UNIVERSITÉ DE LILLE

ÉCOLE DOCTORALE MADIS-631: MATHÉMATIQUES,

SCIENCES DU NUMÉRIQUE ET DE LEURS INTERACTIONS

Automatic Algorithm Multi-Configuration for Combinatorial Optimization

Multi-Configuration Automatique d'Algorithmes pour l'Optimisation Combinatoire

Weerapan SAE-DAN

Thèse préparée et soutenue publiquement le **20 Juin 2022,**

en vue de l'obtention du grade de **Docteur en Informatique et Applications.**

Membres du jury:

M. Nicolas JOZEFOWIEZ	Professeur, Université de Lorraine	*Rapporteur*
M. Frédéric SAUBION	Professeur, Université d'Angers	*Rapporteur*
M. Gilles GONCALVES	Professeur, Université d'Artois	*Examinateur/Président*
Mme. Laetitia JOURDAN	Professeur, Université de Lille	*Directrice de Thèse*
Mme. Marie-Eléonore KESSACI	MCF (HDR), Université de Lille	*Co-Directrice de Thèse*
M. Nadarajen VEERAPEN	MCF, Université de Lille	*Co-Encadrant de Thèse*

Centre de Recherche en Informatique, Signal et Automatique de Lille

Université de Lille - Bâtiment ESPRIT - Avenue Henri Poincaré

59655 Villeneuve d'Ascq Cedex FRANCE

Acknowledgments

My three years as a doctoral student were an opportunity to meet many professional and personal professional and personal encounters. All these people have allowed, in one way or another in one way or another, and for these reasons, I will try to thank them all, trying not to forget anyone. I will try to thank all of them, trying not to forget anyone.

Firstly, I'd like to thank my supervisor for everything she has accomplished for me, Prof. Dr. Laetitia JOURDAN for providing me with such a wonderful experience and guiding me to the hall of automatic local search algorithms and parameter control. Her advice, passion, deep knowledge, continuous support, encouragement, and critical feedback have all made significant contributions to my thesis and professional development throughout the years. Also, thank you so much for all of your assistance and guidance throughout my thesis for both teaching and research. I am delighted for the opportunity to associate with her.

I also owe a heartfelt thanks to my co-supervisor, Assoc. Prof. Dr. Marie-Eléonore KASSECI, throughout my thesis, she provided me with constant assistance and encouragement. Thank you very much for your time and effort; your comments are always relevant and to the point. Without her assistance, having high-quality articles and achieving the process of producing big ideas is quite difficult. Her enthusiasm for research and methodological rigor has had a significant impact on me.

I'd also like to express my gratitude to, Assoc. Prof. Dr. Nadarajen VEERAPEN, co-supervisor of this thesis, for advising, supporting and encouraging me throughout these three years. Three years of doctoral studies. Even in moments of doubt, you managed to push me to bounce back and go to the end of this adventure, and

it is thanks to you that this manuscript was completed, so thank you again.

I'd also like to express my gratitude to Prof. Nicolas JOZEFOWIEZ and Prof. Frédéric SAUBION for agreeing to review my thesis and provide helpful feedback. Prof. Gilles GONCALVES, a member of the jury, deserves special thanks for agreeing to serve on the thesis committee and evaluate my work.

I also thank the members of our ORKAD team for all the advice and good times spent together. I would like to thank Lucien MOUSIN, Camille PAGEAU, Rabin Kumar SAHU, Sara TARI, Nicolas SZCZEPANSKI, Adan JOSE-GARCIA, Laurent PARMENTIER, Mounir HAFSA, Thomas FEUTRIER, Meyssa ZOUAMBI, Agathe MÉTAIREAU, and Clément LEGRAND, and outside of ORKAD team who are Soheila GAMBARI and Lucas M. PAVELSKI.

I owe a debt of gratitude to many kindhearted people outside of the school who have helped me in some manner during this incredible trip. First and foremost, I want to express my thanks to Ramkhamhaeng University (RU), a member of Thailand's Office of the Civil Service Commission (OCSC), for providing me with a complete scholarship to pursue a doctoral degree in France. Second, I'd want to express my gratitude to Dr. Boonchauy SRITHAMMASAK (Former Director of Computer Engineering, RU, Thailand) for providing me with such a wonderful opportunity to develop in-depth knowledge, skills, and research talents in computer science. Third, I'd like to express my gratitude to Miss Panida ROJRATTANACHAI (Minister Counsellor of Education, France), Mrs. Nareenush KAOPAIBOOL (Education senior officer, France), and Mr. Somchai INJORHOR (Education senior officer, OCSC, Thailand) for their unwavering support over the years, which included assisting with all services related to local law and regulation, providing some guidance for health care, resident permit, and accom Thank you for making my time in France more enjoyable.

I will end this page with thanks from my family. First of all, Wirawan SAE-DAN who has been with me every day from the beginning to the end of this thesis, who has supported and comforted me in difficult moments, and I can't thank her enough for that, so I say it here, thank you! here, thank you! Then, I thank my family: my mother and my father that my angle I missed you so much, my brother, my sisters who have supported me since I was born and encouraged me in my academic and professional choices.

Abstract

Metaheuristics are resolution algorithms with a large number of parameters that allow them to adapt to any type of optimization problem. In order to obtain the best performance, the parameters must be chosen scrupulously, which generates a very tedious parameterization work. It is in this context that parameter tuning approaches have been developed in the literature, where a learning phase allows to explore different sets of parameters to select the best one, and parameter control approaches where the values change adaptively during the execution. In this thesis, we propose to use simultaneously these two parameterization approaches by proposing an approach called automatic multi-configuration of algorithms. In particular, we explore several strategies based on sequential or probabilistic models and compare them to classical approaches for automatic algorithm configuration and adaptive algorithms. Experiments have been conducted on the permutation flowshop scheduling problem and the traveling salesman problem and show the relevance of the proposed approach.

Keywords— Automatic Algorithm Configuration - Adaptive Control - Local Search

Résumé

Les métaheuristiques sont des algorithmes de résolution présentant un grand nombre de paramètres qui leur permettent de s'adapter à tout type de problème d'optimisation. Afin d'obtenir les meilleures performances, les paramètres doivent être choisis scrupuleusement ce qui engendre un travail de paramétrage très fastidieux. C'est dans ce contexte qu'ont été développées dans la littérature les approches de réglage de paramètres où une phase d'apprentissage permet d'explorer différents jeux de paramètres pour sélectionner le meilleur, et de contrôle de paramètres où les valeurs changent de manière adaptative pendant l'exécution. Dans cette thèse, nous proposons d'utiliser simultanément ces deux approches de paramétrage en proposant une approche appelée multi-configuration automatique d'algorithmes. En particulier, nous explorons plusieurs stratégies basées sur des modèles séquentiels ou probabilistes et nous les comparons aux approches classiques de la configuration automatique d'algorithmes et d'algorithmes adaptatifs. Des expérimentations ont été conduites sur le problème d'ordonnancement de type flowshop de permutation et le problème de voyageur de commerce et montrent la pertinence de l'approche proposée.

***Mots-clés* —** Configuration automatique des algorithmes - Contrôle adaptatif - Recherche locale

Table of Contents

List of Figures

List of Tables

1 | Introduction

Contents

1.1 Context

The work reported in this thesis were successfully completed within the ORKAD [1] team of the CRIStAL [2] laboratory. The ORKAD team aims to simultaneously exploit expertise in combinatorial optimization and knowledge extraction to address optimization problems. While the two scientific areas have developed more or less independently, the synergy between combinatorial optimization and knowledge extraction offers an opportunity, first, to improve the performance and autonomy of optimization methods thanks to knowledge and, secondly, to efficiently solve knowledge extraction problems thanks to operations research methods. The optimization approaches are mainly based on mono and multi-objective combinatorial optimization and lead to the development of open source software.

1.2 Motivation

Metaheuristics, including local search algorithms, provide a good compromise between solution quality and execution time to solve NP-hard combinatorial problems. Metaheuristics are highly configurable algorithms, considering their numerical parameters and/or their algorithmic components, and they can easily be adapted to solve any combinatorial problem. However, the configuration of a metaheuristic – the setting of the numerical parameters and algorithmic components – influences a lot its performance and depends on the problem/instance solved. Finding the best configuration is then a time consuming and tedious manual task and, it becomes challenging to obtain the optimal solutions. Moreover, metaheuristics evolve and move differently in the search space with local charac-

[1] Operational Research, Knowledge And Data: https://www.cristal.univ-lille.fr/equipes/orkad

[2] Centre de Recherche en Informatique, Signal et Automatique de Lille, UMR 9189, Université de Lille: https://www.cristal.univ-lille.fr

teristics depending on the configurations. Thus, it may be beneficial to alter and locally adapt the configuration during the execution. Finding such a best configuration may be handled through parameter tuning and parameter control (Eiben, Hinterding, & Michalewicz, 1999).

Parameter tuning is an offline process where numerical parameters and algorithmic components are optimized before running the metaheuristics to obtain a final solution (López-Ibáñez, Dubois-Lacoste, Pérez Cáceres, Birattari, & Stützle, 2016). *Parameter control* is an online process where numerical parameters and algorithmic components are adjusted during the execution of the algorithm (Karafotias, Hoogendoorn, & Eiben, 2015). In this thesis, we explore the benefits of both methods by proposing hybrid approaches whereby we alternate between multiple configurations tuned before the final execution.

1.3 Contributions

The iterated local search (ILS) is a metaheuristic that evolves in the search space by moving from solution to neighboring solutions. Different strategies and neighborhood operators can be considered and lead to varying ILS performance. A perturbation strategy is applied as soon as a local optimum is found in order to explore a new adjacent region of the search space and to hopefully reach better quality solutions. However, in some search spaces, it is necessary to jump to a region farther from the ones already explored. In this thesis, we focus on a restart iterated local search (R-ILS) able to cross the search space more largely with its restart mechanism. As mentioned before, it is beneficial to modify the parameter values or the strategic components of a metaheuristic during the execution. Moreover, automatic algorithm configuration (AAC) is a parameter tuning process that selects among a large number of parameter values and components the

best configurations for a problem.

