `Ta100`, with also SA discovering high quality solutions. In particular, GSA finds a solution very close to the best known one (6756860 vs 6755722), while SA and LAHC both find a solution of better quality than the currently best known one (6746818 and 6748131, respectively). It is interesting to note that SA has now found the best solution, while LAHC has continued improving until more than half the time available.

A.6 Conclusions

We have observed how a careful tuning of the numerical parameters is crucial to obtain good results, in terms of both solution quality and convergence. Across our two benchmark problems, the algorithm that obtained the best results is LAHC. It exhibits a good convergence behaviour, quickly discovering good solutions in the beginning of the search, and continuously improving afterwards. It is also very robust, as it is the best performing algorithm across the whole Taillard benchmark for the PFSP-TCT, and it scales well also to instances of different sizes, unseen in the training set. Despite its simplicity (only one parameter to be tuned), LAHC makes a good use of the history of the search, as any solution it accepts is never worse than at least another one it has accepted in the past.

SA obtains overall good results, but it requires a proper tuning, as we have observed, in particular, for the large PFSP instances. It is able to obtain good results, but it might do so slowly; it is therefore advisable to tune SA for anytime behaviour [243] to obtain good results in a shorter time. BSA performs similarly to the standard SA. GSA, instead, has been shown to be flexible, outperforming SA also in terms of scalability and anytime behaviour. The geometric acceptance criterion is overall inferior to tthose derived from the original SA.

TA has obtained results overall not very different from SA. RTR has obtained good results on the random QAP instances, but was among the worse performers in the other scenarios, probably because of the fixed value of its threshold. GDA also showed a lack of flexibility, requiring a different setup and thus making its use within other algorithms more problematic.

As future work, we plan to extend the analysis to different conditions that might improve the performance of the various criteria. For example, a temperature restart, which is a common option in various SA algorithms, may change the conclusions of especially those criteria that rely on the temperature parameter. In addition, we plan to extend the set of acceptance criteria that are considered in this work and also extend the set of test problems to increase the experimental basis on which our conclusions rely. Finally, we intend to test the various acceptance criteria considering other aspects such as anytime behavior or robustness to other scenarios that differ in instance size and termination condition.

Effect of transformations of numerical parameters in automatic algorithm configuration B

We study the impact of altering the sampling space of parameters in automatic algorithm configurators. We show that a proper transformation can strongly improve the convergence towards better configurations; at the same time, biases about good parameter values, possibly based on misleading prior knowledge, may lead to wrong choices in the transformations and be detrimental for the configuration process. To emphasize the impact of the transformations, we initially study their effect on configuration tasks with a single parameter in different experimental settings. We also propose a mechanism of how to adapt the transformation used and give exemplary experimental results with that scheme. We also propose a mechanism for how to adapt towards an appropriate transformation and give exemplary experimental results of that scheme.

B.1 Introduction

Automatic algorithm configuration has shown to be crucial for reaching high performing parameter settings of search algorithms [193] and a number of effective configurators are available [134, 138, 142, 143, 148, 371]. Modern configurators can handle a large number of parameters and different types of parameters including categorical, ordinal and numerical ones. In this article, we focus on the latter type of parameters.

Choosing the value of a parameter for an algorithm can be difficult due to three intertwined issues: (i) the wide range of possible values, (ii) the possibility of parameter interactions, and (iii) the impact a variation of a parameter value has. Automatic algorithm configuration techniques relieve algorithm designers and practitioners of the first two issues. For numerical parameters, the third issue depends in part on the parameter representation and the sensitivity of each single parameter during the configuration task. The representation issue is related to the question whether a parameter should be varied according to an additive or a multiplicative scale. For example, in a population based-algorithm it is intuitively clear that the sensitivity of the population size depends on its value: the effect of increasing the population size by a constant number of, say, ten has a very different impact if the increase is from 1 to 11, from 10 to 20, or from 100 to 110. Instead, it seems more intuitive to change population size by a specific factor. Choosing an additive or multiplicative scale is related to the question whether a parameter should be presented in a normal scale or in a logarithmic scale.

In this article, we study the effect of a transformation of the parameter values for automated algorithm configuration. We consider five transformations of a single parameter and study their effect under different configuration budgets. We show how the right transformation helps considerably to discover good parameter values; this impact is stronger for low configuration budgets. At the same time, we show how a wrong transformation, which may be chosen due to misleading prior knowledge, can instead be harmful. As a possible alternative to a manual choice of a transformation, we explore a scheme to adapt it at execution and we give some illustrative results of its possible effectiveness.

The implementation of the transformation depends on the particular configurator. For example, ParamILS [142] requires a discretization of numerical parameters and one possibility is therefore to discretize the numerical interval in normal, logarithmic, or another fashion. SMAC [138] includes the possibility of exploring a parameter range in a logarithmic way (e.g. $1, 10, 100, \ldots$) by specifying a transformation in the parameter definition, which allows to scrutinize better on the lower part of the parameter range. A more general solution is to apply a transformation in the wrapper script used to execute the experiments. This possiblity is currently the only one available when using irace [148]; it can also straightforwardly be used in other configurators such as SMAC.

The article is structured as follows. In the next section, we define the transformations we studied. We empirically observe their impact in Section B.3, and we propose an automatic selection scheme in Section B.4, along with an experimental evaluation. Further experiments are available as sup-

plementary material at http://iridia.ulb.ac.be/supp/IridiaSupp2017-011.

B.2 Parameter transformations

As a baseline, we consider the parameter space with no transformation, e.g.

$$\mathcal{I}(x) \triangleq f(x \in [a,b]) \mapsto x \in [a,b] \tag{B.1}$$

where $[a,b]$ is the range of possible values for parameter x. This is the default behaviour of configurators such as SMAC or irace. This baseline is well-suited for parameters whose effect reflects an additive scale. For a multiplicative scale, we may apply a logarithmic transformation of the parameter:

$$\mathcal{L}og(x) \triangleq f(x \in [\log a, \log b]) \mapsto 10^x \in [a,b], \tag{B.2}$$

that is, we sample the logarithm of the parameter under consideration. For ease of representation, we consider base-10 exponentials and logarithms. (Note that this is also the transformation that is natively provided by SMAC.) As for $a = 0$, $\log a$ is not possible, we use as lower bound the precision δ that we use when we sample real-valued parameters. For example, if $\delta = 4$ and the parameter range is $[0,1]$, our sampling interval will be $[-4,0]$.

The effect of a $\mathcal{L}og(x)$ transformation is a modification of the probability of sampling specific parameter ranges, favoring the lower part of the interval. For example, assuming uniform distributions, in a logarithmic range $[0,4]$, corresponding to a final parameter value range of $[1,10\,000]$, there is a probability of 0.25 of sampling a value between zero and one (three and four), which actually corresponds to a transformed parameter between one and ten ($1\,000$ and $10\,000$) once rescaled, an interval whose probability of being hit in a non-transformed scale is 0.001 (0.9).

In case we are interested in exploiting the upper subrange of the interval, we define the reflection in the parameter range of the logarithmic transformation:

$$\mathcal{RL}og(x) \triangleq f(x \in [\log a, \log b]) \mapsto b - 10^x + a \in [a,b]. \tag{B.3}$$

This transformation is useful in the dual case of $\mathcal{L}og(x)$: consider a parameter that represents a probability that we want to take values very close to 1, with a precision of four decimal digits, resulting in a sampling interval $[-4,0]$. As an example, with probability 0.25, we could sample a value between -4 and -3, which maps to the range $[0.9991,1]$ after the transformation.[1]

[1] Note that SMAC offers the possibility of a $\mathcal{L}og(x)$ transformation but not that of $\mathcal{RL}og(x)$.

Milder transformations are also possible, for example the power function with exponent greater than 1, that in an interval $[0,1]$ magnifies the area close to 0. Here, we consider a quadratic transformation

$$\mathcal{S}(x) \triangleq f(x \in [a,b]) \mapsto \left(\frac{x-a}{b-a}\right)^2 \cdot (b-a) + a \in [a,b], \qquad \text{(B.4)}$$

exploring the lower subrange; by reflecting it in the parameter range we obtain

$$\mathcal{RS}(x) \triangleq f(x \in [a,b]) \mapsto b - \left(\frac{x-a}{b-a}\right)^2 \cdot (b-a) \in [a,b] \qquad \text{(B.5)}$$

that explores the upper subrange.

Also powers with exponent between 0 and 1 can be useful to exploit the upper part of the interval; we consider the square root of the normalized sampled value

$$\mathcal{S}qrt(x) \triangleq f(x \in [a,b]) \mapsto \left(\frac{x-a}{b-a}\right)^{1/2} \cdot (b-a) + a \in [a,b]. \qquad \text{(B.6)}$$

B.3 Empirical assessment of the transformations

We benchmarked the six transformations of Section B.2 to observe their impact in different scenarios. All the experiments reported in this chapter have been conducted on an Intel Xeon E5-2680 CPU with 2.4GB of RAM available for each experiment and running under Linux Rocks 6.2. The tuning was done with irace version 2.4 [148]. Each tuning was run 30 times, resulting in 30 tuned parameter values. We then computed the average percentage deviation (APD) obtained for each tuned value on the test instances.

In this section we show two artificial examples that employ different variants of Simulated Annealing (SA). SA is a popular metaheuristic, which moves through solutions accepting them using a probabilistic acceptance criterion. In the original formulation, SA uses the Metropolis criterion [198], which accepts a solution in a minimization problem with a probability given by

$$p = \begin{cases} 1, & \text{if } \Delta \leq 0 \\ \exp(-(\Delta/T_k)), & \text{otherwise} \end{cases} \qquad \text{(B.7)}$$

where Δ is the difference between the cost of the new and the current solution and T_k is the temperature parameter after k cooling steps, that is, after

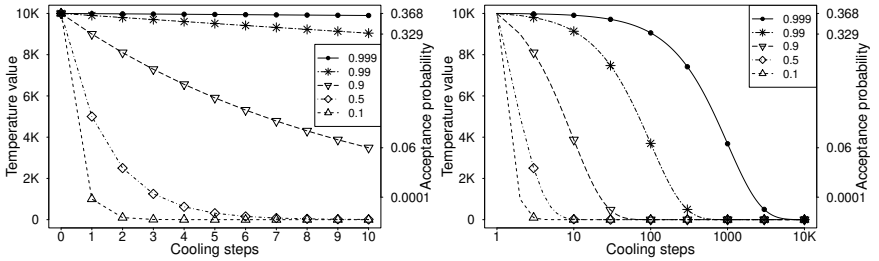

FIGURE B.1: Evolution of the temperature for different values of α, measured over 10 cooling steps (left plot) and 10 000 cooling steps (right plot, in logarithmic scale). On the right side is indicated the acceptance probability of a solution whose objective function value is worse than the optimal one by $\Delta = 10\,000$.

k reductions of the temperature value. A solution is always accepted if it is better than or equal to the current incumbent solution; if it is worse, it is accepted with a probability that depends both on the relative difference in terms of objective function values between the two solutions and on the stage of the execution. The temperature parameter is usually decreased during the search following a so-called cooling scheme; the number of solutions evaluated at a same temperature value T_k is called temperature or epoch length. The effect of the temperature parameter is to gradually transition from an initial exploratory search phase, in which more uphill moves are accepted, to a final exploitative search phase, in which almost only improving moves are accepted.

We evaluate the transformations applying the SA algorithms to the Quadratic Assignment Problem (QAP) [253]. We use instances of size 100 facilities and locations, generated uniformly at random [372]; 25 instances are used for the tuning, and 25 for testing of the tuned parameter settings. For both tuning and testing we stop the algorithm after two seconds of CPU time. The tuning is done using three different budgets, namely 125, 500 and 2 000 experiments per tuning. The convergence of the parameter to the best value under the different transformations is reported in the supplementary material.

B.3.1 Geometric cooling in simulated annealing

We consider a SA with a geometric cooling as

$$T_k = \alpha \times T_{k-1}, \tag{B.8}$$

where $\alpha \in [0, 1]$ controls the decrease of the temperature value. Low values of α enforce a very quick decrease in the temperature, as shown in Figure B.1; for example, for values of α equal to 0.1 or 0.5 the acceptance probability becomes negligible already after a few cooling steps, making the acceptance of worsening solutions very unlikely. As α gets closer to 1, many more cooling steps are needed to decrease the acceptance probability of worsening moves. Traditionally, in the SA literature α is chosen close to 1, to ensure that the algorithm does not converge too quickly.

We consider two cases, one in which the best value of α is close to 1, and one in which the best value is close to 0; such settings are obtained by fixing the temperature length either as very small or very large. These values are manually chosen to be 1 and 500 times the size of the neighbourhood, respectively; they guarantee to observe, in our scenario and computational environment, optimal values for α close to 1 and to 0, respectively. For small values of the temperature length the search shows an exploitative behaviour, updating the temperature very frequently and rejecting non improving moves more often; thus, the preferred values of α are close to 1, to reduce the decreasing rate of the temperature and maintain exploration. As the temperature length increases, the temperature gets updated less frequently, and the search loses exploitative potential; to compensate, the values of α tend to decrease to favour convergence. The other parameters are kept fixed: the initial temperature is given by the maximum gap between consecutive solutions observed during an initial random walk of 10 000 steps in the search space.