The contribution of this thesis is the design of automated multi-configuration ILS. This algorithm is based on the R-ILS and modifies the configuration during the execution. The configurations are tuned and selected by an AAC process using the irace configurator. We propose two multi-configuration models, namely the sequential multi-configuration ILS and the probabilistic multi-configuration ILS. The sequential multi-configuration ILS successively executes a predefined number of tuned configurations while the probabilistic multi-configuration ILS modifies the tuned configurations at each restart according to a fixed or a roulette model. The fixed model executes the tuned configurations in sequence. The roulette model chooses the tuned configuration following a roulette wheel selection.

These approaches are experimented on two well-known combinatorial problems: the Permutation Flowshop Scheduling Problem (PFSP) and the Travelling Salesman Problem (TSP). The experiments show the ability of our approaches to combine elements of both parameter tuning and parameter control.

1.4 Thesis Outline

Chapter 2 describes the context of the work in this thesis. Thus, local search, parameter setting (offline and online), and search strategies are defined. Then we present the two problems studied in this thesis.

Chapter 3 presents the restart-ILS and the random multi-configuration ILS, our two baseline algorithms used for experiments. The first one is the core ILS used in our multi-configuration models and is tuned to find its best parameters. The second one corresponds to a restart-ILS that modifies its parameters during the

run. Experiments are conducted on the PFSP and the TSP to obtain their performance.

Chapter 4 presents the Sequential Multi-Configuration ILS where the tuned configurations are applied successively as soon as a predefined number of evaluations is reached. Experiments are conducted on the PFSP and the TSP. The tuned configurations are presented and discussed for both problems.

Chapter 5 presents the Probabilistic Multi-Configuration ILS where the tuned configurations are applied as soon as the restart mechanism is executed. Two models – fixed and roulette – for choosing the next configuration are detailed. Experiments are conducted on the PFSP and the TSP. The tuned configurations are presented and discussed for both problems. Finally a comparison between the automated multi-configuration ILS and with the baseline algorithms is presented and the results are discussed.

Chapter 6 concludes this thesis and gives some perspectives.

2 | General Context

Contents

2.1 Introduction

Combinatorial optimization problems (COPs) are often NP-hard, and much work has been devoted to the development of metaheuristic algorithms to find good solutions in reasonable time. A powerful, yet conceptually simple, metaheuristic algorithm is Iterated Local Search (ILS) (Lourenço, Martin, & Stützle, 2010). We use it extensively in this work.

Metaheuristics, including the ILS algorithm, are usually composed of many strategy components and parameter values that need to be carefully chosen to obtain a good solution. There exists a set of parameter values that corresponds to the best algorithm configuration for each instance of a problem. Hence, the problem-specific setting of algorithm parameters is needed to achieve the best performance.

In the context of ILS, one example component is the perturbation which is meant to allow the search to escape from the current basin of attraction, while still preserving most of the solution, in the hope of ending up in a better basin of attraction. When this strategy fails, the search can fall back to the local optimum previously reached, remaining stuck in the current basin of attraction. The choice of perturbation and, the strength parameter usually associated, is key to obtaining good ILS performance.

In this chapter, first, we introduce local search (LS). We present different LS strategies and variants. Then we survey multiple automatic design approaches, considering both parameter tuning (off-line) and parameter control (on-line). We describe MH-Builder, the metaheuristics framework upon which much of our work is based. In our work MH-Builder is interfaced with the irace configurator. Finally, we present the two problems that will serve to test our proposed methods: the Permutation Flowshop Scheduling Problem and the Traveling Salesman

Problem.

The material presented here is the basis for the following chapters: Chapter 3 to present the classic restart LS algorithms and some first analyses, the sequential multi-configuration ILS of Chapter 4, and the probabilistic multi-configuration ILS of Chapter 5.

2.2 Local Search

In this section, we describe the general Local Search (LS) algorithm and how to use LS in order to generate practical algorithms for a given problem. The section consists of three parts. First, we describe the LS and its components. Second, we describe some classic LS methods. Finally, we present using restarts as a diversification mechanism.

2.2.1 Description

In brief, local search algorithms (Hoos & Stützle, 2005) follow a repeating pattern where the current solution (s) is replaced with a solution, usually an improving one, obtained in its neighborhood ($N(s)$). It stops when some termination criterion is reached, usually the impossibility of finding any further improving solutions.

This is illustrated in Figure 2.1 where LS creates a finite sequence, or search trajectory, $s_0, s_1, s_2, \ldots, s_m$ of solutions s_i such that for all $i \in \{1, \ldots, m\}$, (s_{i-1}, s_i) is a search step and the neighborhoods are successively explored for the purpose of finding a better solution. Next, we present three key components of a LS: the initial solution, the neighborhood relation and the neighborhood exploration strategy.

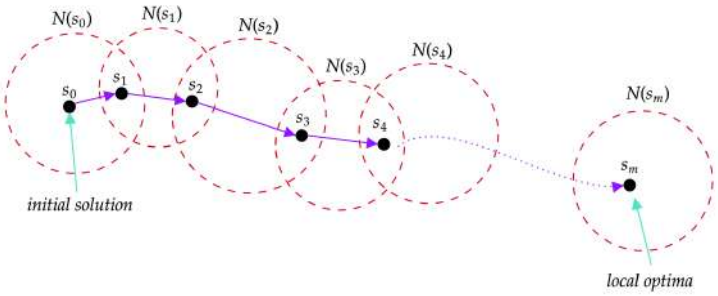


Figure 2.1: A search trajectory - a finite sequence of solutions guided by a neighborhood relation.

2.2.1.1 Initialization

Initialization is the algorithmic component used to build an initial solution. It is an important step as it defines the starting point of the search and, in many problems, it is usually beneficial to start from a relatively good solution in order to get a headstart on the search process. Problem-specific constructive heuristics have been widely investigated. If no specific problem information is available or useful, then it is always possible to use a random solution. Naturally, this random solution should meet all the constraints of the solution representation, e.g., a permutation or a bit string.

Many types of mechanisms are classified under the constructive approach. Conceptually, the simplest of them is probably the greedy approach where a solution is iteratively constructed, choosing the best possible option at each step. More complex constructive heuristics exist, for example the Nawas, Enscore and Ham (NEH) heuristic (Nawaz, Enscore, & Ham, 1983) or the iterated greedy heuristic (Ruiz & Stützle, 2007) for the permutation flowshop problem, or graph-based heuristics for timetabling problems (Sabar, Ayob, Qu, & Kendall, 2011; Burke, McCollum, Meisels, Petrovic, & Qu, 2007). For some problems, like the Quadratic Assignment Problem, a random initial solution suffices.

2.2.1.2 Neighborhood

The concept of local search consists in the traversal of the network structure of the search space, composed of solutions as nodes and edges given by neighborhood relations.

Let us consider an objective or fitness function f, a neighborhood relation N, and a search space Ω. The **objective function** quantifies the quality of a solution, $f : s \rightarrow \mathbb{R}$, where $s \in \Omega$. The **neighborhood relation** is defined by $N : \Omega \rightarrow 2^{\Omega}$, which assigns to a solution $s \in \Omega$ a set of neighboring solutions $N(s) \subseteq \Omega$. Starting from an initial solution, its neighborhood is searched for a better solution according to the given objective function f. If a better solution is achieved, the process is repeated, starting from the better solution, and is repeated until no improving solution can be obtained in a neighborhood. This solution is then called a **local optimum**. Equation 2.1 formalizes the definition of the local optimum s' in the context of a minimization problem without loss of generality.

$$\forall s \in N(s^{'}), f(s^{'}) \leqslant f(s) \tag{2.1}$$

Neighborhoods are closely linked to the solution representation. For example, if we consider the bit-string representation, a simple neighborhood is the bit-flip. This consists in inverting the value of one bit from 0 to 1 and, alternatively from 1 to 0, as illustrated in Figure 2.2. In the permutation space, the swap neighborhood involves switching the values of the i^{th} and j^{th} elements of the permutation.

2.2.1.3 Neighborhood Exploration Strategies

The neighborhood search, or exploration, strategy is a critical component of the local search that determines how the neighborhood is explored and which neigh-

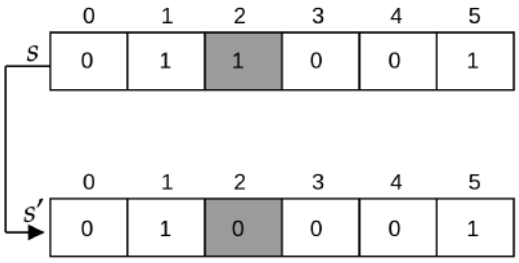


Figure 2.2: The example of simple bit-flip for local search

bor to select at each search step. The strategies are usually simple, for example choosing the first improving neighbor when the neighborhood is explored in random order, or choosing the best neighbor when the neighborhood is explored in a predefined order. We provide additional details on neighbor selection strategies in the context of hill-climbing in Section 2.2.2.1.

2.2.2 Classical Local Search

There are multiple local search metaheuristics (Hoos & Stützle, 2005). Here we present some of the most well-known algorithms: Hill-Climbing (HC) (Papadimitriou & Steiglitz, 1982), Simulated Annealing (SA) (Kirkpatrick, Gelatt, & Vecchi, 1983), Tabu Search (TS) (Glover, 1986), Variable Neighborhood Search (VNS) (Mladenović & Hansen, 1997), and Iterated Local Search (ILS) (Lourenço et al., 2010).

2.2.2.1 Hill-Climbing (HC)

Hill-climbing (Papadimitriou & Steiglitz, 1982) is fast and easy to use, but it rarely yields the best result because it stops at the first local optimum found. It is also called iterative improvement. The pseudocode for strict acceptance in the context of a minimization problem is given in Algorithm 2.1. With strict acceptance, only a strictly improving neighbor can be accepted. It is also possible to use neutral acceptance, where a neighbor can be accepted as long as it is not worse in

terms of the objective function. This can be useful for neutral problems.

A **basin of attraction** corresponds to the set of solutions that, when used as the initial solution of hill-climbing, end up in the same local optimum s. We then say that these solutions are in the basin of attraction of s.

There exist additional strategies that allow the search to continue even after a locally optimal solution has been discovered. Hill-climbing can therefore be a key component in other metaheuristics, for example in Iterated Local Search.

Algorithm 2.1: Algorithm Outline for Hill Climbing

input : $s \leftarrow$ `InitialSolution(`Ω`)`
output: $s \leftarrow$ `Best Solution`

```
1 while (s is not a local optimum) do
2 |   choose s' such that f(s') < f(s), s' ∈ N(s)
3 |   s ← s'
4 end
5 return s
```

We list below different exploration strategies corresponding to how neighbors are visited and selected, the first two being by far the most common.