B.3.1.1 Short temperature length

The APD in dependence of the values for α obtained with a small temperature length is shown in Figure B.2 for the whole parameter range (left plot) and for the best-performing value (right plot). There is a slow but clear improvement as α grows, until $\alpha = 0.993$; for $\alpha = 0.994$ the results are already very bad, as shown by the spike in the plots for values close to $\alpha = 1$. This is an extreme condition, where a tiny variation can make a huge difference but also a good benchmark as the very best parameter values stem from a rather narrow range.

In Figure B.3, we show the results obtained by each transformation with a budget of 125, 500 and 2 000 experiments per tuning. It is quite evident how the normal sampling is outperformed by the transformations aimed to exploit the region of the range close to 1, which are $\mathcal{RL}og(x)$ and $\mathcal{RS}(x)$. The transformations $\mathcal{L}og(x)$ and $\mathcal{S}(x)$ obtain worse results, as they tend to exploit the wrong range of values. Analyzing more in detail the behaviour

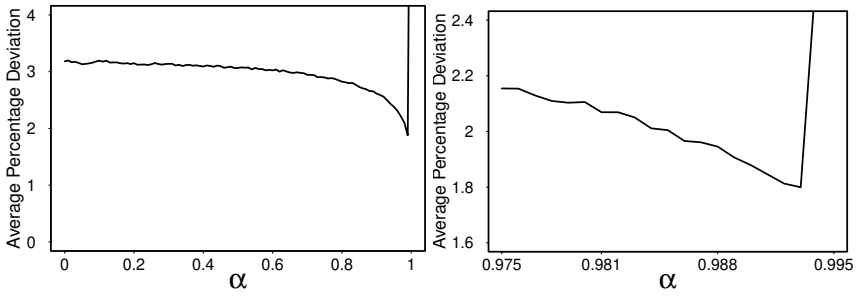

FIGURE B.2: APD in dependence of the value of parameter α for small temperature length; the left plot gives the full parameter range and the right plot the subinterval with the best-performing values.

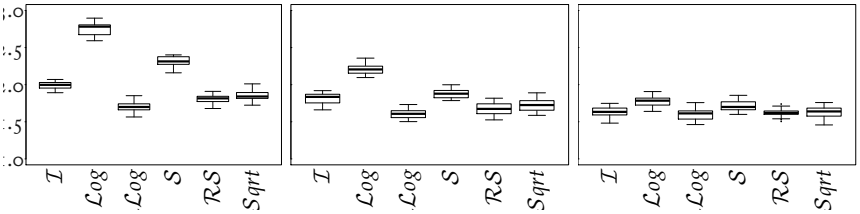

FIGURE B.3: APD (given on y-axis) obtained by the different transformations for a small temperature length. The three plots show the results obtained with a budget of 125, 500 and 2 000 experiments, respectively.

of the transformations aimed to exploit the upper subrange of parameter values, $\mathcal{RL}og(x)$ converges very quickly to the best values, while $\mathcal{RS}(x)$ and $\mathcal{S}qrt(x)$ exhibit a slower convergence. The remaining transformations are able to converge to a narrow, though suboptimal, subrange of values, but only for the high budget case.

B.3.1.2 High temperature length

In Figure B.4, we show the APD values for a high temperature length in dependence of the parameter α over the full (left plot) and the best-performing (right plot) subrange. In this case, we have a smoother and relatively wider "good area" of parameter values than in the case of a short temperature length, roughly centered between $\alpha = 0.025$ and $\alpha = 0.035$.

Figure B.5 shows the obtained results. As expected, the situation is reversed with respect to the previous situation. In this case, the $\mathcal{L}og(x)$ and $\mathcal{S}(x)$ transformations exhibit a slightly quicker convergence to the best values than the identity transformation, while the remaining transformations

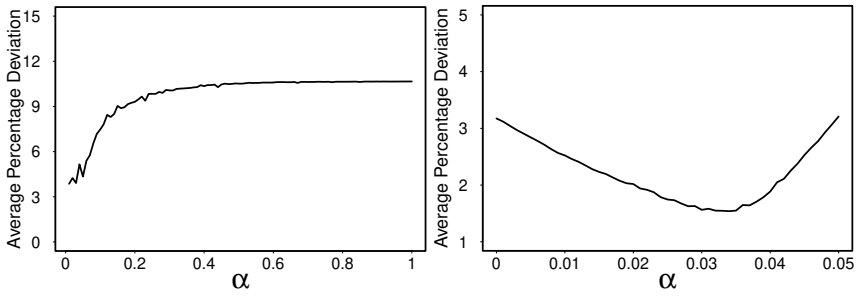

Figure B.4: APD in dependence of the value of parameter α for high temperature length; the left plot gives the full parameter range and the right plot the subinterval with the best-performing values.

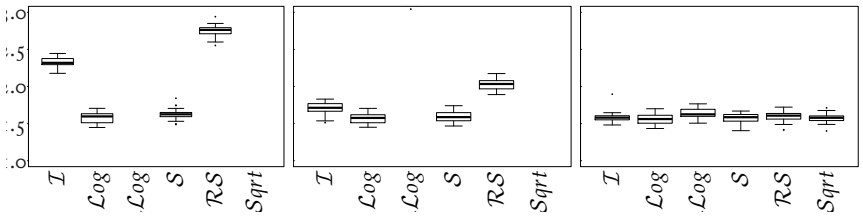

Figure B.5: APD (given on y-axis) obtained by the different transformations for a high temperature length. The three plots show the results obtained, respectively, with a budget of 125, 500 and 2 000 experiments.

$\mathcal{S}qrt(x)$, $\mathcal{RS}(x)$ and $\mathcal{RL}og(x)$ obtain much worse results. These latter ones require a large number of experiments (here 2 000) to converge to good values. A more detailed examination shows that $\mathcal{L}og(x)$ and $\mathcal{S}(x)$ converge slightly faster than $\mathcal{I}(x)$ to the best values; however, here the differences are not very marked, as the range of very good values appears to be somewhat larger as for the case with low temperature length and the variability between similar values larger.

B.3.2 Simulated annealing with a fixed temperature

In this example, we study a fixed temperature variant of the original SA formulation [207, 208], where the temperature is held constant during the whole runtime. The cooling scheme is therefore replaced by the formula

$$T_{k+1} = T_k = T_0 \quad \forall\, k, \tag{B.9}$$

where T_0 is the initial temperature. The acceptance probability of a worsening move depends therefore only on Δ. Hence, the only numerical parameter to be considered is T_0.

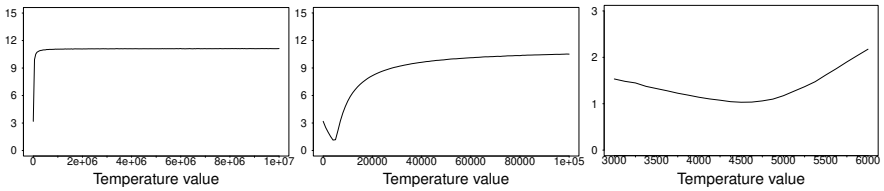

FIGURE B.6: Landscape of the results obtained by different fixed temperature values, in terms of APD (given on y-axis). Given are respectively, the full landscape, the landscape of the results obtained in the range $(0, 10\,000)$ (the lowest 1% of possible values) of the fixed temperature, and the subinterval close to optimal parameter values.

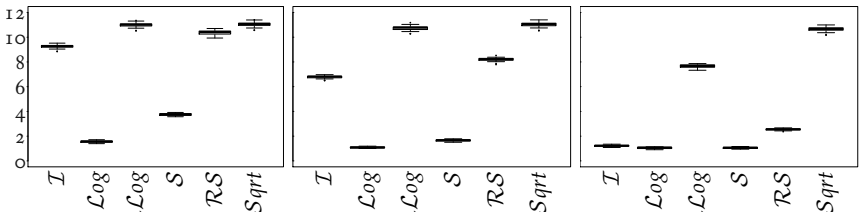

FIGURE B.7: APD values obtained by the different transformations. The three plots show the results obtained with a budget of 125, 500 and 2000 experiments, respectively.

We allow the temperature T_0 to take any integer value in the area $[1, 10\,000\,000]$, which translates in an interval $[0, 7]$ for the logarithmic transformations $\mathcal{L}og(x)$ and $\mathcal{RL}og(x)$. In our instance set, the values that give the best results lie in a very narrow range close to the lower part of the interval, as shown in the plots of Figure B.6. In this context, we expect the logarithmic transformation $\mathcal{L}og(x)$, and the quadratic one $\mathcal{S}(x)$, to outperform their counterparts, especially in the low budget case.

The results of the five transformations are shown in Figure B.7, giving the results for the 30 tunings for each of the budgets of 125, 500 and 2000 experiments. As expected, the logarithmic transformation outperforms all the other ones, with very good results already with the smallest budget, followed by the quadratic transformation. The identity transformation is only able to catch up for the largest available interval.

B.3.3 Discussion

In Sections B.3.1.2 and B.3.2, we have shown two experiments in which the optimal parameter values lie in the lower part of the allowed range. In this case, the transformations that are designed to exploit that specific area, namely the logarithmic transformation $\mathcal{L}og(x)$ and the square transformation $\mathcal{S}(x)$, also show a faster convergence towards good values with respect to the identity transformation. Analogously, in Section B.3.1.1 we see how their bounded reflections, namely the transformations $\mathcal{RL}og(x)$ and $\mathcal{RS}(x)$, outperform the identity transformation. The differences in convergence speed are more marked in the case of a large range of possible parameter values, as shown in Section B.3.2, where the identity transformation needs eight times the budget needed by the logarithmic transformation to reach comparable results, even if only a single parameter is tuned.

However, a wrong transformation is detrimental for the tuning process. This is illustrated by the example in Section B.3.1.2. Traditionally, the cooling coefficient α of the geometric cooling in Simulated Annealing is set to values close to 1, but in the example of Section B.3.1.2 the best-performing values are actually close to zero. While this experiment is somewhat artificial, it is more important to note that in absolute terms, well-tuned parameter settings from the example in Section B.3.1.2 perform better than well-tuned parameter settings from the example in Section B.3.1.1. In fact, the solutions obtained are on average 1.55% and 1.59% from the best known solutions, respectively; while small, a pairwise Wilcoxon test shows that this difference is actually statistically significant. In a sense, while the definition and application of transformations is often intuitive if the parameter landscape is known, our results show that inaccurate assumptions may lead to wrong choices of transformations, and be harmful to the quality of the final results. Note that this does not only happen in case of artificial examples. In the supplementary material at `http://iridia.ulb.ac.be/supp/IridiaSupp2017-011` we also include some experiments with the SMAC configurator version 2.10 [138] applied to configure various scenarios that include 26 to 323 parameters of which for several numerical parameters a logarithmic transformation was manually chosen based on a priori information. However, our experimental results indicate that this manual choice of the transformation actually does not results in improved performance of SMAC, confirming that manually picking the transformations does not necessarily improve performance.

Hence, it is desirable to unburden the user of the choice of the right transformation to apply. In the context of Bayesian optimization (or, say, surrogate-assisted optimization), Snoek *et al.* proposed an automatic mech-

anism to adapt a parameter transformation [373], applying a Beta transformation $\textsc{Beta}(x;\alpha,\beta)$ to a numerical parameter x, where α and β are the control parameters that define the shape of the transformation. The control parameters α and β are in turn adapted using the same Bayesian optimization approach that is used also for parameter tuning. Interestingly, in [373] the authors also report observations about surprising choices of the right transformation to apply, which may contrast with intuition and established practices. As irace is not a method that works within a Bayesian optimization scheme, in the next section we propose a simple, heuristic scheme that is suitable for use in irace.

B.4 Heuristic transformation in irace

The irace package iteratively samples configurations that then undergo a racing process to identify the best performing candidates. We propose a method to automatically adapt the proper transformation when sampling numerical parameters in irace. Here, we consider only $\mathcal{L}og(x)$ and $\mathcal{RL}og(x)$ transformations alongside $\mathcal{I}(x)$, as they have dominated the corresponding polynomial transformations. Our idea is to sample each numerical parameter in the first iteration according to a uniform random sampling as done by default in irace. After the results of the first iteration are available we simulate a sampling according to the three possibilities for the following races. The simulated sampling of numerical parameters is done as follows. First, for each parameter value appearing in the configurations an estimate of the mean performance is computed as the average deviation from the best result across the already seen instances. These estimates are saved in an array, which is sorted according to the parameter values. Next, three subsets of results are extracted from this array by choosing positions that (i) simulate the generation of values close to the lower part of the parameter range (as it would happen when using $\mathcal{L}og(x)$), (ii) simulate the generation of values close to the upper part of the parameter range (as it would happen when using $\mathcal{RL}og(x)$), and (iii) simulate a sampling according to an identity transformation (that is, no transformation). If the mean of the first subset is significantly better than the others, $\mathcal{L}og(x)$ is applied; if the mean of the second subset is significantly better than the others, $\mathcal{RL}og(x)$ is used. Otherwise, no transformation is applied. This process is repeated for every new candidate configuration, that is, the same parameter may be subject to different transformations for different configurations.

This method is quite simple but effective, as we show in the next examples. In Figure B.8 we show the results obtained by this automatic trans-

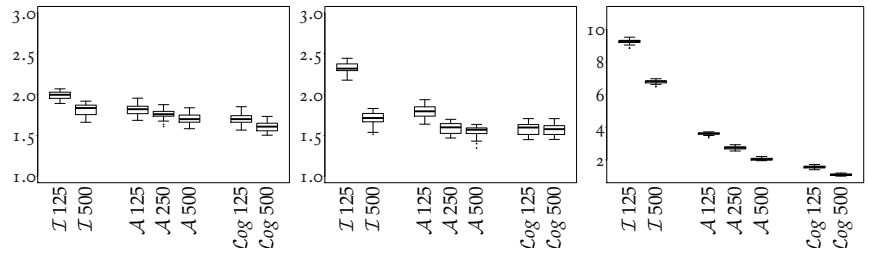

Figure B.8: Comparison of the convergence observed for the automatic transformation selection described in Section B.4 for, left to right, experiments from Sections B.3.1.1, B.3.1.2 and B.3.2, respectively. We compare the results in terms of average percentage deviation from the best known solutions (given on y-axis) for $\mathcal{I}(x)$, the automatic selection, and the "right" transformation using a budget of 125 and 500 evaluations (for the adaptive variant an additional budget of 250 evaluations is considered).

formation selection on the scenarios we considered in the experiments of Sections B.3.1.1, B.3.1.2 and B.3.2, respectively. For each benchmark we compare the automatically chosen transformation for a budget of 125, 250 and 500 experiments per tuning (the three central boxplots) with $\mathcal{I}(x)$ with a budget of 125 and 500 (leftmost boxplots) and the best transformations ($\mathcal{RL}og(x)$, $\mathcal{L}og(x)$ and $\mathcal{L}og(x)$, respectively) for each of these cases with a budget of 125 and 500 (rightmost boxplots). In all cases, the automatic selection scheme improves clearly over $\mathcal{I}(x)$ when considering the same budget. The automatic scheme also catches up with the a priori chosen best possible transformation at a budget of 125, though still lagging behind the results obtained with the "right" transformation at a budget of 500. While in low budget scenarios we cannot expect results as good as applying the right transformation from the beginning, the loss in terms of performance is limited.

In the following experiments we compare the automatic selection only with $\mathcal{I}(x)$, that is, the original sampling scheme in irace. The results for both of them are reportd in Figure B.9. In the first one (left plot) we show the results obtained when using a linear cooling scheme for an SA algorithm for budgets of 125, 250 and 500 experiments per tuning. As in the previous experiments, only the cooling scheme coefficient is tuned, chosen in the range $[0, 100000]$; the temperature length is one times the size of the neighbourhood. While $\mathcal{I}(x)$ with the given budget cannot make significant improvements, the automatic scheme obtains better results already with the lowest budget, and continues improving as more experiments are allowed.

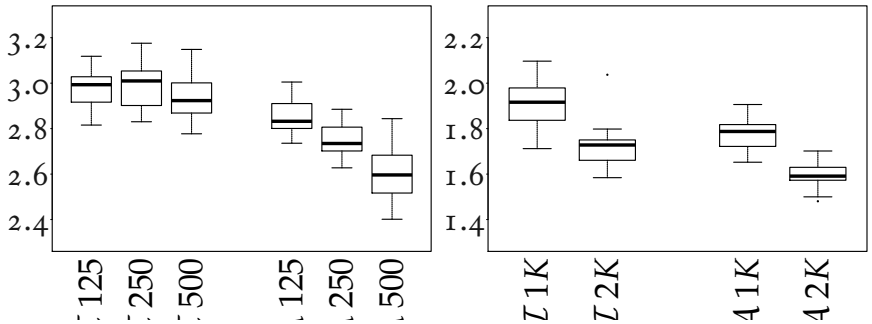

FIGURE B.9: Comparison of the convergence observed for the automatic transformation selection for a linear cooling scheme (left plot) and tuning an SA algorithm with four parameters (right plot). We compare the results in terms of average percentage deviation from the best known solutions APD (given on y-axis) of the automatic scheme with those of $\mathcal{I}(x)$.

In the second experiment (right plot) we show a more realistic scenario, tuning the SA algorithm for the QAP proposed in [215], with budgets of 500 and 2000 experiments. This SA algorithm has four numerical parameters: the α coefficient for the geometric cooling scheme in the range $[0, 1]$; the temperature length coefficient in the range $[1, 100]$ (proportional to the size of the neighbourhood); the initial value for the temperature is chosen as a the the solution cost of a randomly generated solution rescaled by a value in the range $[0, 10]$; the temperature resets to its initial value when it reaches a certain minimum value, chosen in the range $[1, 10000]$. In this case, the improvement is not as strong as in the other examples reported, because of the combined effect of different parameters. Nonetheless, for both budgets SA algorithm tuned using the automatic transformation selection obtains better performance than the default irace.

In general, tuning an algorithm with several parameters is a complex task, with possibly surprising outcomes and also an expert may be misled by wrong assumptions and intuitions, with possibly detrimental effects on the search. Our proposed automatic selection scheme for parameter transformations relieves the user from the possibly very detrimental effect of a wrong choice. Additionally, as shown in the example above, it helps in the cases where a transformation is useful to speed-up convergence to good parameter values when compared to not using any transformation. Contrarily to [373], our scheme does not require to specify a prior, as it makes its decisions solely based on the data observed in the already executed experiments. In summary, our automatic selection scheme is not meant to replace expert

knowledge, or to prevent users from making decisions, but rather to provide an optional, robust data-driven support when said decisions are difficult to make.

B.5 Conclusions

Parameter space transformations have been considered before, but their impact on automatic configuration techniques has not been studied explicitly in the optimization community, unlike in other research communities such as medicine [374, 375] or machine learning [373]. The results reported in this chapter illustrate the usefulness of knowledge about the parameter space in practice. Tuning a single parameter is a seemingly trivial task, but choosing the right transformation can make this task even less computationally expensive. This is highly desirable, as the hardness of the tuning task grows strongly with the number of parameters. However, a wrong transformation is detrimental for the tuning process, as we have also illustrated with several examples in this chapter. In a sense, while the definition and application of transformations is often intuitive if the parameter landscape is known, our results show that inaccurate assumptions may lead to wrong choices of transformations, and be harmful for the quality of the final results. Hence, our recommendation would be to apply transformations of the parameter domain only if a careful deliberation indicates that such transformation may be useful as the best parameter values are expected to be at the boundary of a parameter domain. As an alternative one may apply adaptive schemes that choose appropriate transformations during the run of a configuration algorithm. Such approaches have previously been proposed in the context of Bayesian optimization [373]. In this chapter, we have proposed another, heuristic strategy to choose an appropriate transformation at run-time. We have shown that on various of the configuration scenarios we have considered in this chapter, the proposed strategy is useful and improves performance when compared to not applying any transformation at all.

Bibliography

[1] R. E. Steuer, Multiple Criteria Optimization: Theory, Computation and Application, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, New York, NY, 1986.

[2] D. Bertsimas, N. Kallus, From predictive to prescriptive analytics, Management Science 66 (3) (2020) 1025–1044.

[3] N. C. Jones, P. A. Pevzner, An introduction to bioinformatics algorithms, MIT Press, Cambridge, MA, 2004.

[4] S. Sra, S. Nowozin, S. J. Wright, Optimization for machine learning, MIT Press, Cambridge, MA, 2012.

[5] B. Balcik, B. M. Beamon, Facility location in humanitarian relief, International Journal of Logistics 11 (2) (2008) 101–121.

[6] A. M. Campbell, P. C. Jones, Prepositioning supplies in preparation for disasters, European Journal of Operational Research 209 (2) (2011) 156–165.

[7] A. M. Caunhye, X. Nie, S. Pokharel, Optimization models in emergency logistics: A literature review, Socio-Economic Planning Sciences 46 (1) (2012) 4–13.

[8] J. O. Berkey, P. Y. Wang, Two-dimensional finite bin-packing algorithms, Journal of the Operational Research Society 38 (5) (1987) 423–429. doi:10.2307/2582731.

[9] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: A survey, European Journal of Operational Research 141 (2) (2002) 241–252. doi:10.1016/S0377-2217(02)00123-6.

[10] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.

[11] R. Martí, P. M. Pardalos, M. G. C. Resende (Eds.), Handbook of Heuristics, Springer International Publishing, 2018.

[12] H. H. Hoos, T. Stützle, Stochastic Local Search—Foundations and Applications, Morgan Kaufmann Publishers, San Francisco, CA, 2005.

[13] W. J. Cook, Computing in combinatorial optimization, in: B. Steffen, G. Woeginger (Eds.), Computing and Software Science: State of the Art and Perspectives, Vol. 10000 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2019, pp. 27–47. doi:10.1007/978-3-319-91908-9_3.

[14] K. Sörensen, M. Sevaux, F. Glover, A history of metaheuristics, in: Martí et al. [11], pp. 1–27.

[15] D. Sculley, J. Snoek, A. Rahimi, A. B. Wiltschko, Winner's curse? on pace, progress and empirical rigor, in: I. Murray, M. Ranzato, O. Vinyals (Eds.), 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings, OpenReview.net, 2018, pp. 1–4.
URL https://openreview.net/pdf?id=rJWF0Fywf

[16] A. Franzin, T. Stützle, Revisiting simulated annealing: A component-based analysis, Computers & Operations Research 104 (2019) 191–206. doi:10.1016/j.cor.2018.12.015.

[17] A. Franzin, T. Stützle, Comparison of acceptance criteria in randomized local searches, in: E. Lutton, P. Legrand, P. Parrend, N. Monmarché, M. Schoenauer (Eds.), EA 2017: Artificial Evolution, Vol. 10764 of Lecture Notes in Computer Science, Springer, Heidelberg, 2017, pp. 16–29.

[18] A. Franzin, T. Stützle, A landscape-based analysis of fixed temperature and simulated annealing, Tech. Rep. TR/IRIDIA/2021-005, IRIDIA, Université Libre de Bruxelles, Belgium (2021).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2021-005.pdf

[19] A. Franzin, T. Stützle, A causal framework for understanding optimisation algorithms, in: F. Heintz, M. Milano, B. O'Sullivan (Eds.), Trustworthy AI – Integrating Learning, Optimization and Reasoning. TAILOR 2020, Vol. 12641 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2020, pp. 140–145.

[20] A. Franzin, T. Stützle, Towards transferring algorithm configurations across problems, in: M. Vlastelica, J. Song, A. Ferber, B. Amos, G. Martius, B. Dilkina, Y. Yue (Eds.), Learning Meets Combinatorial Algorithms Workshop at NeurIPS 2020, LMCA 2020, Vancouver, Canada, December 12, 2020, 2020, pp. 1–6.

[21] A. Franzin, T. Stützle, Exploration of metaheuristics through automatic algorithm configuration techniques and algorithmic frameworks, in: T. Friedrich, F. Neumann, A. M. Sutton (Eds.), Proceedings of the Genetic

and Evolutionary Computation Conference Companion, GECCO Companion 2016, ACM Press, New York, NY, 2016, pp. 1341–1347.

[22] A. Franzin, L. Pérez Cáceres, T. Stützle, Effect of transformations of numerical parameters in automatic algorithm configuration, Optimization Letters 12 (8) (2018) 1741–1753. doi:10.1007/s11590-018-1240-3.

[23] L. Pérez Cáceres, F. Pagnozzi, A. Franzin, T. Stützle, Automatic configuration of GCC using irace, in: E. Lutton, P. Legrand, P. Parrend, N. Monmarché, M. Schoenauer (Eds.), EA 2017: Artificial Evolution, Vol. 10764 of Lecture Notes in Computer Science, Springer, Heidelberg, 2017, pp. 202–216. doi:10.1007/978-3-319-78133-4_15.

[24] M. Silva-Muñoz, A. Franzin, H. Bersini, Automatic configuration of the cassandra database using irace, PeerJ Computer Science 7 (2021) e634. doi:10.7717/peerj-cs.634.

[25] M. Silva-Muñoz, G. Calderon, A. Franzin, H. Bersini, Determining a consistent experimental setup for benchmarking and optimizing databases, in: F. Chicano, K. Krawiec (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2021, ACM Press, New York, NY, 2021, pp. 1614–1621. doi:10.1145/3449726.3463180.

[26] A. Franzin, F. Sambo, B. Di Camillo, bnstruct: an R package for Bayesian network structure learning in the presence of missing data, Bioinformatics 33 (8) (2016) 1250–1252.