1. **The first-improvement** selects the first neighbor that improves over the current solution. This is usually paired with the random exploration of the neighborhood.

2. **The best-improvement** explores the whole neighborhood and selects the best. As such, the order of the exploration is not important. Some measure of randomness may be used if any ties need to be broken.

3. **The worst-improvement** (Tari, Basseur, & Goëffon, 2018) also explores the whole neighborhood and chooses the improving move with the lowest-quality.

4. **The late acceptance hill-climbing-improvement** (Burke & Bykov, 2017) accepts non-improving moves when a candidate cost function is better than it was a number of iterations before.

The first- and best-improvement have been widely used in classical exploration strategies in the literature (Ochoa, Verel, & Tomassini, 2010; Whitley, Howe, & Hains, 2013). Best improvement is usually associated with fast convergence in terms of number of iterations but can be slow because the whole neighborhood has to be explored. It may gets stuck in a local optimum early on. First improvement may require more iterations since the rate of improvement is usually smaller than for best-improvement, but the end result is often a better local optimum (as illustrated in Figure 2.3). Worst-improvement has been shown to work well in some select cases although it converges slowly. Late acceptance hill-climbing is better on average on some problems.

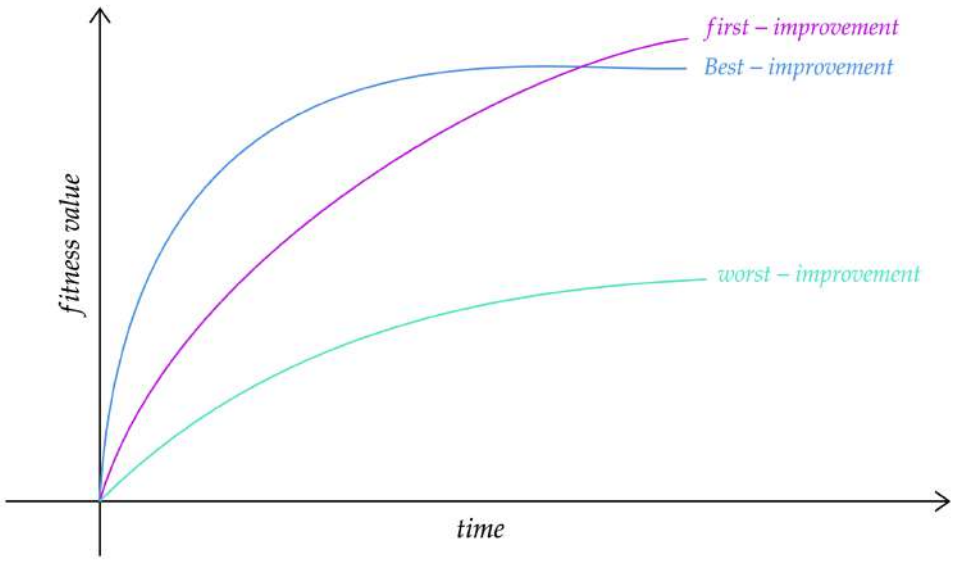


Figure 2.3: The performance of neighborhood exploration strategies (first, best, and worst improvement).

Strict hill-climbing algorithms choose strictly better neighbors at each iteration and stop when a local optimum is met. This is illustrated in Figure 2.4 where the initial solution is shown in blue and the hill-climbing gets stuck in a relatively poor local optimum. Employing additional or different exploration strategies can lead to a better exploration of the search space, and thus finding better local optima and, potentially, the global optimum.

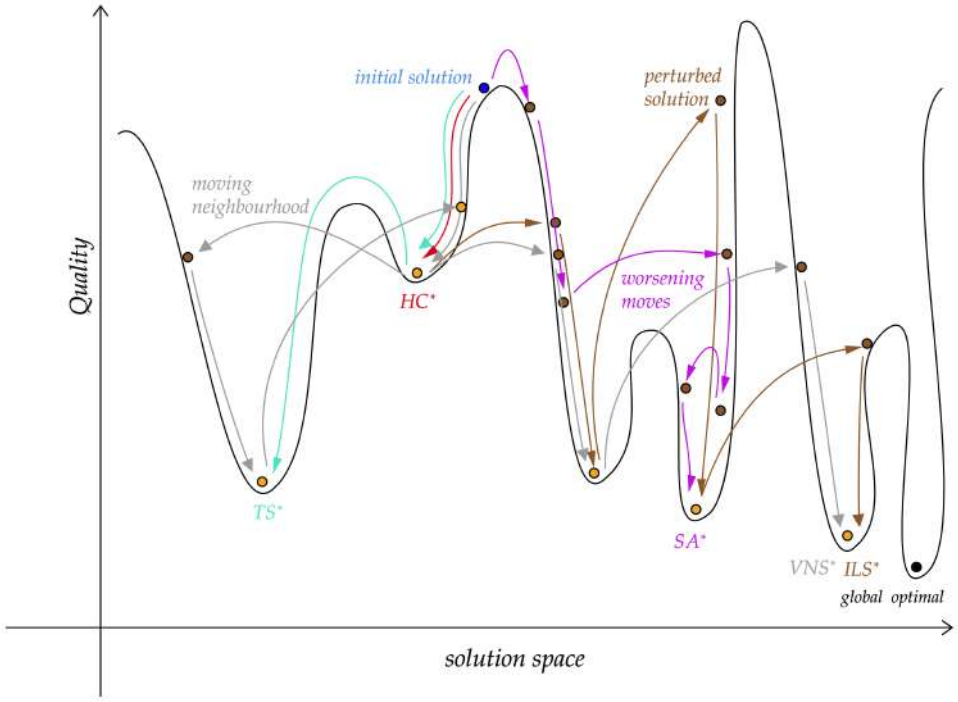


Figure 2.4: Local search algorithm behaviors: Hill-Climbing (HC), Simulating Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS), and Iterated Local Search (ILS).

The different exploration strategies allow for the search to continue when a local optimum is reached. For example, simulated annealing accepts worsening moves on a current solution if no improving solution is available, while tabu search prohibits coming back to previously visited solutions. VNS changes neighborhood

when it gets stuck, meanwhile, ILS applies a perturbation when a local optimum is encountered. We look at each of these algorithms in the following.

2.2.2.2 Simulated Annealing (SA)

Simulated annealing (Kirkpatrick et al., 1983) employs a strategy that reduces the risk of getting stuck in a local optimum (Algorithm 2.2). One neighbor s' is picked at random. If it outperforms the current solution s, it is directly accepted, otherwise it is accepted with a probability equal to $exp(\frac{f(s)-f(s')}{T})$ (with T=temperature, a parameter of the algorithm). This rule is called the Metropolis criterion and the temperature determines how likely it is to perform a worsening choice. Thus, it gives the system a chance to extract itself from a local optimum.

Algorithm 2.2: Algorithm Outline for Simulated Annealing.

input : $s \leftarrow$ `InitialSolution`(Ω)
output: $s \leftarrow$ `Best Solution`

```
k = 1
while (termination condition criterion not met) do
    T ← UpdateTemperature(k, ...)
    choose s' ∈ N(s)
    if (f(s') ≤ f(s)) then
        s ← s'
    else
        s ← s' with probability exp((f(s)−f(s'))/T)
    end
    k ← k + 1
end
return s
```

2.2.2.3 Tabu Search (TS)

The basic concept of Tabu Search was developed by Glover (1986) for solving optimization problems (Glover, 1989, 1990). The concept is to control a search

procedure using a memory structure which explicitly constrains the search direction of the procedure. In each iteration, the best move is chosen among all the admissible solutions. Previously visited solutions (or parts of previously visited solutions) are not admissible in order for the method not to always fall back to the previous visited solutions thus causing inefficient loops. The admissibility of a solution is enforced using the so-called Tabu list. Usually only previous solutions within a specific sliding window are stored in this list. Algorithm 2.3 gives the pseudo code of Tabu search.

Algorithm 2.3: Algorithm Outline for Tabu Search.

input : $s \leftarrow$ `InitialSolution`(Ω), $L \leftarrow$ `Tabu list`
output: $s^* \leftarrow$ `Best Solution`

```
1 L = {}
2 while (termination condition criterion not met) do
3 |   N' ← admissibleNeighbors(s, L)
4 |   s' ← chooseBest(N'(s))
5 |   L ← update(s', L)
6 |   s ← s'
7 end
8 return s
```

2.2.2.4 Variable Neighborhood Search (VNS)

The Variable Neighborhood Search (VNS) proposed by Mladenović and Hansen (1997) is based on several neighborhood structures. In the most basic form of VNS, the neighborhood structures are systematically and deterministically exchanged in a specific order, as presented in Algorithm 2.4. First, the neighborhood structures N_i must be defined and initialized. Usually simpler/smaller neighborhoods are examined first and more complex/larger neighborhoods are examined last. The best or next improvement is chosen as the step function. If an improving solution is found, then the first neighborhood, N_1, is used at the next iteration. Otherwise, the next neighborhood is picked in order to try to escape from the local

optimum.

Algorithm 2.4: Algorithm Outline for Variable Neighborhood Search.

input : $s \leftarrow$ `InitialSolution(`Ω`)`,
$N_k \leftarrow$ `a set of neighborhoods` $k \in \{1, 2, 3, \ldots, k_{max}\}$
output: $s \leftarrow$ `Best Solution`

```
1  k = 1
2  while (termination condition criterion not met) do
3      s' ← RandomSolution(N_k(s))
4      s'' ← LocalSearch(s', N_k)
5      if (f(s'') < f(s)) then
6          s ← s''
7          k ← 1
8      else
9          k ← k + 1
10     end
11 end
12 return s
```

2.2.2.5 Iterated Local Search (ILS)

The search method called Iterated Local Search (ILS) (Lourenço et al., 2010) offers the possibility to escape from locally optimal solutions by applying a perturbation operator. The basic components of a typical ILS are shown in Algorithm 2.5: (i) the function `InitialSolution` generates an initial solution, (ii) a local search is applied to this initial solution using the function `LocalSearch`, which generates a locally optimal solution. Within the while loop, which is terminated by a termination condition (time, number of evaluation of the while loop, etc.), (iii) the `perturbation` function generates a modified solution from the locally optimal solution, (iv) apply a local search algorithm again to this solution, (vi) choose via an acceptance criterion if the new best solution becomes the current solution and loop back to (iii) until the stopping criterion is met.