[27] F. Sambo, B. Di Camillo, A. Franzin, A. Facchinetti, L. Hakaste, J. Kravic, G. Fico, J. Tuomilehto, L. Groop, R. Gabriel, T. Tuomi, C. Cobelli, A bayesian network analysis of the probabilistic relations between risk factors in the predisposition to type 2 diabetes, in: N. Lovell, L. Mainardi (Eds.), 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2015, Proceedings, Annual International Conference of the IEEE Engineering in Medicine and Biology, IEEE Press, 2015, pp. 2119–2122.

[28] A. Franzin, R. Gyory, J.-C. Nadé, G. Aubert, G. Klenkle, H. Bersini, Philéas: Anomaly detection for IoT monitoring, in: L. Cao, W. Kosters, J. Lijffijt (Eds.), Proceedings of the 32nd Benelux Conference on Artificial Intelligence, BNAIC 2020, Leiden, The Netherlands, 19-20 November 2020, 2020, pp. 56–70.
URL https://bnaic.liacs.leidenuniv.nl/wordpress/wp-content/uploads/bnaic2020proceedings.pdf

[29] D. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton University Press, Princeton, NJ, 2006.

[30] W. J. Cook, The traveling salesman problem, http://www.math.uwaterloo.ca/tsp, version visited last on 15 April 2014 (2010).

[31] P. Toth, D. Vigo, The vehicle routing problem, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[32] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, A unified solution framework for multi-attribute vehicle routing problems, European Journal of Operational Research 234 (3) (2014) 658–673.

[33] M. W. P. Savelsbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1) (1985) 285–305. doi:10.1007/BF02022044.

[34] J. Cirasella, D. S. Johnson, L. A. McGeoch, W. Zhang, The asymmetric traveling salesman problem: Algorithms, instance generators, and tests, in: A. L. Buchsbaum, J. Snoeyink (Eds.), Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, Washington, DC, USA, January 5-6, 2001, Revised Papers, Vol. 2153 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2001, pp. 32–59. doi:10.1007/3-540-44808-X_3.

[35] W. J. Cook, In Pursuit of the Traveling Salesman, Princeton University Press, Princeton, NJ, 2012.

[36] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization – Algorithms and Complexity, Prentice Hall, Englewood Cliffs, NJ, 1982.

[37] A. Langevin, F. Soumis, J. Desrosiers, Classification of travelling salesman problem formulations, Operations Research Letters 9 (2) (1990) 127–132.

[38] S. Arora, B. Barak, Computational complexity: a modern approach, Cambridge University Press, 2009.

[39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms, MIT Press, Cambridge, MA, 2009.

[40] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical society 7 (1) (1956) 48–50.

[41] R. C. Prim, Shortest connection networks and some generalizations, Bell System Technical Journal 36 (6) (1957) 1389–1401.

[42] H. W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1–2) (1955) 83–97.

[43] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman & Co, San Francisco, CA, 1979.

[44] S. A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, ACM, 1971, pp. 151–158. doi:10.1145/800157.805047.

[45] L. Levin, Universal'nyie perebornyie zadachi, Problemy Peredachi Informatsii 9 (1973) 265–266.

[46] R. M. Karp, Reducibility among combinatorial problems, in: R. E. Miller, W. Thatcher, James (Eds.), Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, The IBM Research Symposia Series, Springer, 1972, pp. 85–103.

[47] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: R. A. DeMillo (Ed.), Proceedings of the sixteenth annual ACM Symposium on Theory of Computing, ACM Press, 1984, pp. 302–311.

[48] D. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook, D. Espinoza, M. Goycoolea, K. Helsgaun, Certification of an optimal TSP tour through 85,900 cities, Operations Research Letters 37 (1) (2009) 11–15.

[49] M. Fischetti, M. Monaci, D. Salvagnin, Three ideas for the quadratic assignment problem, Operations Research 60 (4) (2012) 954–964.

[50] D. A. Spielman, S.-H. Teng, Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time, Journal of the ACM 51 (3) (2004) 385–463.

[51] T. C. Koopmans, M. J. Beckmann, Assignment problems and the location of economic activities, Econometrica 25 (1957) 53–76.

[52] R. E. Burkard, E. Çela, P. M. Pardalos, L. S. Pitsoulis, The quadratic assignment problem, in: P. M. Pardalos, D.-Z. Du (Eds.), Handbook of Combinatorial Optimization, Vol. 2, Kluwer Academic Publishers, 1998, pp. 241–338.

[53] M. R. Garey, D. S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research 1 (1976) 117–129.

[54] J. M. Framiñán, R. Leisten, R. Ruiz, Manufacturing Scheduling Systems: An Integrated View on Models, Methods, and Tools, Springer, New York, NY, 2014.

[55] M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 4th Edition, Springer, New York, NY, 2012.

[56] V. Fernandez-Viagas, R. Ruiz, J. M. Framiñán, A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation, European Journal of Operational Research 257 (3) (2017) 707–721.

[57] Q.-K. Pan, R. Ruiz, A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime, Computers & Operations Research 40 (1) (2013) 117–128.

[58] Q.-K. Pan, R. Ruiz, Local search methods for the flowshop scheduling problem with flowtime minimization, European Journal of Operational Research 222 (1) (2012) 31–43.

[59] A. H. Land, A. G. Doig, An automatic method of solving discrete programming problems, Econometrica 28 (3) (1960) 497–520.

[60] E. L. Lawler, D. E. Wood, Branch-and-bound methods: A survey, Operations Research 14 (4) (1966) 699–719. doi:10.1287/opre.14.4.699.

[61] R. E. Gomory, An algorithm for integer solutions to linear programs, in: R. Graves, P. Wolfe (Eds.), Recent Advances in Mathematical Programming, McGraw Hill, New York, NY, 1963, pp. 260–302.

[62] H. Marchand, A. Martin, R. Weismantel, L. Wolsey, Cutting planes in integer and mixed integer programming, Discrete Applied Mathematics 123 (1–3) (2002) 397–446.

[63] M. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33 (1) (1991) 60–100.

[64] T. Achterberg, SCIP: Solving constraint integer programs, Mathematical Programming Computation 1 (1) (2009) 1–41.

[65] IBM, ILOG CPLEX optimizer, http://www.ibm.com/software/integration/optimization/cplex-optimizer/ (2017).

[66] Gurobi, Gurobi optimizer, http://www.gurobi.com/products/gurobi-optimizer (2017).

[67] M. Held, R. M. Karp, The traveling-salesman problem and minimum spanning trees, Operations Research 18 (6) (1970) 1138–1162.

[68] D. P. Williamson, D. B. Shmoys, The design of approximation algorithms, Cambridge University Press, 2011.

[69] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Tech. Rep. 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA (1976).

[70] G. Gutin, A. Yeo, A. Zverovich, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP, Discrete Applied Mathematics 117 (1–3) (2002).

[71] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Operations Research 21 (2) (1973) 498–516.

[72] K. Helsgaun, An effective implementation of the Lin-Kernighan traveling salesman heuristic, European Journal of Operational Research 126 (2000) 106–130.

[73] M. M. Flood, The travelling salesman problem, Operations Research 4 (1956) 61–75.

[74] G. A. Croes, A method for solving traveling salesman problems, Operations Research 6 (1958) 791–812.

[75] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Vol. 2, Prentice Hall, Englewood Cliffs, NJ, 2003.

[76] A. Lodi, A. Tramontani, Performance variability in mixed-integer programming, in: H. Topaluglu (Ed.), Theory Driven by Influential Applications, INFORMS, 2013, pp. 1–12.

[77] M. Fischetti, M. Monaci, Exploiting erraticism in search, Operations Research 62 (1) (2014) 114–122. doi:10.1287/opre.2013.1231.

[78] H. R. Lourenço, O. Martin, T. Stützle, Iterated local search: Framework and applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Vol. 272 of International Series in Operations Research & Management Science, Springer, 2019, Ch. 5, pp. 129–168. doi:10.1007/978-3-319-91086-4_5.

[79] T. Stützle, R. Ruiz, Iterated local search, in: Martí et al. [11], pp. 1–27. doi:10.1007/978-3-319-07153-4_8-1.

[80] F. Pagnozzi, T. Stützle, Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems, European Journal of Operational Research 276 (2019) 409–421. doi:10.1016/j.ejor.2019.01.018.

[81] F. Pagnozzi, T. Stützle, Evaluating the impact of grammar complexity in automatic algorithm design, International Transactions in Operational Research (2020) 1–26doi:10.1111/itor.12902.

[82] T. Stützle, R. Ruiz, Iterated greedy, in: Martí et al. [11], pp. 1–31. doi:10.1007/978-3-319-07153-4_10-1.

[83] U. Benlic, J.-K. Hao, Breakout local search for the quadratic assignment problem, Applied Mathematics and Computation 219 (9) (2013) 4800–4815.

[84] P. Morris, The breakout method for escaping from local minima, in: R. Fikes, W. G. Lehnert (Eds.), Proceedings of the 11th National Conference on Artificial Intelligence, AAAI Press/MIT Press, Menlo Park, CA, 1993, pp. 40–45.

[85] T. A. Feo, M. G. C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Operations Research Letters 8 (2) (1989) 67–71.

[86] T. A. Feo, M. G. C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (2) (1995) 109–113.

[87] T. Bäck, Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford University Press, 1996.

[88] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[89] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Ph.D. thesis, Department of Process Engineering, Technical University of Berlin (1971).

[90] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann-Holzboog, Stuttgart, Germany, 1973.

[91] H.-P. Schwefel, Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie, Birkhäuser, Basel, Switzerland, 1977.

[92] H.-G. Beyer, H.-P. Schwefel, Evolution strategies: A comprehensive introduction, Natural Computing 1 (2002) 3–52.

[93] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[94] K. Price, R. M. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, New York, NY, 2005. doi:10.1007/3-540-31306-0.

[95] P. Moscato, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Caltech Concurrent Computation Program, C3P Report 826, Caltech (1989).

[96] P. Moscato, Memetic algorithms: a short introduction, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw Hill, London, UK, 1999, pp. 219–234.

[97] A. Jaszkiewicz, H. Ishibuchi, Q. Zhang, Multiobjective memetic algorithms, in: F. Neri, C. Cotta, P. Moscato (Eds.), Handbook of Memetic Algorithms, Vol. 379 of Studies in Computational Intelligence, Springer, 2011, pp. 201–217.

[98] Y. Nagata, S. Kobayashi, A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem, INFORMS Journal on Computing 25 (2) (2013) 346–363. doi:10.1287/ijoc.1120.0506.

[99] M. Dorigo, Optimization, learning and natural algorithms, Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, in Italian (1992).

[100] M. Dorigo, V. Maniezzo, A. Colorni, Ant System: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1) (1996) 29–41.

[101] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.

[102] M. Dorigo, M. Birattari, T. Stützle, Ant colony optimization: Artificial ants as a computational intelligence technique, IEEE Computational Intelligence Magazine 1 (4) (2006) 28–39.

[103] M. Dorigo, L. M. Gambardella, Ant Colony System: A cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 53–66.

[104] T. Stützle, H. H. Hoos, Improving the Ant System: A detailed report on the *MAX–MIN* Ant System, Tech. Rep. AIDA–96–12, FG Intellektik, FB Informatik, TU Darmstadt, Germany (Aug. 1996).

[105] T. Stützle, H. H. Hoos, *MAX–MIN* Ant System, Future Generation Computer Systems 16 (8) (2000) 889–914.

[106] K. Socha, M. Dorigo, Ant colony optimization for continuous domains, European Journal of Operational Research 185 (3) (2008) 1155–1173. doi:10.1016/j.ejor.2006.06.046.

[107] R. C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995, pp. 39–43.

[108] M. Fischetti, A. Lodi, Local branching, Mathematical Programming Series B 98 (2003) 23–47.

[109] M. Fischetti, M. Monaci, Proximity search for 0-1 mixed-integer convex programming, Journal of Heuristics 20 (6) (2014) 709–731.

[110] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, Mathematical Programming 102 (1) (2005) 71–90.

[111] M. Fischetti, F. Glover, A. Lodi, The feasibility pump, Mathematical Programming 104 (1) (2005) 91–104.

[112] T. Achterberg, T. Berthold, Improving the feasibility pump, Discrete Optimization 4 (1) (2007) 77–86.

[113] L. Bertacco, M. Fischetti, A. Lodi, A feasibility pump heuristic for general mixed-integer problems, Discrete Optimization 4 (1) (2007) 63–76.

[114] M. Fischetti, D. Salvagnin, Feasibility pump 2.0, Mathematical Programming Computation 1 (2–3) (2009) 201–222.

[115] E. Rothberg, An evolutionary algorithm for polishing mixed integer programming solutions, INFORMS Journal on Computing 19 (4) (2007) 534–541.

[116] J. R. Rice, The algorithm selection problem, Advances in Computers 15 (1976) 65–118.

[117] L. Kotthoff, Algorithm selection for combinatorial search problems: A survey, AI Magazine 35 (3) (2014) 48–60.

[118] P. Kerschke, H. H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives, Evolutionary Computation 27 (1) (2019) 3–45.