The perturbation in ILS is usually meant to allow the algorithm to escape from the

Algorithm 2.5: Algorithm Outline for ILS.

input : $s \leftarrow$ `InitialSolution(`Ω`)`
output: $s^* \leftarrow$ `Best Solution`

```
1 s*, s ← LocalSearch(s)
2 while termination criterion is not satisfied do
3 |   s ← Perturbation(s)
4 |   s ← LocalSearch(s)
5 |   if s better than s* then
6 |   |   s* ← s
7 |   end
8 end
9 return s*
```

current basin of attraction, while still preserving most of the solution, in the hopes of ending up in a more favorable basin. When this strategy fails or is too small, as illustrated in Figure 2.5, it can get stuck in the current basin of attraction as the current solution cost does not improve, i.e., the search stagnates. To overcome such an issue, one method is to introduce a restart diversification strategy into the ILS structure that forces a much larger change to be made.

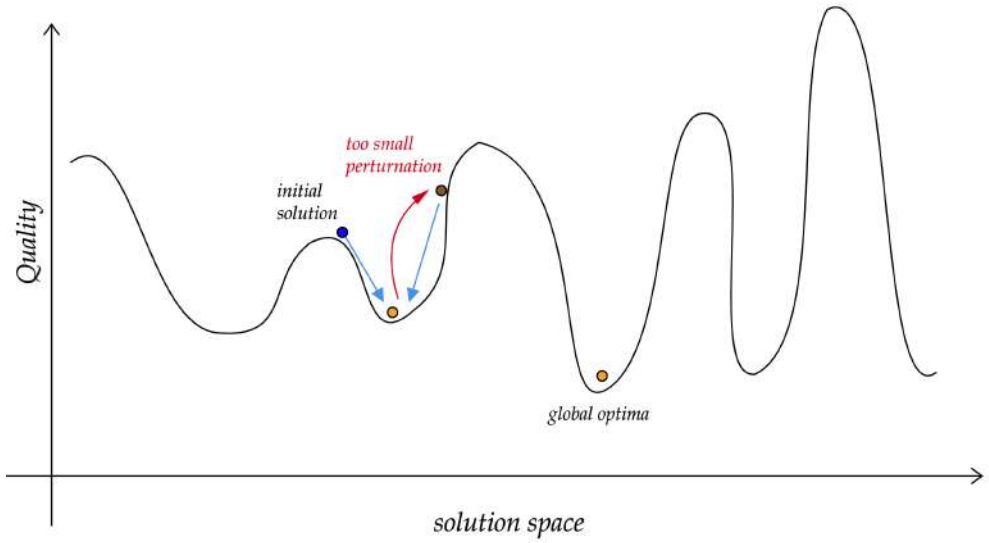


Figure 2.5: Example of the situation where a perturbation is too small.

In the subsequent chapters of this thesis, we will use different versions of ILS, based on hill-climbing combined with some exploration strategies:

- **IHCbest** – Iterated Hill-Climbing with best-improvement acceptance;
- **IHCfirst** – Iterated Hill-Climbing with first-improvement acceptance;
- **IHCworst** – Iterated Hill-Climbing with worst-improvement acceptance;
- **IHClahc** – Iterated Hill-Climbing with late-acceptance, and in particular we use the parameterless version of late acceptance hill-climbing proposed by Bazargani and Lobo (2017).

2.2.3 Repeated Diversification

Restarts in local search algorithms involves restarting the search from a new solution in an attempt to explore a different part of the search space. Restart strategies have been explored for instance by Stützle and Hoos (2002); Hoos and Stützle (2005). Restarts have a powerful influence on the balance of the trade-off between diversification, in order to escape from a current basin of attraction, and intensification, the ability to improve and find better solutions, thus leading to increases in robustness and performance of ILS. The restart can be complete, starting from a completely new solution, or partial, starting from a solution modified with a strong perturbation. We illustrate the behavior of the so-called restart ILS (R-ILS) in Figure 2.6.

When perturbation fails, restarts can be used to start afresh from a new solution and explore a different part of the search space. The deciding criterion for a restart is often a number of stagnation steps. In the literature, common approaches include considering a fixed number of such iterations, some fraction of the total budget (Lobo, Bazargani, & Burke, 2020), instance size (Hoos & Stützle, 2005),

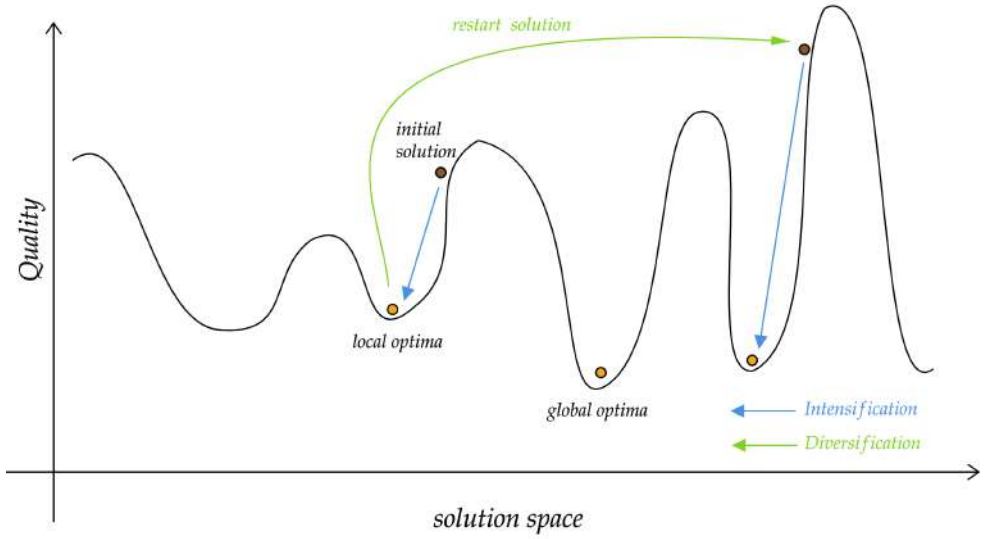


Figure 2.6: Example behavior of combining the diversification of the search space in order to escape the stagnation and the intensification of the better solution found during the search process called Restart Iterated Local Search (R-ILS).

or even neighborhood size (Burke & Bykov, 2017).

A common trigger for restarts is stagnation of the ILS, i.e., failing to improve the best found solution over a certain number of iterations. Algorithm 2.6 shows the pseudo-code for ILS with restarts, where ts is the counter for non-improving iterations, stg is a parameter indicating the maximum stagnation, or number of non-improving iterations, and `diversification()` is the operator that carries out partial or complete restarts.

Algorithm 2.6: Algorithm Outline for Restarts ILS

input : $s \leftarrow$ `InitialSolution(`Ω`)`, $ts = 0$
output: $s^* \leftarrow$ `Best Solution`

```
1  s*, s ← LocalSearch(s)
2  while termination criterion is not satisfied do
3  |   s ← Perturbation(s)
4  |   s ← LocalSearch(s)
5  |   if s better than s* then
6  |   |   s* ← s
7  |   |   ts = 0
8  |   else
9  |   |   if (ts = stg) then
10 |   |   |   ts = 0
11 |   |   |   s ← diversification(s)
12 |   |   else
13 |   |   |   ts++
14 |   |   end
15 |   end
16 end
17 return s*
```

2.3 Automatic Design of Algorithms

As we have seen, algorithms and metaheuristics are composed of different components and parameters. Automatic algorithm design involves choosing the right components and parameters in some specific context, for example for a specific problem or class of problem instances. Eiben et al. (1999) identified two main ways of making such a choice: first tuning, an off-line process which involves finding a configuration of well-performing components/parameters of a given algorithm before execution of an algorithm; second control, an online process where the components/parameters change on the fly (Figure 2.7).

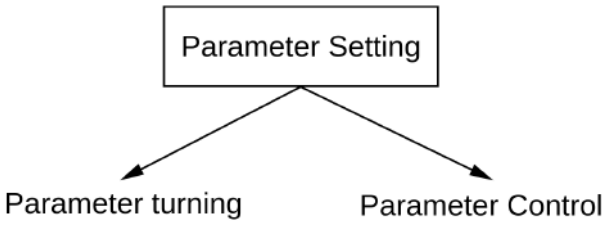


Figure 2.7: Parameter Setting Model

2.3.1 Parameter Tuning

Parameter tuning or automatic algorithm configuration (AAC) can be considered as an optimization problem. The objective is to identify the best configuration out of a set of possible configurations that are assessed on a set of training instances. The goal is to find the configuration best adapted to the instances of the problem to solve. The basic elements for automatic configuration algorithms consist of the five components: search space, instance set, configurator, target algorithm, and optimization configuration. The interaction between the configurator and the target algorithm is illustrated in Figure 2.8. The pseudocode for AAC is given in Algorithm 2.7

Algorithm 2.7: Algorithm Outline for Generic AAC

input : $\Theta \leftarrow$ `Search Space(`$\Theta_1, \Theta_2, \ldots$`)`
$\mathcal{I} \leftarrow$ `Instance Set(`$I_1, I_2, \ldots$`)`
output: $\Theta_{elite} \leftarrow$ `Optimisation Configuration`

1 $\Theta_i \leftarrow$ `initial configuration(s)`(Θ)
2 **while** *termination criterion is not satisfied* **do**
3 $I_i \leftarrow$ `choose instance of problem`$(\mathcal{I})$
4 $\Theta_{elite} \leftarrow$ `evaluation`(I_i, Θ_i)
5 $\Theta_{new} \leftarrow$ `initial new configuration(s)` $\cup\, \Theta_{elite}$
6 **end**
7 **return** Θ_{elite}

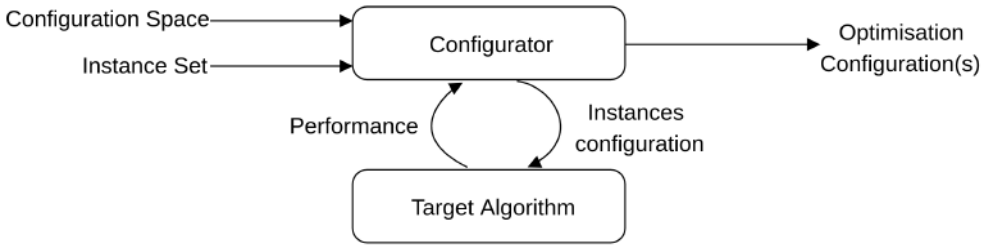


Figure 2.8: Automatic configuration of a given parameterized target algorithm for a given set of problem instances

Search Space: the search space Θ is a set of configurations $\Theta_1, \Theta_2, \ldots, \Theta_i$ which a target algorithm searches. In particular, the search space may include a number of the parameter types.

Instance Set: the instance set $\{I_1, I_2, \ldots, I_i\}$ or single instance $\mathcal{I}$ of a problem is required by the configurator, to evaluate the configurations.

Target Algorithm: the configurator optimizes a target single or multi-objective algorithm.

Optimisation Configuration: the configurator may return one configuration or set of configurations. Additionally, the configurator can return a policy that maps instances to configurations.

Configurator: the configurator tests a number of pairs (instances and parameters), measures the performance of the algorithm and then returns the best configuration found. For example, we may cite from the literatur: irace based on statistics (López-Ibáñez et al., 2016), that uses racing and Friedman tests – non-parametric two-way analysis of variance by ranks, that was then modified into Iterated F-race based (Balaprakash, Birattari, & Stützle, 2007); ParamILS (Hutter, Hoos, Leyton-Brown, & Stuetzle, 2009) based on Iterated Local Search across the space of configurations, and MO-ParamILS (Blot, Hoos, Jourdan, Marmion, & Trautmann, 2016) its multi-objective version; SMAC (Hutter, Hoos, & Leyton-Brown, 2012) based on random forests which is a machine learning approach for classification using an ensemble of decision trees; GGA+ (Ansótegui, Malitsky, Samulowitz, Sellmann, & Tierney, 2015; Ansótegui, Sellmann, & Tierney, 2009) based on a genetic algorithm and random forests – it accepts either Gaussian process model or random decision tree; SPRINT-race (Zhang, Georgiopoulos, & Anagnostopoulos, 2013) which exploits statistic racing, which involves considering two metrics for model selection in machine learning simultaneously. We summarize the different configurators and their specific features in Table 2.1.

Table 2.1: Example of the configurator and specific features.

Configurator	Reference	Configuration Space	Instance Set	Target Algorithm	Optimisation Configuration	Specific Method
F-race	Birattari, Stützle, Paquete, and Varrentrapp (2002)	small	set	single	single	Racing with F-test
CALIBRA	Adenso-Díaz and Laguna (2006)	small	set	single	single	Factorial design with Local Search
Iterated F-race	Balaprakash et al. (2007)	large	set	single	single	F-race with re-sampling
ParamILS	Hutter et al. (2009)	large	set	single	single	Iterated Local Search
GGA	Ansótegui et al. (2009)	large	set	single	single	Genertic Algorithm with Racing
SMAC	Bartz-Beielstein, Lasarczyk, and Preuss (2010)	large	set	single	single	Gaussian process model or Random decision tree
SPRINT-race	Zhang et al. (2013)	small	set	multi	set	Sequential probability test with racing
GGA+	Ansótegui et al. (2015)	large	set	single	single	GGA with random forest model crossover
irace	López-Ibáñez et al. (2016)	large	set	single	single	F-race with Racing
MO-ParamILS	Blot et al. (2016)	large	set	multi	set	Iterated Local Search
HORA	de Moraes Barbosa and Senne (2017)	small	set	single	single	Racing with Local Search
DAC	Biedenkapp, Bozkurt, Eimer, Hutter, and Lindauer (2020)	large	set	single	single	Reinforcement Learning
PyDGGA	Ansótegui, Pon, Sellmann, and Tierney (2021)	large	set	single	single	Genetic Distributed GGA

2.3.2 Parameter Control

Parameter control aims at adjusting parameter values during execution. Eiben et al. (1999) describes three categories of algorithms: (1) deterministic; (2) adaptive; and (3) self-adaptive. Although, it was initially applied only to evolutionary algorithms, now it is used in a broader sense.

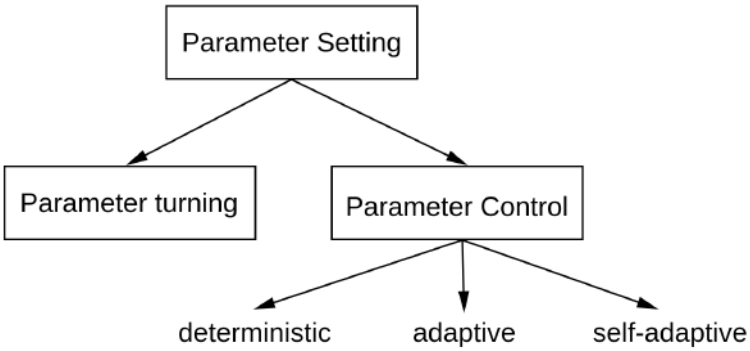


Figure 2.9: Parameter Control Taxonomy

1. **Deterministic**: the parameter value changes using certain deterministic rules (Eiben et al., 1999) such as time, time-dependent, scheduled, progress-independent, i.e. and not any other feedback to the optimization process.

2. **Adaptive**: the parameter value is modified according to some pre-described rules and uses feedback to the optimization process such as diversity of the solutions encounterd, function value of the solutions encounterd, etc.

3. **Self-adaptive**: the parameter value is encoded in a new genotype and evolves during the optimization process. The literature showed that the self-adjusting approach can for example link fitness with crossover rate (Cheng, Li, & Lin, 2019) or mutation rate (Doerr, Witt, & Yang, 2018; Fan & Yan, 2016; Hevia Fajardo & Sudholt, 2021) in evolution algorithms.

Over the 20 years following Eiben et al. 's classification in 1999, new ideas and

methodologies have been proposed by Doerr and Doerr (2018) across 5 subcategories that had been identified: state-dependent, success-based, learning-inspired, endogenous/self-adaptive, and hyper-heuristics. These are illustrated in Figure 2.10.

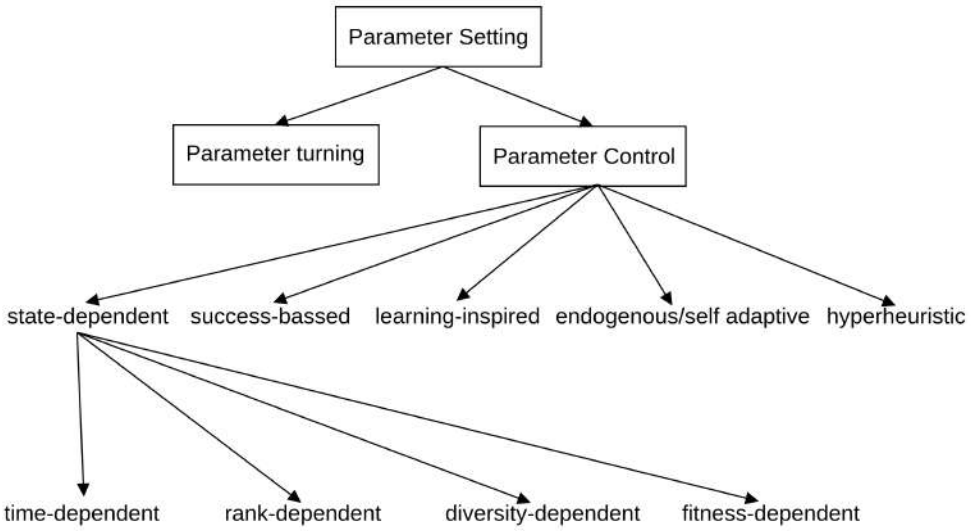


Figure 2.10: An extended version of the classification schema

1. **State-Dependent Parameter Selection**: it depends only on current state of the optimization process. This approach can be split into subcategories:

   (a) **time-dependent** corresponds to some strategy based on time where the configuration is changed at time specific points during the run. The time-dependent aspect can be based iteration count (Henderson, Jacobson, & Johnson, 2003), fitness evaluation (Kirkpatrick et al., 1983), CPU time, etc. It corresponds to deterministic control of parameters in the taxonomy proposed by (Eiben et al., 1999).

   (b) **fitness-dependent** corresponds to some strategy based on the fitness of solutions encounterd where the configuration is changed based on the value of said fitness (Oliveto, Lehre, & Neumann, 2009; Fialho, Da Costa, Schoenauer, & Sebag, 2008). It corresponds to adaptive

parameter control in the taxonomy by Eiben et al. (1999).

(c) **rank-dependent** is based on fitness ranking. Böttcher, Doerr, and Neumann (2010) proposed such a raking-based approach to select a fitness and uses it to modify the mutation rate.

(d) **diversity-dependent** have been studied empirically. However, the theory of evolutionary algorithms community has given them far less consideration (Doerr & Doerr, 2018).

2. **Success-Based Parameter Selection**: it corresponds to adaptive parameter control in the taxonomy of Eiben et al. (1999). The current parameter value is adjusted depending on whether or not the last step has been successful. For example, the current iteration is successful when it actually produces offspring with a fitness value that has been better than the previous best (Lässig & Sudholt, 2011; Rowe & Sudholt, 2014).

3. **Learning Inspired Parameter Selection**: it aims at exploiting longer search history, which most commonly involves reward or learning adjustment rule. The process includes multi-armed bandits (MAB) (Blot, Hoos, Kessaci, & Jourdan, 2018; DaCosta, Fialho, Schoenauer, & Sebag, 2008), reinforcement learning (Karafotias et al., 2015), adaptive pursuit (Rajaraman & Sastry, 1996; Thierens, 2005), probability matching (Thierens, 2005), to dynamically find elite configurations.

4. **Endogenous/self-adaptive Parameter Selection**: it is the same as the self-adaptive control systems for parameters. The name endogenous parameter control is better known as encoding the parameters in the genotype and enabling them to develop through the general processes of variety and choice of the evolutionary system (Hevia Fajardo & Sudholt, 2021).