[119] J. Pihera, N. Musliu, Application of machine learning to algorithm selection for TSP, in: G. A. Papadopoulos (Ed.), 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014, IEEE Press, 2014, pp. 47–54.

[120] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, SATzilla: portfolio-based algorithm selection for SAT, Journal of Artificial Intelligence Research 32 (2008) 565–606. doi:10.1613/jair.2490.

[121] B. Bischl, O. Mersmann, H. Trautmann, M. Preuss, Algorithm selection based on exploratory landscape analysis and cost-sensitive learning, in: T. Soule, J. H. Moore (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2012, ACM Press, New York, NY, 2012, pp. 313–320.

[122] M. A. Muñoz, Y. Sun, M. Kirley, S. K. Halgamuge, Algorithm selection for black-box continuous optimization problems: a survey on methods and challenges, Information Sciences 317 (2015) 224–245.

[123] P. Kerschke, H. Trautmann, The R-package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems, in: Proceedings of the 2016 Congress on Evolutionary Computation (CEC 2016), IEEE Press, Piscataway, NJ, 2016, pp. 5262–5269. doi:10.1109/CEC.2016.7748359.

[124] P. Kerschke, H. Trautmann, Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning, Evolutionary Computation 27 (1) (2019) 99–127. doi:10.1162/evco_a_00236.

[125] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown, Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, in: I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman, R. Uthurusamy (Eds.), The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, ACM Press, New York, NY, 2013, pp. 847–855.

[126] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, K. Leyton-Brown, Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA, Journal of Machine Learning Research 17 (2016) 1–5.

[127] R. Battiti, G. Tecchiolli, The reactive tabu search, ORSA Journal on Computing 6 (2) (1994) 126–140.

[128] R. Battiti, M. Brunato, F. Mascia, Reactive Search and Intelligent Optimization, Vol. 45 of Operations Research/Computer Science Interfaces, Springer, New York, NY, 2008. doi:10.1007/978-0-387-09624-7.

[129] J. S. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (2012) 281–305.

[130] G. S. Peace, Taguchi Methods: A Hands-On Approach, Addison-Wesley, 1993.

[131] L. Pronzato, W. G. Müller, Design of computer experiments: space filling and beyond, Statistics and Computing 22 (3) (2012) 681–701.

[132] V. Nannen, A. E. Eiben, A method for parameter calibration and relevance estimation in evolutionary algorithms, in: M. Cattolico, et al. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006, ACM Press, New York, NY, 2006, pp. 183–190. doi:10.1145/1143997.1144029.

[133] C. Audet, D. Orban, Finding optimal algorithmic parameters using derivative-free optimization, SIAM Journal on Optimization 17 (3) (2006) 642–664.

[134] Z. Yuan, M. A. Montes de Oca, T. Stützle, M. Birattari, Continuous optimization algorithms for tuning real and integer algorithm parameters of swarm intelligence algorithms, Swarm Intelligence 6 (1) (2012) 49–75.

[135] J. S. Bergstra, D. Yasmin, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures,

in: S. Dasgupta, D. McAllester (Eds.), Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Vol. 28, 2013, pp. 115–123.
URL http://jmlr.org/proceedings/papers/v28/

[136] B. Adenso-Díaz, M. Laguna, Fine-tuning of algorithms using fractional experimental design and local search, Operations Research 54 (1) (2006) 99–114.

[137] T. Bartz-Beielstein, O. Flasch, P. Koch, W. Konen, SPOT: A toolbox for interactive and automatic tuning in the R environment, in: Proceedings 20. Workshop Computational Intelligence, KIT Scientific Publishing, Karlsruhe, 2010, pp. 264–273.

[138] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: C. A. Coello Coello (Ed.), Learning and Intelligent Optimization, 5th International Conference, LION 5, Vol. 6683 of Lecture Notes in Computer Science, Springer, Heidelberg, 2011, pp. 507–523. doi:10.1007/978-3-642-25566-3_40.

[139] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, K. Tierney, Model-based genetic algorithms for algorithm configuration, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15), IJCAI/AAAI Press, Menlo Park, CA, 2015, pp. 733–739. doi:10.5555/2832249.2832351.

[140] L. Pérez Cáceres, B. Bischl, T. Stützle, Evaluating random forest models for irace, in: P. A. N. Bosman (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2017, ACM Press, New York, NY, 2017, pp. 1146–1153. doi:10.1145/3067695.3082057.

[141] F. Hutter, H. H. Hoos, T. Stützle, Automatic algorithm configuration based on local search, in: R. C. Holte, A. Howe (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press/MIT Press, Menlo Park, CA, 2007, pp. 1152–1157.

[142] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: an automatic algorithm configuration framework, Journal of Artificial Intelligence Research 36 (2009) 267–306. doi:10.1613/jair.2861.

[143] C. Ansótegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: I. P. Gent (Ed.), Principles and Practice of Constraint Programming, CP 2009, Vol. 5732 of Lecture Notes in Computer Science, Springer, Heidelberg, 2009, pp. 142–157. doi:10.1007/978-3-642-04244-7_14.

[144] M.-C. Riff, E. Montero, A new algorithm for reducing metaheuristic design effort, in: Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013), IEEE Press, Piscataway, NJ, 2013, pp. 3283–3290. doi:10.1109/CEC.2013.6557972.

[145] C. Audet, C.-K. Dang, D. Orban, Optimization of algorithms with OPAL, Mathematical Programming Computation 6 (3) (2014) 233–254.

[146] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: W. B. Langdon, et al. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, Morgan Kaufmann Publishers, San Francisco, CA, 2002, pp. 11–18.

[147] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration, Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, published in Operations Research Perspectives [148] (2011).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf

[148] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, M. Birattari, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58. doi:10.1016/j.orp.2016.09.002.

[149] Z. Karnin, T. Koren, O. Somekh, Almost optimal exploration in multi-armed bandits, in: S. Dasgupta, D. McAllester (Eds.), Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Vol. 28, 2013, pp. 1238–1246.
URL http://jmlr.org/proceedings/papers/v28/

[150] K. G. Jamieson, A. Talwalkar, Non-stochastic best arm identification and hyperparameter optimization, in: A. Gretton, C. C. Robert (Eds.), Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016, Vol. 51 of JMLR Workshop and Conference Proceedings, JMLR.org, 2016, pp. 240–248.
URL http://proceedings.mlr.press/v51/jamieson16.html

[151] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, Journal of Machine Learning Research 18 (185) (2018) 1–52.
URL http://jmlr.org/papers/v18/16-558.html

[152] T. Stützle, M. López-Ibáñez, Automated design of metaheuristic algorithms, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Vol. 272 of International Series in Operations Research & Management Science, Springer, 2019, pp. 541–579. doi:10.1007/978-3-319-91086-4_17.

[153] F. Pagnozzi, T. Stützle, Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems, Tech. Rep. TR/IRIDIA/2018-005, IRIDIA, Université Libre de Bruxelles, Belgium (Apr. 2018).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2018-005.pdf

[154] F. Pagnozzi, T. Stützle, Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems with additional constraints, Operations Research Perspectives 8 (2021). doi:10.1016/j.orp.2021.100180.

[155] T. Stützle, ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem (2002).
URL http://www.aco-metaheuristic.org/aco-code

[156] M. López-Ibáñez, T. Stützle, Automatic configuration of multi-objective ACO algorithms, in: M. Dorigo, et al. (Eds.), Swarm Intelligence, 7th International Conference, ANTS 2010, Vol. 6234 of Lecture Notes in Computer Science, Springer, Heidelberg, 2010, pp. 95–106. doi:10.1007/978-3-642-15461-4_9.

[157] M. López-Ibáñez, T. Stützle, The automatic design of multi-objective ant colony optimization algorithms, IEEE Transactions on Evolutionary Computation 16 (6) (2012) 861–875. doi:10.1109/TEVC.2011.2182651.

[158] M. López-Ibáñez, T. Stützle, M. Dorigo, Ant colony optimization: A component-wise overview, in: Martí et al. [11], pp. 371–407. doi:10.1007/978-3-319-07124-4_21.

[159] M. A. Montes de Oca, T. Stützle, M. Birattari, M. Dorigo, Frankenstein's PSO: A composite particle swarm optimization algorithm, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 1120–1132. doi:10.1109/TEVC.2009.2021465.

[160] C. L. Camacho-Villalón, T. Stützle, M. Dorigo, Pso-x: A component-based framework for the automatic design of particle swarm optimization algorithms, Tech. Rep. TR/IRIDIA/2021-002, IRIDIA, Université Libre de Bruxelles, Belgium (2021).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2021-002.pdf

[161] L. C. T. Bezerra, M. López-Ibáñez, T. Stützle, Automatic design of evolutionary algorithms for multi-objective combinatorial optimization, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), PPSN 2014, Vol. 8672 of Lecture Notes in Computer Science, Springer, Heidelberg, 2014, pp. 508–517. doi:10.1007/978-3-319-10762-2_50.

[162] L. C. T. Bezerra, M. López-Ibáñez, T. Stützle, Automatic component-wise design of multi-objective evolutionary algorithms, IEEE Transactions on Evolutionary Computation 20 (3) (2016) 403–417. doi:10.1109/TEVC.2015.2474158.

[163] F. Campelo, L. S. Batista, C. Aranha, The MOEADr package: A component-based framework for multiobjective evolutionary algorithms based on decomposition, Journal of Statistical Software 92 (2020). doi:10.18637/jss.v092.i06.

[164] L. C. T. Bezerra, M. López-Ibáñez, T. Stützle, To DE or not to DE? Multi-objective differential evolution revisited from a component-wise perspective, in: A. Gaspar-Cunha, C. H. Antunes, C. A. Coello Coello (Eds.), Evolutionary Multi-criterion Optimization, EMO 2015 Part I, Vol. 9018 of Lecture Notes in Computer Science, Springer, Heidelberg, 2015, pp. 48–63. doi:10.1007/978-3-319-15934-8_4.

[165] L. Di Gaspero, A. Schaerf, EasyLocal++: An object-oriented framework for flexible design of local search algorithms, Software — Practice & Experience 33 (8) (2003) 733–765. arXiv:http://www.diegm.uniud.it/satt/papers/DiSc03.pdf.

[166] A. Liefooghe, L. Jourdan, E.-G. Talbi, A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO, European Journal of Operational Research 209 (2) (2011) 104–112.

[167] J. Humeau, A. Liefooghe, E.-G. Talbi, S. Verel, ParadisEO-MO: From fitness landscape analysis to efficient local search algorithms, Journal of Heuristics 19 (6) (2013) 881–915. doi:10.1007/s10732-013-9228-8.

[168] A. J. Nebro, J. J. Durillo, M. Vergne, Redesigning the jMetal multi-objective optimization framework, in: J. L. Jiménez Laredo, S. Silva, A. I. Esparcia-Alcázar (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2015, ACM Press, New York, NY, 2015, pp. 1093–1100.

[169] A. Aziz-Alaoui, C. Doerr, J. Dréo, Towards large scale automated algorithm design by integrating modular benchmarking frameworks, in: F. Chicano, K. Krawiec (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2021, ACM Press, New York, NY, 2021, pp. 1365–1374. doi:10.1145/3449726.3463155.

[170] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness, in: P. M. Pardalos, G. Nicosia (Eds.), Learning and Intelligent Optimization, 7th International Conference, LION 7, Vol. 7997 of Lecture Notes in Computer Science, Springer, Heidelberg, 2013, pp. 321–334. doi:10.1007/978-3-642-44973-4_36.

[171] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools, Computers & Operations Research 51 (2014) 190–199. doi:10.1016/j.cor.2014.05.020.

[172] F. Pagnozzi, Automatic design of hybrid stochastic local search algorithms, Ph.D. thesis, IRIDIA, École polytechnique, Université Libre de Bruxelles, Belgium (2019).

[173] J. Koza, Genetic Programming: On the Programming of Computers By the Means of Natural Selection, MIT Press, Cambridge, MA, 1992.

[174] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Transactions on Evolutionary Computation 5 (4) (2001) 349–358.

[175] P. Ross, Hyper-heuristics, in: E. K. Burke, G. Kendall (Eds.), Search Methodologies, Springer, Boston, MA, 2005, pp. 529–556. doi:10.1007/0-387-28356-0_17.

[176] E. K. Burke, M. Gendreau, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724.

[177] H. Stegherr, M. Heider, J. Hähner, Classifying metaheuristics: Towards a unified multi-level classification system, Natural Computing (2020). doi:10.1007/s11047-020-09824-0.

[178] D. Weyland, A critical analysis of the harmony search algorithm: How not to solve Sudoku, Operations Research Perspectives 2 (2015) 97–105.

[179] C. L. Camacho-Villalón, M. Dorigo, T. Stützle, The intelligent water drops algorithm: why it cannot be considered a novel algorithm, Swarm Intelligence 13 (2019) 173–192.

[180] P. Moscato, J. F. Fontanari, Stochastic versus deterministic update in simulated annealing, Physics Letters A 146 (4) (1990) 204–208.

[181] G. Dueck, T. Scheuer, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1) (1990) 161–175.