5. **Hyperheuristic Parameter Selection** is an approac which operates on a set of low level heuristic, selects an algorithm and executes it for a period of time before reassessing the heuristic that is next to be used. Hyperheuristics are mainly intended to automate the selection and configuration of algorithms, so that distinct algorithmic concepts can profit from each step. Hyperheuristics have been applied to a metaheuristics for single-objective (Zamli, Din, Kendall, & Ahmed, 2017; Kalender, Kheiri, Özcan, & Burke, 2013) and multi-objective (Yang, Zhang, & Li, 2021; Guizzo, Vergilio, Pozo, & Fritsche, 2017).

2.3.3 Conclusion

Several algorithms for solving the combinatorial optimization problems (COP), such as local search or metaheuristic algorithms, include algorithm-specific parameters or several design choices that must be strictly set to achieve the best performance. Most frequently, obtaining good values for these parameters is a time-intensive, tedious manual task, and finally leads to a biased evaluation of their performance. The problem of finding parameter values to achieve the best performance can be approached through parameter tuning and parameter control (Eiben et al., 1999).

Parameter tuning or automatic algorithm configuration (AAC) automatically determines a configuration for the optimized performance of an algorithm on a given set of training problem instances. The target algorithm is configured by a so-called configurator such as irace (López-Ibáñez et al., 2016) or ParamILS (Hutter et al., 2009). In this thesis, we have used irace configurator for automatic configuration. On the other hand, parameter control is an online process, where relatively few configurations can be explored effectively. In the following chapters in this thesis, we will explore how multiple configurations that change during the run of an op-

timisation algorithm can be configured using automatic algorithm configuration.

2.4 Framework and Experiment Management

In this section, we will present the MH-Builder framework that allows us to implement metaheuristics based on combinations of components. We also present the irace configurator that is used in this thesis. A large part of the work in this thesis has revolved around the extension of the MH-Builder framework to accommodate and implement the contributions we present in subsequent chapters.

2.4.1 MH-Builder

MH-Builder is an object-oriented software platform for building flexible metaheuristics for solving combinatorial optimization problems. The MH-Builder is developed in C++ (C14 or higher) and it operates on UNIX systems (Linux, MacOS) and Windows. This platform is developed by the ORKAD team. The aim is to ultimately make it publicly available. As illustrated in the Figure 2.11, it contains four connected modules that describe an algorithm.

The modules that constitute the MH-Building platform are the following:

1. **core** for the implementation of fundamental aspects of metaheuristics (solution, fitness, etc.)

2. **opt** is the implementation of metaheuristics dedicated to problem-independent optimization, it is divided into several sub-categories:

   (a) **criterion** for the implementation of components dedicated to the stopping search algorithm (time, iteration, evaluation).

   (b) **single solution** (Sae-Dan, Kessaci, Veerapen, & Jourdan, 2020; Clay,

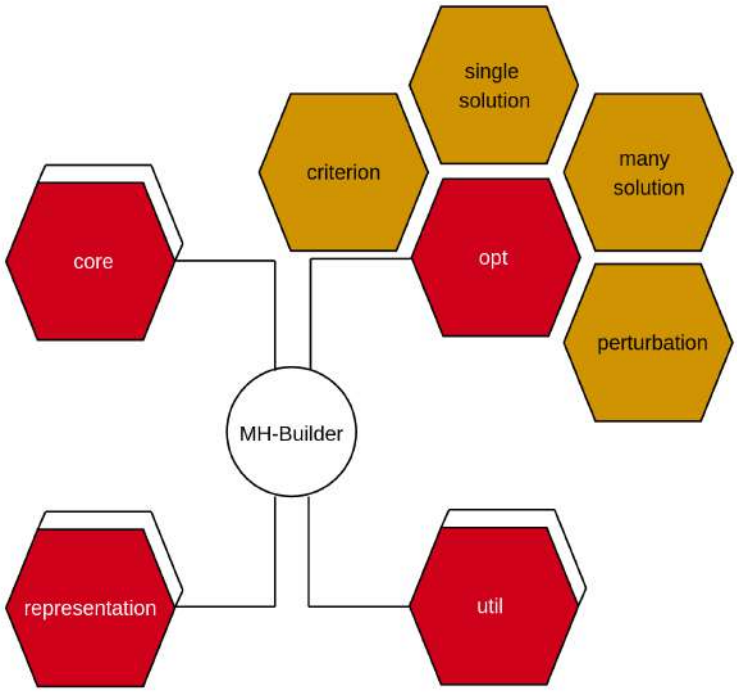


Figure 2.11: The modules of the MH-Builder platform

Mousin, Veerapen, & Jourdan, 2021; Sae-Dan, Kessaci, Veerapen, & Jourdan, 2021) for the implementation of metaheuristics dedicated to single solution optimization (single-objective local search, evaluation algorithm, etc.).

(c) **many solution** (Pageau, Blot, Hoos, Kessaci, & Jourdan, 2019; Szczepanski, Mousin, Veerapen, & Jourdan, 2020; Szczepanski et al., 2021) for the implementation of metaheuristics dedicated to multi-objective optimization.

(d) **perturbation** for the implementation of components dedicated to perturbation methods.

3. **representation** (Feutrier, Kessaci, & Veerapen, 2021; Tari, Hoos, Jacques, Kessaci, & Jourdan, 2020) for the implementation of components dedicated

to a particular type of problem (rulemining, timetabling, etc.)

4. **util** for the implementation of components dedicated to useful functions: parser, random number generator, time management, etc.

We will now take a look at the MH-Builder framework. First of all, Figure 2.12 shows the current implementations of the *Local Search* object in the MH-Builder of *Single Solution*. We notice five algorithms, that we will detailed in previous sections.

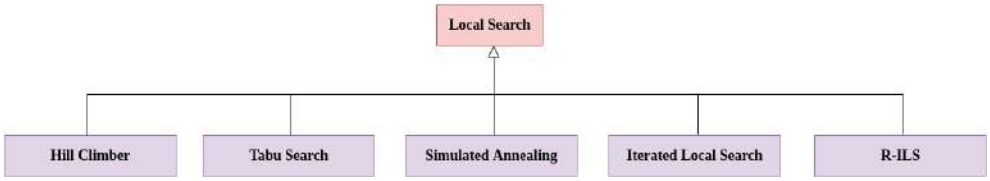


Figure 2.12: The diagram of the Local Search implementations currently existing in MH-Builder

A single solution metaheuristic under MH-Builder is a Local Search object. We explain each of these notions in the following.

2.4.1.1 Hill Climbing (HC)

As seen in section 2.2.2.1, a hill climber is a type of local search. The aim is to improve a current solution by exploring its neighborhood. The best neighbor replaces the current solution if it is better. Thus, each part of a local search is symbolized by an object type. Figure 2.13 shows a schematic of the Hill Climber class and the algorithm implemented in this class can be seen in Algorithm 2.1.

The six objects necessary for the instantiating of a Hill Climber object are :

1. **InitialSolution**: This object must be used to require a method to build an initial solution.

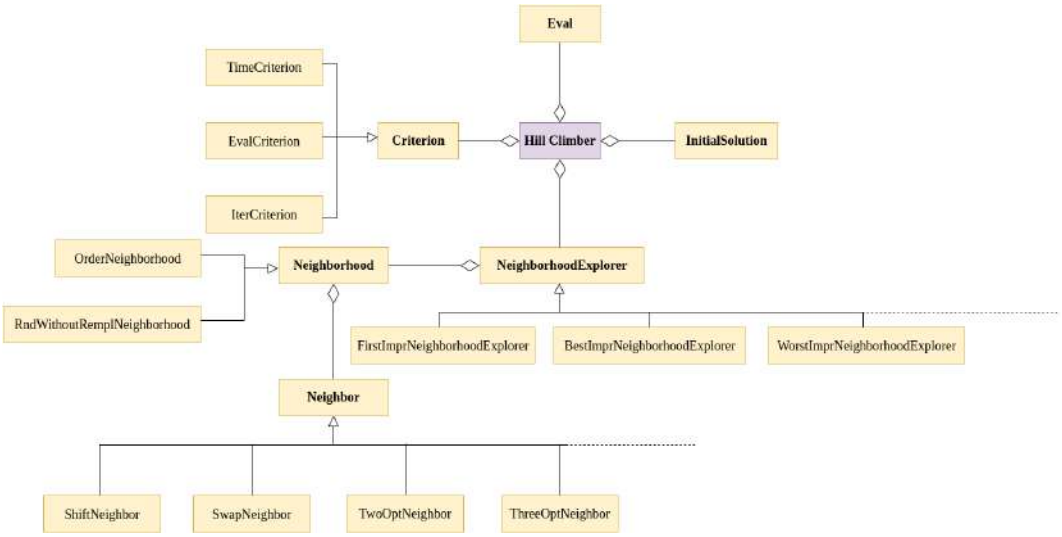


Figure 2.13: The schematic of a Hill Climber object

2. **Eval**: This object is used to evaluate the quality of a solution. It is not the object that will be used to evaluate the neighbors during our neighborhood exploration. This object is of interest in the case where the algorithm is given an initial solution or solution improved that has not yet been evaluated. An incremental evaluation is used in other cases in order to make the computation more efficient.

3. **Criterion**: This object handles the management of stopping an algorithm which consists of three criteria: TimeCriterion, EvalCriterion, and IterCriterion.

4. **Neighborhood**: This object handles the management of the neighborhood. Two exploration orders exist in MH-Builder:

   (a) RndWithoutRemplNeighborhood: explore neighbor solutions uniformly at random (`random`).

   (b) OrderNeighborhood: always parse the neighborhood in some predefined order (`order`).

5. **Neighbor**: Its variants are neighborhood-based algorithms and require an operator to generate neighbor solutions from the current solution such as `shift`, `swap`, `2-opt`, `3opt`, i.e. We will explain each of these notions in Sections 2.5.1.2 and 2.5.2.2.

6. **Explorer**: This object allows the selection strategy of the best neighbor to be specified such as `first-improvement`, `best-improvement`, `worst-improvement`, and `lahc-improvement`. The four strategies available in MH-Builder where detailed in Section 2.2.2.1.

2.4.1.2 Simulated Annealing (SA)

Simulated annealing is a local search that draws on a physical process and is described in Section 2.2.2.2. The main advantage of simulated annealing is that, unlike the hill climber, it allows us to go backward in obtaining better solutions. The interest of such a possibility is to allow, if necessary, to escape local optima during the search. Simulated annealing can be seen as a search with a diversification phase that gradually leads to an intensification phase.

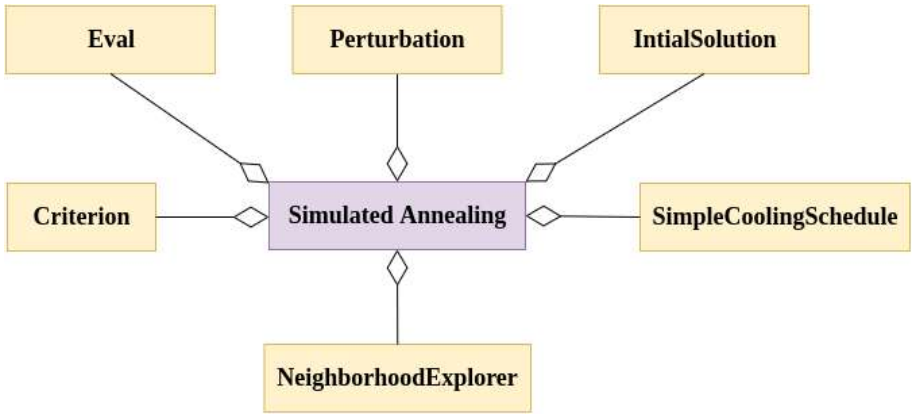


Figure 2.14: The schematic of a Simulated Annealing object

Figure 2.14 shows the simulated annealing implementation, we can notice the SimpleCoolingSchedule object which concerns the temperature cooling principle.

2.4.1.3 Tabu Search (TS)

The particularity of Tabu Search is the notion of memory. In this type of local search, a so-called tabu list is used to remember which solutions have already been visited (see section 2.2.2.3). Tabu Search is represented in MH-Builder by a specific object: the Tabu Search object. Figure 2.15 details how a Tabu Search is constituted in MH-Builder. If we carefully look at this figure and compare it with that of the Hill Climber, you can see that the objects are the same. The differences between a Hill Climber and Tabu Search object are symbolized by one generic object: TabuList object takes care of the management of the tabu list. In MH-Builder, it is an object that the user must define.

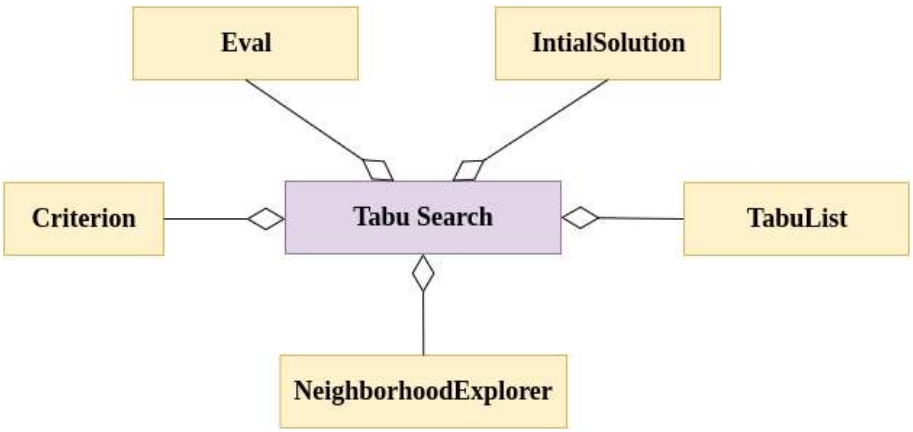


Figure 2.15: The schematic of a Tabu Search object

2.4.1.4 Iterated Local Search (ILS)

This metaheuristic is one of the four that has been implemented in MH-Builder. As its name indicates, this iterated local search is based on local search and the addition of a perturbation mechanism (see section 2.2.2.5). Figure 2.16 shows a diagram of an iterated local search object that represents this type of local search in MH-Builder. The algorithm 2.5 details the implementation of iterated local search in MH-Builder.

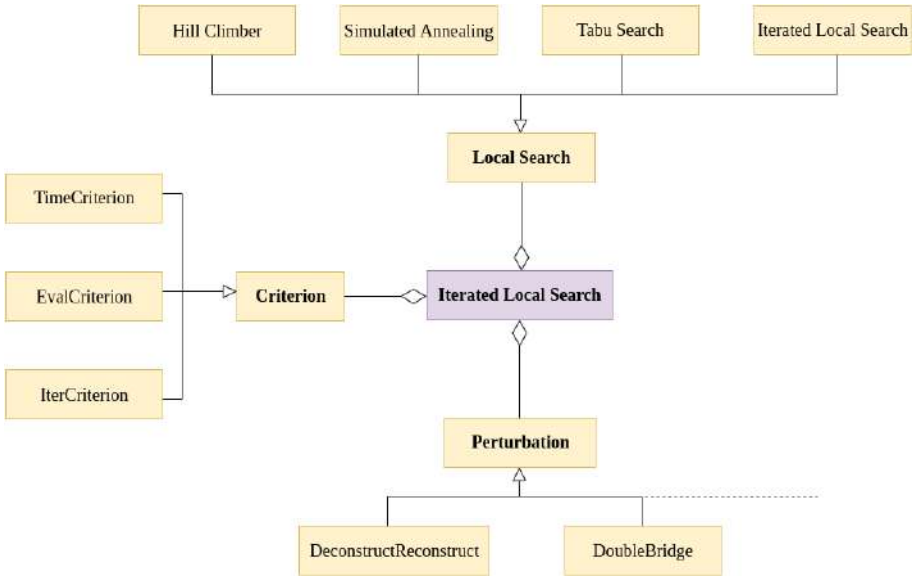


Figure 2.16: The schematic of a Iterated Local Search object

Two objects are needed to instantiate an iterated local search object:

1. **Local Search**: The classical ILS can use any type of local search available in MH-Builder or even other types of local searches.

2. **Perturbation**: An ILS uses the perturbation to escape from a local optimum. The aim is to escape from the basin of attraction of the current local optimum, ideally to find a better optimum.

2.4.1.5 Restart Iterated Local Search (R-ILS)

Figure 2.17 shows the diagram of a R-ILS object that represents this type of local search with restarts in MH-Builder. The algorithm 2.6 details the implementation of this object. It is consistent with the idea to balance the trade-off between intensification of the best solution and diversification of the solutions encountered.

The two objects necessary for the instantiating of a R-ILS object are:

1. **diversification**: It is necessary to perform a jump in the search space to find

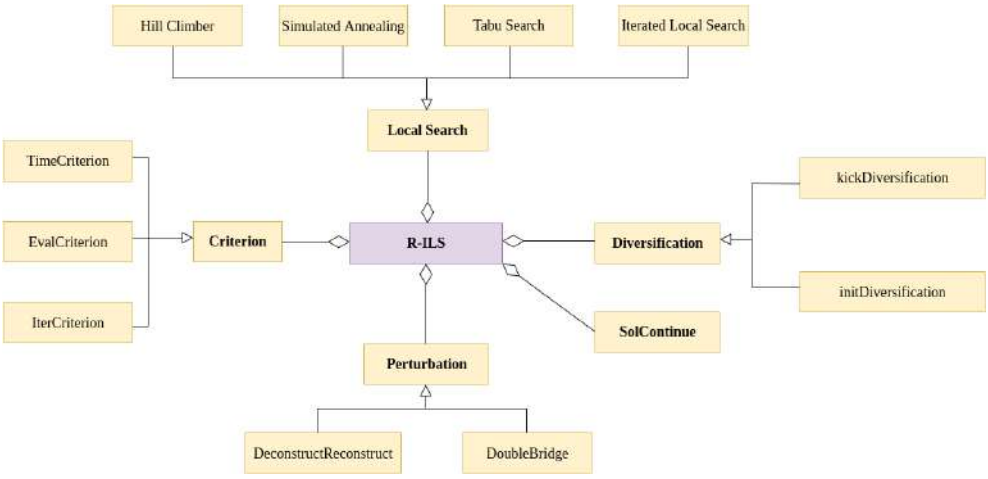


Figure 2.17: The schematic of a Restart Iterated Local Search object

a better region to explore. This is achieved via a full or sometimes a partial restart. We retain two different strategies to make a large step possible: either the initialization is performed (equivalent to a full restart) or a (fairly large) `kick` is applied in the current solution. This last strategy could consist in applying the swap operator $k \in \{3, 4, 5, 10\}$ times for example.

2. **SolContinue**: This is the criterion for triggering a restart. The SolContinue object corresponds to a maximum number of stagnation (ts). In MH-Builder, it is an object that the user must define.

2.4.2 Iterated Racing (Irace)

Irace (López-Ibáñez et al., 2016) is the configurator that we use in the subsequent chapters of this thesis to find good algorithm configurations. Given an algorithm ($\mathcal{A}$), a parameter space Θ of configurations of $\mathcal{A}$, and a set of the problem instances, irace samples the configuration space and the best configurations are selected using racing. A race starts from a certain number of sampled configurations. At each step of the race, the configurations are evaluated on the instances and those that perform statistically worse than at least one of the others are deleted. Friedman's non-parametric test is used for this purpose. The race then continues with the remaining configurations. This process is repeated until a certain number of configurations remain, a certain number of instances have been evaluated or the computation budget has been used up. In irace, the racing is repeated multiple times. Each time a new race starts, elitism is used to include the best previously found solutions in the sample of configurations used to start the race.

The irace configurator considers that only one configuration is used throughout a single run of an algorithm on some specific instance. However, for our purposes we will need to be able to represent that multiple configurations can occur during a single run. This is done by re-encoding a configuration to represent multiple configurations as well as the information about when to switch configurations. We use a budget of 5000 runs. We consider a specific set of training instances and another set of validation instances to minimize any potential overfitting (Figure 2.18).

Figure 2.18: The automated design system component and the arrows define the information flow

2.5 Problems and Instances

Combinatorial optimization is a very important area in operations research, applied mathematics and computer science. It includes a large number of difficult combinatorial optimization problems (Papadimitriou & Steiglitz, 1982) from real world applications in different fields such as manufacturing, the financial sector or the military (Barahona, Grötschel, Jünger, & Reinelt, 1988). The objective of these combinatorial problems is to find a better solution in a discrete space of feasible solutions, which respects a set of conditions, also called constraints. The evaluation of a solution is carried out using a function known as the objective function. A better alternative (optimal solution) is a feasible solution that minimizes or maximizes, depending on the context, the objective function.