[182] K. Sörensen, Metaheuristics—the metaphor exposed, International Transactions in Operational Research 22 (1) (2015) 3–18. doi:10.1111/itor.12001.

[183] C. Aranha, F. Campelo, Evolutionary computation bestiary, https://github.com/fcampelo/EC-Bestiary, online; accessed 12 May 2017 (2013).

[184] Journal of Heuristics. Policies on Heuristic Search Research, http://www.springer.com/journal/10732, version visited last on June 10, 2015 (2015).

[185] M. Laguna, Editor's note on the MIC 2013 special issue of the journal of heuristics (volume 22, issue 4, august 2016), Journal of Heuristics 22 (5) (2016) 665–666.

[186] M. Dorigo, Swarm intelligence: A few things you need to know if you want to publish in this journal, Swarm Intelligence (Nov. 2016).
URL https://static.springer.com/sgw/documents/1593723/application/pdf/Additional_submission_instructions.pdf

[187] M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, 2nd Edition, Vol. 146 of International Series in Operations Research & Management Science, Springer, New York, NY, 2010.

[188] D. B. Fogel, A. J. Owens, M. J. Walsh, Artificial Intelligence Through Simulated Evolution, John Wiley & Sons, 1966.

[189] F. Glover, Heuristics for integer programming using surrogate constraints, Decision Sciences 8 (1977) 156–166.

[190] D. Weyland, A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology, International Journal of Applied Metaheuristic Computing 12 (2) (2010) 50–60.

[191] A. G. Nikolaev, S. H. Jacobson, Simulated annealing, in: Gendreau and Potvin [187], pp. 1–39.

[192] E. H. L. Aarts, J. H. M. Korst, W. Michiels, Simulated annealing, in: E. K. Burke, G. Kendall (Eds.), Search Methodologies, Springer, Boston, MA, 2005, pp. 187–210. doi:10.1007/0-387-28356-0.

[193] H. H. Hoos, Programming by optimization, Communications of the ACM 55 (2) (2012) 70–80. doi:10.1145/2076450.2076469.

[194] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, K. Leyton-Brown, SATenstein: Automatically building local search SAT solvers from components, in: C. Boutilier (Ed.), Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09), AAAI Press, Menlo Park, CA, 2009, pp. 517–524.

[195] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, K. Leyton-Brown, SATenstein: Automatically building local search SAT Solvers from Components, Artificial Intelligence 232 (2016) 20–42. doi:10.1016/j.artint.2015.11.002.

[196] T. Liao, T. Stützle, M. A. Montes de Oca, M. Dorigo, A unified ant colony optimization algorithm for continuous optimization, European Journal of Operational Research 234 (3) (2014) 597–609.

[197] A. Franzin, T. Stützle, Revisiting simulated annealing: a component-based analysis: Supplementaty material, http://iridia.ulb.ac.be/supp/IridiaSupp2018-001 (2018).

[198] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, Journal of Chemical Physics 21 (1953) 1087–1092.

[199] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[200] V. Černý, A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1) (1985) 41–51.

[201] M. Lundy, A. Mees, Convergence of an annealing algorithm, Mathematical Programming 34 (1) (1986) 111–124.

[202] B. Hajek, Cooling schedules for optimal annealing, Mathematics of Operations Research 13 (2) (1988) 311–329.

[203] D. Mitra, F. Romeo, A. Sangiovanni-Vincentelli, Convergence and finite-time behavior of simulated annealing, in: Decision and Control, 1985 24th IEEE Conference on, IEEE, 1985, pp. 761–767.

[204] M. Jerrum, Large cliques elude the Metropolis process, Random Structures & Algorithms 3 (4) (1992) 347–359.

[205] A. W. Johnson, S. H. Jacobson, On the convergence of generalized hill climbing algorithms, Discrete Applied Mathematics 119 (1) (2002) 37–57.

[206] J. E. Orosz, S. H. Jacobson, Analysis of static simulated annealing algorithms, Journal of Optimization Theory and Applications 115 (1) (2002) 165–182.

[207] H. Cohn, M. J. Fielding, Simulated annealing: Searching for an optimal temperature, SIAM Journal on Optimization 9 (3) (1999) 779–802.

[208] M. J. Fielding, Simulated annealing with an optimal fixed temperature, SIAM Journal on Optimization 11 (2) (2000) 289–307.

[209] G. Dueck, New optimization heuristics: the great deluge algorithm and the record-to-record travel, Journal of Computational Physics 104 (1) (1993) 86–92.

[210] E. K. Burke, Y. Bykov, The late acceptance hill-climbing heuristic, Tech. Rep. CSM-192, University of Stirling (2012).

[211] E. K. Burke, Y. Bykov, The late acceptance hill-climbing heuristic, European Journal of Operational Research 258 (1) (2017) 70–78.

[212] D. Henderson, S. H. Jacobson, A. W. Johnson, The theory and practice of simulated annealing, in: Handbook of Metaheuristics, Springer, 2003, pp. 287–319.

[213] C. Andrieu, N. de Freitas, A. Doucet, M. I. Jordan, An introduction to MCMC for machine learning, Machine Learning 50 (1-2) (2003) 5–43.

[214] I. H. Osman, C. N. Potts, Simulated annealing for permutation flow-shop scheduling, Omega 17 (6) (1989) 551–557.

[215] M. S. Hussin, T. Stützle, Tabu search vs. simulated annealing for solving large quadratic assignment instances, Computers & Operations Research 43 (2014) 286–291.

[216] R. E. Burkard, F. Rendl, A thermodynamically motivated simulation procedure for combinatorial optimization problems, European Journal of Operational Research 17 (2) (1984) 169–174. doi:10.1016/0377-2217(84)90231-5.

[217] D. T. Connolly, An improved annealing scheme for the QAP, European Journal of Operational Research 46 (1) (1990) 93–100.

[218] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation: Part I, graph partitioning, Operations Research 37 (6) (1989) 865–892.

[219] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation: Part II, graph coloring and number partitioning, Operations Research 39 (3) (1991) 378–406.

[220] K. Y. Tam, A simulated annealing algorithm for allocating space to manufacturing cells, International Journal of Production Research 30 (1) (1992) 63–87.

[221] A. Misevičius, A modified simulated annealing algorithm for the quadratic assignment problem, Informatica 14 (4) (2003) 497–514.

[222] S. Jajodia, I. Minis, G. Harhalakis, J.-M. Proth, CLASS: computerized layout solutions using simulated annealing, International Journal of Production Research 30 (1) (1992) 95–108.

[223] H. Ishibuchi, S. Misaki, H. Tanaka, Modified simulated annealing algorithms for the flow shop sequencing problem, European Journal of Operational Research 81 (2) (1995) 388–398.

[224] R.-M. Chen, F.-R. Hsieh, An exchange local search heuristic based scheme for permutation flow shop problems, Applied Mathematics & Information Sciences 8 (1) (2014) 209–215.

[225] I. O. Bohachevsky, M. E. Johnson, M. L. Stein, Generalized simulated annealing for function optimization, Technometrics 28 (3) (1986) 209–217.

[226] F. A. Ogbu, D. K. Smith, The application of the simulated annealing algorithm to the solution of the n/m/C max flowshop problem, Computers & Operations Research 17 (3) (1990) 243–253.

[227] B. Hajek, G. Sasaki, Simulated annealing–to cool or not, System & Control Letters 12 (5) (1989) 443–447.

[228] J. S. Appleby, D. V. Blake, E. A. Newman, Techniques for producing school timetables on a computer and their application to other scheduling problems, The Computer Journal 3 (4) (1961) 237–245. doi:10.1093/comjnl/3.4.237.

[229] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (6) (1984) 721–741.

[230] P. N. Strenski, S. Kirkpatrick, Analysis of finite length annealing schedules, Algorithmica 6 (1-6) (1991) 346–366.

[231] H. Szu, R. Hartley, Fast simulated annealing, Physics Letters A 122 (3) (1987) 157–162.

[232] K. Andersen, R. V. V. Vidal, V. B. Iversen, Design of a teleprocessing communication network using simulated annealing, in: R. V. V. Vidal (Ed.), Applied Simulated Annealing, Springer, 1993, pp. 201–215.

[233] S. Kirkpatrick, Optimization by simulated annealing: Quantitative studies, Journal of Statistical Physics 34 (5-6) (1984) 975–986.

[234] M. Jerrum, A. Sinclair, The Markov chain Monte Carlo method: an approach to approximate counting and integration, in: D. S. Hochbaum (Ed.), Approximation Algorithms For NP-hard Problems, PWS Publishing Co., 1996, pp. 482–520.

[235] T. C. Hu, A. B. Kahng, C.-W. A. Tsao, Old bachelor acceptance: A new class of non-monotone threshold accepting methods, ORSA Journal on Computing 7 (4) (1995) 417–425.

[236] P. Hansen, B. Jaumard, Algorithms for the maximum satisfiability problem, Computing 44 (1990) 279–303.

[237] D. Abramson, Constructing school timetables using simulated annealing: Sequential and parallel algorithms, Management Science 37 (1) (1991) 98–113.

[238] J. Rose, W. Klebsch, J. Wolf, Temperature measurement and equilibrium dynamics of simulated annealing placements, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 9 (3) (1990) 253–259.

[239] A. Souilah, Simulated annealing for manufacturing systems layout design, European Journal of Operational Research 82 (3) (1995) 592–614.

[240] D. Abramson, M. K. Amoorthy, H. Dang, Simulated annealing cooling schedules for the school timetabling problem, Asia-Pacific Journal of Operational Research 16 (1) (1999) 1–22.

[241] T. Stützle, Some thoughts on engineering stochastic local search algorithms, in: A. Viana, et al. (Eds.), Proceedings of the EU/MEeting 2009: Debating the future: new areas of application and innovative approaches, 2009, pp. 47–52.

[242] S. Zilberstein, Using anytime algorithms in intelligent systems, AI Magazine 17 (3) (1996) 73–83. arXiv:http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1232, doi:10.1609/aimag.v17i3.1232.

[243] M. López-Ibáñez, T. Stützle, Automatically improving the anytime behaviour of optimisation algorithms, European Journal of Operational Research 235 (3) (2014) 569–582. doi:10.1016/j.ejor.2013.10.043.

[244] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.

[245] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the F-race algorithm: Sampling design and iterative refinement, in: T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, Vol. 4771 of Lecture Notes in Computer Science, Springer, Heidelberg, 2007, pp. 108–122. doi:10.1007/978-3-540-75514-2_9.

[246] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: An overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, 2010, pp. 311–336. doi:10.1007/978-3-642-02538-9_13.

[247] M. N. Wright, A. Ziegler, ranger: A fast implementation of random forests for high dimensional data in c++ and r, Journal of Statistical Software 77 (1) (2017) 1–17. doi:10.18637/jss.v077.i01.

[248] É. D. Taillard, Comparison of iterative searches for the quadratic assignment problem, Location Science 3 (2) (1995) 87–105.

[249] M. Nawaz, E. Enscore, Jr, I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11 (1) (1983) 91–95.

[250] É. D. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (2) (1993) 278–285.

[251] E. Vallada, R. Ruiz, J. M. Framiñán, New hard benchmark for flowshop scheduling problems minimising makespan, European Journal of Operational Research 240 (3) (2015) 666–677. doi:10.1016/j.ejor.2014.07.033.

[252] J. Paulli, A computational comparison of simulated annealing and tabu search applied to the quadratic assignment problem, in: R. V. V. Vidal (Ed.), Applied Simulated Annealing, Springer, 1993, pp. 85–102.

[253] E. Çela, The Quadratic Assignment Problem: Theory and Algorithms, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[254] R. Battiti, G. Tecchiolli, Simulated annealing and tabu search in the long run: A comparison on QAP tasks, Computer and Mathematics with Applications 28 (6) (1994) 1–8. doi:10.1016/0898-1221(94)00147-2.

[255] G. Paul, Comparative performance of tabu search and simulated annealing heuristics for the quadratic assignment problem, Operations Research Letters 38 (6) (2010) 577–581.

[256] R. Genuer, J.-M. Poggi, C. Tuleau-Malot, Variable selection using random forests, Pattern Recognition Letters 31 (14) (2010) 2225–2236.

[257] B. L. Fox, Integrating and accelerating tabu search, simulated annealing, and genetic algorithms, Annals of Operations Research 41 (2) (1993) 47–67.

[258] B. L. Fox, Simulated annealing: folklore, facts, and directions, in: Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Springer, 1995, pp. 17–48.

[259] W. G. Jackson, E. Özcan, R. I. John, Move acceptance in local search metaheuristics for cross-domain search, Expert Systems with Applications 109 (2018) 131–151.

[260] K. Smith-Miles, Towards insightful algorithm selection for optimisation using meta-learning concepts, in: D. Liu, et al. (Eds.), Proceedings of the International Joint Conference on Neural Networks (IJCNN 2008), Hong Kong, China, June 1-6, 2008, IEEE Press, 2008, pp. 4118–4124.

[261] E. Pitzer, A. Beham, M. Affenzeller, Automatic algorithm selection for the quadratic assignment problem using fitness landscape analysis, in: M. Middendorf, C. Blum (Eds.), Proceedings of EvoCOP 2013 – 13th European Conference on Evolutionary Computation in Combinatorial Optimization, Vol. 7832 of Lecture Notes in Computer Science, Springer, Heidelberg, 2013, pp. 109–120.

[262] A. Beham, M. Affenzeller, S. Wagner, Instance-based algorithm selection on quadratic assignment problem landscapes, in: P. A. N. Bosman (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2017, ACM Press, New York, NY, 2017, pp. 1471–1478.

[263] A. L. Dantas, A. T. R. Pozo, A meta-learning algorithm selection approach for the quadratic assignment problem, in: Proceedings of the 2018 Congress on Evolutionary Computation (CEC 2018), IEEE Press, Piscataway, NJ, 2018, pp. 1–8.

[264] R. E. Burkard, S. E. Karisch, F. Rendl, QAPLIB–a quadratic assignment problem library, Journal of Global Optimization 10 (4) (1997) 391–403.

[265] M. López-Ibáñez, M.-E. Kessaci, T. Stützle, Automatic design of hybrid metaheuristics from algorithmic components, Tech. Rep. TR/IRIDIA/2017-012, IRIDIA, Université Libre de Bruxelles, Belgium (Dec. 2017).
URL http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2017-012.pdf

[266] M.-E. Marmion, F. Mascia, M. López-Ibáñez, T. Stützle, Automatic design of hybrid stochastic local search algorithms, in: M. J. Blesa, C. Blum, P. Festa, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics, Vol. 7919 of Lecture Notes in Computer Science, Springer, Heidelberg, 2013, pp. 144–158. doi:10.1007/978-3-642-38516-2_12.

[267] H. Mühlenbein, J. Zimmermann, Size of neighborhood more important than temperature for stochastic local search, in: Proceedings of the 2000 Congress on Evolutionary Computation (CEC'00), IEEE Press, Piscataway, NJ, 2000, pp. 1017–1024.

[268] I. Wegener, Simulated annealing beats metropolis in combinatorial optimization, in: L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005, Vol. 3580 of Lecture Notes in Computer Science, Springer, Heidelberg, 2005, pp. 589–601.

[269] K. Meer, Simulated annealing versus Metropolis for a TSP instance, Information Processing Letters 104 (6) (2007) 216–219.

[270] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in: N. Krasnogor, P. L. Lanzi (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, ACM Press, New York, NY, 2011, pp. 829–836.

[271] M. Jerrum, G. Sorkin, The Metropolis algorithm for graph bisection, Discrete Applied Mathematics 82 (1) (1998) 155–175.

[272] H. Haario, E. Saksman, J. Tamminen, An adaptive Metropolis algorithm, Bernoulli 7 (2) (2001) 223–242.

[273] M. Sambridge, Geophysical inversion with a neighbourhood algorithm–i. searching a parameter space, Geophysical Journal International 138 (2) (1999) 479–494.

[274] D. H. Rothman, Nonlinear inversion, statistical mechanics, and residual statics estimation, Geophysics 50 (12) (1985) 2784–2796.

[275] D. H. Rothman, Automatic estimation of large residual statics corrections, Geophysics 51 (2) (1986) 332–346.

[276] A. Basu, L. N. Frazer, Rapid determination of the critical temperature in simulated annealing inversion, Science 249 (4975) (1990) 1409–1412.

[277] A. Franzin, T. Stützle, A landscape-based analysis of fixed temperature and simulated annealing: Supplementaty material, http://iridia.ulb.ac.be/supp/IridiaSupp2021-002 (2021).

[278] J.-P. Watson, L. Barbulescu, D. Whitley, A. E. Howe, Contrasting structured and random permutation flow-shop scheduling problems: Search space topology and algorithm performance, INFORMS Journal on Computing 14 (2) (2002) 98–123.

[279] A. Dantas, A. Pozo, On the use of fitness landscape features in meta-learning based algorithm selection for the quadratic assignment problem, Theoretical Computer Science 805 (2020) 62–75. doi:10.1016/j.tcs.2019.10.033.

[280] M. Kuhn, Building predictive models in R using the caret package, Journal of Statistical Software 28 (5) (2008) 1–26.

[281] B. Doerr, C. Gießen, C. Witt, J. Yang, The (1+λ) evolutionary algorithm with self-adjusting mutation rate, Algorithmica 81 (2) (2019) 593–631.

[282] N. Dang, C. Doerr, Hyper-parameter tuning for the (1 + (λ, λ)) GA, in: M. López-Ibáñez, A. Auger, T. Stützle (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, ACM Press, New York, NY, 2019, pp. 889–897. doi:10.1145/3321707.3321725.

[283] B. Doerr, C. Doerr, J. Yang, Optimal parameter choices via precise black-box analysis, Theoretical Computer Science 801 (2020) 1–34. doi:10.1016/j.tcs.2019.06.014.

[284] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, W. R. Stewart, Designing and reporting on computational experiments with heuristic methods, Journal of Heuristics 1 (1) (1995) 9–32. doi:10.1007/BF02430363.

[285] F. Hutter, H. H. Hoos, K. Leyton-Brown, Identifying key algorithm parameters and instance features using forward selection, in: P. M. Pardalos, G. Nicosia (Eds.), Learning and Intelligent Optimization, 7th International Conference, LION 7, Vol. 7997 of Lecture Notes in Computer Science, Springer, Heidelberg, 2013, pp. 364–381. doi:10.1007/978-3-642-44973-4_40.

[286] F. Hutter, H. H. Hoos, K. Leyton-Brown, An efficient approach for assessing hyperparameter importance, in: E. P. Xing, T. Jebara (Eds.), Proceedings of the 31st International Conference on Machine Learning, ICML 2014, Vol. 32, 2014, pp. 754–762.
URL http://jmlr.org/proceedings/papers/v32/hutter14.html

[287] J. Corstjens, B. Depaire, A. Caris, K. Sörensen, A multilevel evaluation method for heuristics with an application to the VRPTW, International Transactions in Operational Research 27 (1) (2020) 168–196. doi:10.1111/itor.12631.

[288] C. R. Reeves, A. V. Eremeev, Statistical analysis of local search landscapes, Journal of the Operational Research Society 55 (7) (2004) 687–693. arXiv:http://www.jstor.org/stable/4102015.

[289] J. Pearl, Causality: Models, Reasoning and Inference, 2nd Edition, Cambridge University Press, 2009.

[290] J. Pearl, D. Mackenzie, The book of why: the new science of cause and effect, Basic books, 2018.

[291] M. Birattari, M. Chiarandini, M. Saerens, T. Stützle, Learning graphical models for algorithm configuration, in: T. Berthold, A. M. Gleixner, S. Heinz, T. Koch (Eds.), Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science, Springer, Heidelberg, 2011.

[292] A. E. Eiben, G. Rudolph, Theory of evolutionary algorithms: A bird's eye view, Theoretical Computer Science 229 (1-2) (1999) 3–9.

[293] B. Doerr, D. Johannsen, C. Winzen, Multiplicative drift analysis, Algorithmica 64 (4) (2012) 673–697.

[294] A. Auger, B. Doerr (Eds.), Theory of Randomized Search Heuristics: Foundations and Recent Developments, Vol. 1 of Series on Theoretical Computer Science, World Scientific Publishing Co., Singapore, 2011.

[295] J. N. Hooker, Needed: An empirical science of algorithms, Operations Research 42 (2) (1994) 201–212.

[296] J. N. Hooker, Testing heuristics: We have it all wrong, Journal of Heuristics 1 (1) (1996) 33–42. doi:10.1007/BF02430364.

[297] D. S. Johnson, A theoretician's guide to the experimental analysis of algorithms, in: M. H. Goldwasser, D. S. Johnson, C. C. McGeoch (Eds.), Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, American Mathematical Society, Providence, RI, 2002, pp. 215–250.

[298] F. Campelo, E. F. Wanner, Sample size calculations for the experimental comparison of multiple algorithms on multiple problem instances, Journal of Heuristics (2020). doi:10.1007/s10732-020-09454-w.

[299] F. Romeo, A. Sangiovanni-Vincentelli, A theoretical framework for simulated annealing, Algorithmica 6 (1-6) (1991) 302–345.

[300] P. J. M. van Laarhoven, E. H. L. Aarts, Simulated Annealing: Theory and Applications, Vol. 37, Springer, 1987.

[301] W. J. Gutjahr, A Graph-based Ant System and its convergence, Future Generation Computer Systems 16 (8) (2000) 873–888.

[302] W. J. Gutjahr, ACO algorithms with guaranteed convergence to the optimal solution, Information Processing Letters 82 (3) (2002) 145–153.

[303] G. Rudolph, Convergence of non-elitist strategies, in: Z. Michalewicz (Ed.), Proceedings of the First IEEE International Conference on Evolutionary Computation (ICEC'94), IEEE Press, Piscataway, NJ, 1994, pp. 63–66.

[304] G. Rudolph, Convergence analysis of canonical genetic algorithms, tnn 5 (1) (1994) 96–101.

[305] H.-G. Beyer, H.-P. Schwefel, I. Wegener, How to analyse evolutionary algorithms, Theoretical Computer Science 287 (1) (2002) 101–130.

[306] P. S. Oliveto, J. He, X. Yao, Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results, International Journal of Automation and Computing 4 (3) (2007) 281–293.

[307] P. S. Oliveto, C. Witt, Improved time complexity analysis of the simple genetic algorithm, Theoretical Computer Science 605 (2015) 21–41. doi:10.1016/j.tcs.2015.01.002.

[308] T. Stützle, M. Dorigo, A short convergence proof for a class of ACO algorithms, IEEE Transactions on Evolutionary Computation 6 (4) (2002) 358–365.

[309] B. Doerr, F. Neumann, D. Sudholt, C. Witt, Runtime analysis of the 1-ANT ant colony optimizer, Theoretical Computer Science 412 (1) (2011) 1629–1644.

[310] M. A. Arostegui Jr, S. N. Kadipasaoglu, B. M. Khumawala, An empirical comparison of tabu search, simulated annealing, and genetic algorithms for facilities location problems, International Journal of Production Economics 103 (2) (2006) 742–754.

[311] J.-K. Hao, J. Pannier, Simulated annealing and tabu search for constraint solving, in: M. C. Golumbic, et al. (Eds.), Fifth International Symposium on Artificial Intelligence and Mathematics, AIM 1998, Fort Lauderdale, Florida, USA, January 4-6, 1998, 1998, pp. 1–15.

[312] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2) (2005) 479–494.

[313] J.-P. Watson, A. E. Howe, D. Whitley, Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem, Computers & Operations Research 33 (9) (2006) 2623–2644.

[314] S. M. Oliveira, M. S. Hussin, A. Roli, M. Dorigo, T. Stützle, Analysis of the population-based ant colony optimization algorithm for the TSP and the QAP, in: Proceedings of the 2017 Congress on Evolutionary Computation (CEC 2017), IEEE Press, Piscataway, NJ, 2017, pp. 1734–1741.

[315] L. Di Gaspero, M. Chiarandini, A. Schaerf, A study on the short-term prohibition mechanisms in tabu search, in: G. Brewka, S. Coradeschi, A. Perini, P. Traverso (Eds.), Proceedings of the 17th European Conference on Artificial Intelligence, ECAI 2006, Riva del Garda, Italy, August29 - September 1, 2006, IOS Press, 2006, pp. 83–87.

[316] A. Santini, S. Ropke, L. M. Hvattum, A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic, Journal of Heuristics 24 (2018) 783–815. doi:10.1007/s10732-018-9377-x.

[317] A. Misevičius, D. Kuznecovaitė, Investigating some strategies for construction of initial populations in genetic algorithms, Computational Science and Techniques 5 (1) (2018) 560–573.

[318] A. Misevičius, D. Kuznecovaitė, J. Platužienė, Some further experiments with crossover operators for genetic algorithms, Informatica 29 (3) (2018) 499–516.

[319] M. Foster, M. Hughes, G. O'Brien, P. S. Oliveto, J. Pyle, D. Sudholt, J. Williams, Do sophisticated evolutionary algorithms perform better than simple ones?, in: C. A. Coello Coello (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2020, ACM Press, New York, NY, 2020, pp. 184–192. doi:10.1145/3377930.

[320] C. L. Camacho-Villalón, M. Dorigo, T. Stützle, Why the intelligent water drops cannot be considered as a novel algorithm, in: M. Dorigo, M. Birattari, A. L. Christensen, A. Reina, V. Trianni (Eds.), Swarm Intelligence, 11th International Conference, ANTS 2018, Vol. 11172 of Lecture Notes in Computer Science, Springer, Heidelberg, 2018, pp. 302–314.

[321] K. Sörensen, F. Arnold, D. Palhazi Cuervo, A critical analysis of the "improved clarke and wright savings algorithm", International Transactions in Operational Research 26 (1) (2017) 54–63. doi:10.1111/itor.12443.

[322] C. L. Camacho-Villalón, T. Stützle, M. Dorigo, Cuckoo search $\equiv (\mu + \lambda)$-evolution strategy – a rigorous analysis of an algorithm that has been misleading the research community for more than 10 years and nobody seems to have noticed, Tech. Rep. TR/IRIDIA/2021-006, IRIDIA, Université Libre de Bruxelles, Belgium (2021).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2021-006.pdf

[323] K. Smith-Miles, J. I. van Hemert, X. Y. Lim, Understanding TSP difficulty by learning from evolved instances, in: C. Blum, R. Battiti (Eds.), Learning and Intelligent Optimization, 4th International Conference, LION 4, Vol. 6073 of Lecture Notes in Computer Science, Springer, Heidelberg, 2010, pp. 266–280. doi:10.1007/978-3-642-13800-3.

[324] P. C. Cheeseman, B. Kanefsky, W. M. Taylor, Where the really hard problems are, in: J. Mylopoulos, R. Reiter (Eds.), Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), Morgan Kaufmann Publishers, 1995, pp. 331–340.

[325] K. Smith-Miles, S. Bowly, Generating new test instances by evolving in instance space, Computers & Operations Research 63 (2015) 102–113.

[326] A. A. Albrecht, P. C. R. Lane, K. Steinhöfel, Analysis of local search landscapes for k-SAT instances, Mathematics in Computer Science 3 (4) (2010) 465–488. doi:10.1007/s11786-010-0040-7.

[327] P. Kerschke, H. Wang, M. Preuss, C. Grimme, A. H. Deutz, H. Trautmann, M. T. M. Emmerich, Towards analyzing multimodality of continuous multiobjective landscapes, in: J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, B. Paechter (Eds.), Parallel Problem Solving from Nature – PPSN XIV, Vol. 9921 of Lecture Notes in Computer Science, Springer, Heidelberg, 2016, pp. 962–972. doi:10.1007/978-3-319-45823-6_90.

[328] F. Daolio, S. Verel, G. Ochoa, M. Tomassini, Local optima networks and the performance of iterated local search, in: T. Soule, J. H. Moore (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2012, ACM Press, New York, NY, 2012, pp. 369–376.

[329] C. Blum, G. Ochoa, A comparative analysis of two matheuristics by means of merged local optima networks, European Journal of Operational Research 290 (1) (2021) 36–56.

[330] A. Liefooghe, B. Derbel, S. Verel, H. E. Aguirre, K. Tanaka, Towards landscape-aware automatic algorithm configuration: Preliminary experiments on neutral and rugged landscapes, in: H. Trautmann, G. Rudolph, K. Klamroth, O. Schütze, M. M. Wiecek, Y. Jin, C. Grimme (Eds.), Evolutionary Multi-criterion Optimization, EMO 2017, Lecture Notes in Computer Science, Springer International Publishing, Cham, Switzerland, 2017, pp. 215–232.

[331] A. Kaznatcheev, D. A. Cohen, P. Jeavons, Representing fitness landscapes by valued constraints to understand the complexity of local search, Journal of Artificial Intelligence Research 69 (2020) 1077–1102. doi:10.1613/jair.1.12156.

[332] J.-P. Watson, J. C. Beck, A. E. Howe, D. Whitley, Problem difficulty for tabu search in job-shop scheduling, Artificial Intelligence 143 (2) (2003) 189–217.

[333] L. M. Pavelski, M. R. Delgado, M.-E. Kessaci, Meta-learning on flowshop using fitness landscape analysis, in: M. López-Ibáñez, A. Auger, T. Stützle (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, ACM Press, New York, NY, 2019, pp. 925–933. doi:10.1145/3321707.

[334] F. Arnold, K. Sörensen, What makes a VRP solution good? the generation of problem-specific knowledge for heuristics, Computers & Operations Research 106 (2019) 280–288. doi:10.1016/j.cor.2018.02.007.

[335] F. Arnold, K. Sörensen, Knowledge-guided local search for the vehicle routing problem, Computers & Operations Research 105 (2019) 32–46. doi:10.1016/j.cor.2019.01.002.

[336] L. Mousin, M.-E. Kessaci, C. Dhaenens, Exploiting promising subsequences of jobs to solve the no-wait flowshop scheduling problem, Arxiv preprint arXiv:1903.09035 (2019).
URL http://arxiv.org/abs/1903.09035

[337] S. Droste, T. Jansen, I. Wegener, A new framework for the valuation of algorithms for black-box-optimization, in: K. A. De Jong, R. Poli, J. E. Rowe (Eds.), Proceedings of the Seventh Workshop on Foundations of Genetic Algorithms (FOGA), Morgan Kaufmann Publishers, 2002, pp. 253–270.

[338] S. Droste, T. Jansen, I. Wegener, Upper and lower bounds for randomized search heuristics in black-box optimization, Theory of Computing Systems 39 (4) (2006) 525–544.

[339] B. Doerr, T. Kötzing, J. Lengler, C. Winzen, Black-box complexities of combinatorial problems, Theoretical Computer Science 471 (2013) 84–106.

[340] P. K. Lehre, C. Witt, Black-box search by unbiased variation, Algorithmica 64 (4) (2012) 623–642.

[341] B. Doerr, C. Doerr, F. Ebel, From black-box complexity to designing new genetic algorithms, Theoretical Computer Science 567 (2015) 87–104. doi:10.1016/j.tcs.2014.11.028.

[342] S. Koziel, D. E. Ciaurri, L. Leifsson, Surrogate-based methods, in: S. Koziel, X.-S. Yang (Eds.), Computational Optimization, Methods and Algorithms, Vol. 356 of Studies in Computational Intelligence, Springer, Berlin/Heidelberg, 2011, pp. 33–59.

[343] A. Moraglio, A. Kattan, Geometric generalisation of surrogate model based optimization to combinatorial spaces, in: P. Merz, J.-K. Hao (Eds.), Proceedings of EvoCOP 2011 – 11th European Conference on Evolutionary Computation in Combinatorial Optimization, Vol. 6622 of Lecture Notes in Computer Science, Springer, Heidelberg, 2011, pp. 142–154.

[344] A. Moraglio, Y. Kim, Y. Yoon, Geometric surrogate-based optimisation for permutation-based problems, in: N. Krasnogor, P. L. Lanzi (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2011, ACM Press, New York, NY, 2011, pp. 133–134.

[345] B. S. Saini, M. López-Ibáñez, K. Miettinen, Automatic surrogate modelling technique selection based on features of optimization problems, in: M. López-Ibáñez, A. Auger, T. Stützle (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2019, ACM Press, New York, NY, 2019, pp. 1765–1772. doi:10.1145/3319619.3326890.

[346] T. Chugh, K. Sindhya, J. Hakanen, K. Miettinen, A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms, Soft Computing 23 (9) (2019) 3137–3166. doi:10.1007/s00500-017-2965-0.

[347] N. Dang Thi Thanh, L. Pérez Cáceres, P. De Causmaecker, T. Stützle, Configuring irace using surrogate configuration benchmarks, in: P. A. N. Bosman (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, ACM Press, New York, NY, 2017, pp. 243–250. doi:10.1145/3071178.3071238.

[348] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, M. Lang, mlrMBO: A modular framework for model-based optimization of expensive black-box functions, Arxiv preprint arXiv:1703.03373 [stat.ML] (2017).
URL http://arxiv.org/abs/1703.03373

[349] K. Eggensperger, F. Hutter, H. H. Hoos, K. Leyton-Brown, Efficient benchmarking of hyperparameter optimizers via surrogates, in: B. Bonet, S. Koenig (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2015, pp. 1114–1120.

[350] G. Hooker, Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables, Journal of Computational and Graphical Statistics 16 (3) (2012) 709–732. doi:10.1198/106186007X237892.

[351] A. Biedenkapp, M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, H. H. Hoos, Efficient parameter importance analysis via ablation with surrogates, in: S. P. Singh, S. Markovitch (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2017.
URL https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14750

[352] C. Fawcett, H. H. Hoos, Analysing differences between algorithm configurations through ablation, Journal of Heuristics 22 (4) (2016) 431–458.

[353] L. Xu, A. R. KhudaBukhsh, H. H. Hoos, K. Leyton-Brown, Quantifying the similarity of algorithm configurations, in: P. Festa, M. Sellmann, J. Vanschoren (Eds.), Learning and Intelligent Optimization, 10th International Conference, LION 10, Vol. 10079 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2016, pp. 203–217.

[354] J. Corstjens, N. Dang, B. Depaire, A. Caris, P. De Causmaecker, A combined approach for analysing heuristic algorithms, Journal of Heuristics 25 (4) (2019) 591–628. doi:10.1007/s10732-018-9388-7.

[355] H. H. Hoos, T. Stützle, On the empirical scaling of run-time for finding optimal solutions to the traveling salesman problem, European Journal of Operational Research 238 (1) (2014) 87–94.

[356] J. Dubois-Lacoste, H. H. Hoos, T. Stützle, On the empirical scaling behaviour of state-of-the-art local search algorithms for the euclidean TSP, in: S. Silva, A. I. Esparcia-Alcázar (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, ACM Press, New York, NY, 2015, pp. 377–384. doi:10.1145/2739480.2754747.

[357] J. Dréo, C. Doerr, Y. Semet, Coupling the design of benchmark with algorithm in landscape-aware solver design, in: M. López-Ibáñez, A. Auger, T. Stützle (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2019, ACM Press, New York, NY, 2019, pp. 1419–1420. doi:10.1145/3319619.

[358] M. Alissa, K. Sim, E. Hart, Algorithm selection using deep learning without feature extraction, in: M. López-Ibáñez, A. Auger, T. Stützle (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, ACM Press, New York, NY, 2019, pp. 198–206. doi:10.1145/3321707.

[359] M. Seiler, J. Pohl, J. Bossek, P. Kerschke, H. Trautmann, Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem, in: T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. T. M. Emmerich, H. Trautmann (Eds.), Parallel Problem Solving from Nature - PPSN XVI, Vol. 12269 of Lecture Notes in Computer Science, Springer, Cham, Switzerland, 2020, pp. 48–64.

[360] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney, ISAC: Instance-specific algorithm configuration, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), Proceedings of the 19th European Conference on Artificial Intelligence, IOS Press, 2010, pp. 751–756.

[361] N. Belkhir, J. Dréo, P. Savéant, M. Schoenauer, Feature based algorithm configuration: A case study with differential evolution, in: J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, B. Paechter (Eds.), Parallel Problem Solving from Nature – PPSN XIV, Vol. 9921 of Lecture Notes in Computer Science, Springer, Heidelberg, 2016, pp. 156–166. doi:10.1007/978-3-319-45823-6.

[362] N. Belkhir, J. Dréo, P. Savéant, M. Schoenauer, Per instance algorithm configuration of CMA-ES with limited budget, in: P. A. N. Bosman (Ed.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, ACM Press, New York, NY, 2017, pp. 681–688.

[363] O. Maron, A. W. Moore, The racing algorithm: Model selection for lazy learners, Artificial Intelligence Research 11 (1–5) (1997) 193–225.

[364] S. Thrun, L. Pratt, Learning to learn, springer, 1998.

[365] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering 22 (10) (2009) 1345–1359.

[366] J. Pearl, E. Bareinboim, Transportability of causal and statistical relations: A formal approach, in: W. Burgard, D. Roth (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2011, pp. 247–254.

[367] E. Bareinboim, J. Pearl, Transportability of causal effects: Completeness results, in: J. Hoffmann, B. Selman (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2012, pp. 698,704.

[368] E. Van Wolputte, E. Korneva, H. Blockeel, MERCS: multi-directional ensembles of regression and classification trees, in: S. A. McIlraith, K. Q. Weinberger (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2018, pp. 4276–4283.

[369] D. S. Johnson, L. A. McGeoch, Experimental analysis of heuristics for the STSP, in: G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and its Variations, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002, pp. 369–443.

[370] T. Stützle, Iterated local search for the quadratic assignment problem, European Journal of Operational Research 174 (3) (2006) 1519–1539.

[371] A. E. Eiben, S. K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 19–31. doi:10.1016/j.swevo.2011.02.001.

[372] É. D. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Computing 17 (4-5) (1991) 443–455.

[373] J. Snoek, K. Swersky, R. Zemel, R. P. Adams, Input warping for Bayesian optimization of non-stationary functions, in: E. P. Xing, T. Jebara (Eds.), Proceedings of the 31st International Conference on Machine Learning, ICML 2014, Vol. 32, 2014, pp. 1674–1682.
URL http://jmlr.org/proceedings/papers/v32/

[374] N. Girerd, M. Rabilloud, P. Pibarot, P. Mathieu, P. Roy, Quantification of treatment effect modification on both an additive and multiplicative scale, PLoS One 11 (4) (2016) 1–14. doi:10.1371/journal.pone.0153010.

[375] M. J. Knol, T. J. VanderWeele, R. H. H. Groenwold, O. H. Klungel, M. M. Rovers, D. E. Grobbee, Estimating measures of interaction on an additive scale for preventive exposures, European Journal of Epidemiology 26 (6) (2011) 433–438.