We are going to consider a context of minimization for two combinatorial problems: the permutation flowshop scheduling problem and the traveling salesman problem. In this section, we will also describe the instances we use and how we group instances with similar characteristics. The groups are created in order to allow us to study what configurations work better in different contexts in the chapters that follow.

2.5.1 Permutation Flowshop Problem (PFSP)

The permutation flowshop problem (PFSP) is among the most well-known classical NP-hard combinatorial optimization problems. The aim of the problem is to find an optimal sequence of jobs according to some measurement.

In the literature, Fernandez-Viagas, Ruiz, and Framinan (2017); Ruiz and Stützle (2007) and Nawaz et al. (1983) studied the minimization of the *makespan* (i.e., the total completion time of the schedule) which we use in this thesis. Other objectives

include minimization of total flowtime Allahverdi and Aldowaisan (2002); Framinan, Leisten, and Ruiz-Usano (2005); Dong, Chen, Huang, and Nowak (2013); Rajendran (1993) and minimization of total tardiness (Valente & Alves, 2008; Framinan & Leisten, 2008).

PFSP consists in scheduling a set of n jobs $(J_1, \ldots, J_n)$ jobs on a set of m machines $(M_1, \ldots, M_m)$. Machines are so-called critical resources because at most one task can be executed at the same time on a machine. Job J_i is composed of m consecutive tasks to be performed in order on the m machines. Each task has a specific execution time on each machine, with $J_{i,k}$ indicating that J_i is run on machine k with an associated processing time $P(i, k)$. For the permutation flow shop, each machine processes the jobs in the same order and a solution is represented by a permutation $\Pi = (\Pi_1, \Pi_2, \Pi_3, \ldots, \Pi_n)$. In this thesis, we are trying to minimize the makespan, i.e, the completion time of the latest scheduled task. The completion times $C(\Pi_i, j)$ for each job on each machine for a permutation Π are given as follows:

$$C(\Pi_1, 1) = P(\Pi_1, 1) \tag{2.2}$$

$$C(\Pi_i, 1) = C(\Pi_{i-1}, 1) + P(\Pi_i, 1) \quad \forall i \in \{2, \ldots, n\} \tag{2.3}$$

$$C(\Pi_1, j) = C(\Pi_1, j-1) + P(\Pi_1, j) \quad \forall j \in \{2, \ldots, m\} \tag{2.4}$$

$$C(\Pi_i, j) = max(C(\Pi_{i-1}, j), C(\Pi_i, j-1)) + P(\Pi_i, j) \quad \forall i \in \{2, \ldots, n\}, \forall j \in \{2, \ldots, m\} \tag{2.5}$$

Figure 2.19 shows the example of a solution to a permutation flowshop problem where three jobs (J_1, J_2, J_3) are scheduled on four machines (M_1, M_2, M_3, M_4). The objective is to minimize the makespan, which is equal to 12 here.

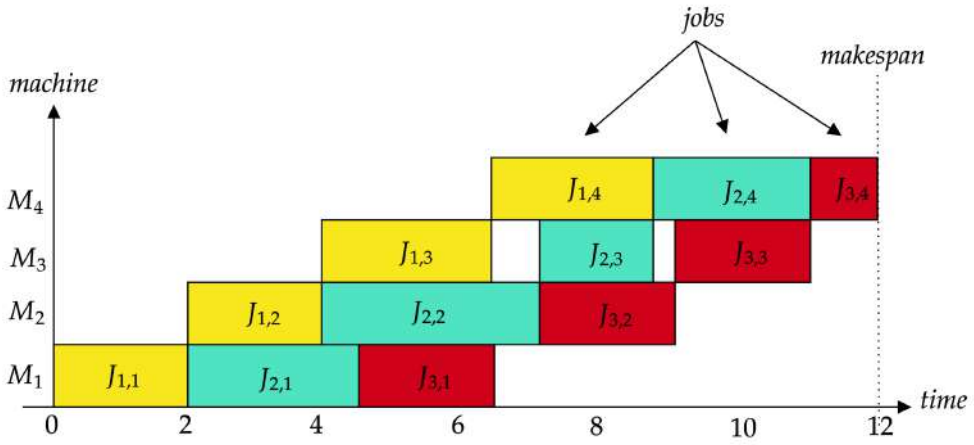


Figure 2.19: An example of PFSP schedule for $n = 3$ jobs, $m = 4$ machines

2.5.1.1 Instances

The Taillard instances (Taillard, 1993) are widely used in the literature to assess the performance of algorithms. Different sizes are available depending on the number of jobs and machines. The processing times of jobs are integers uniformly generated in the range $[\![1, 100]\!]$. For each size, ten instances are provided. In this study, to assess the performance, we keep only larger instances with $\{50, 100, 200\}$ jobs and $\{10, 20\}$ machines as follows: 50×20, 100×10, 100×20, 200×10, and 200×20.

In addition, we generate 100 training instances per instance size for the automatic configuration phase following the same uniform distribution of processing times. This prevents the over-fitting of parameter values to the actual test instances and mimics real-life situations where the one would want to apply the configuration step infrequently and then use it the configuration multiple times on new instances.

In order to assess what the best configurations on different groups of instances, we create instance sets, or scenarios. We have four scenarios for the PFSP: S_{all} that contains all instance sizes, $S_{N=100}$ that contains all the instances with 100 jobs,

$S_{N=200}$ that contains all the instances with 200 jobs, and $S_{M=20}$ that contains all the instances with 20 machines.

2.5.1.2 Neighborhood Operator

We consider two PFSP neighborhood operators in this thesis: shift and swap neighborhood operators. The shift neighborhood consists in selecting a job and inserting in another position in the permutation. Thus, for a job position in i, it can be re-inserted at a position $j \neq i$. The jobs positioned between these two positions are thus shifted. Figure 2.20 shows the shift operator.

The swap consists in exchanging the place of two jobs of the permutation. Thus, for positions i and j where the position $i \neq j$, the job in i moves to j and the job in j moves to i. Figure 2.21 shows the swap operation between positions $i = 2$ and $j = 5$.

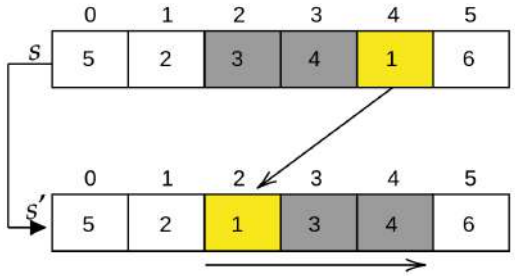


Figure 2.20: Example of PFSP schedule for shift neighborhood operator

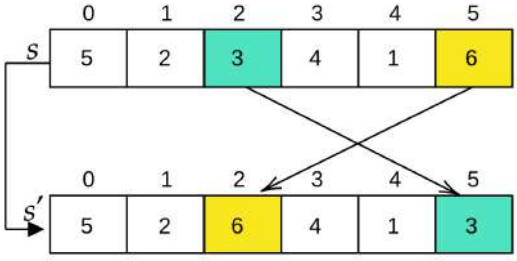
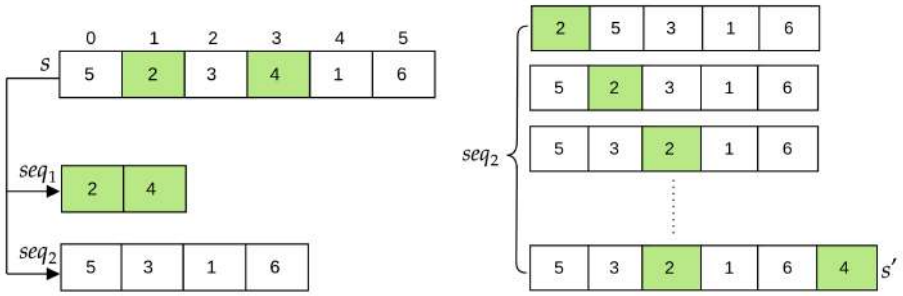


Figure 2.21: Example of PFSP schedule for swap neighborhood operator

There are other interesting operators for the PFSP, such as the deconstruct and reconstruct operator (Ruiz & Stützle, 2006) used in the Iterated Greedy (IG) algorithm. First, the deconstruction phase chooses randomly to reject the jobs (d) for the current solution and the reconstruction phases inserts the jobs back in different better positions. In subsequent chapters, we will refer to this operator as IG-D/R. As an example, Figure 2.22 shows that the current solution (s) represents a job sequence, then we reject d=2 jobs (2 and 4) from the current solution (s). We then have two job sequences: seq_1 with all jobs rejected and seq_2 is the partial sequence. In the next step, we insert job 2 of seq_1 in the partial sequence so as to obtain the best partial sequence (seq_2). We continue the same way for other jobs. Finally, the best partial sequence becomes the current solution (s').



(a) Deconstruction phase (b) Reconstruction phase

Figure 2.22: Example of PFSP schedule for Deconstruct and Reconstruct

2.5.2 Traveling Salesman Problem (TSP)

The TSP is probably the best-known example of an NP-hard combinatorial problem. It is often formulated in terms of visiting a number of cities. The task is that all cities are visited exactly once during a round trip, where the cities are all connected by edges whose weights are calculated by the distances of the individual cities from each other, and the total distance of the round trip must be minimized. In addition, the round trip must end in the city where it began. Here, the cities are

considered as nodes (V) and the connecting roads as edges (E), with each road having a positive weight.

More formally, the problem is described by a complete graph $G = (V, E)$ with edge weights $d_{ij} > 0 \; \forall (i, j) \in E, i \neq j$. We are looking for a so-called Hamiltonian cycle with the lowest edge (sum) weight. Due to the fact that each vertex may be visited exactly once, the solutions of a TSP are also called "permutations" (Jungnickel, 1999). Figure 2.23 shows an example of a Hamiltonian cycle, in blue, in a TSP with 5 cities.

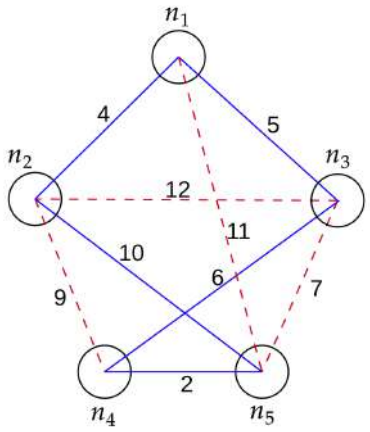


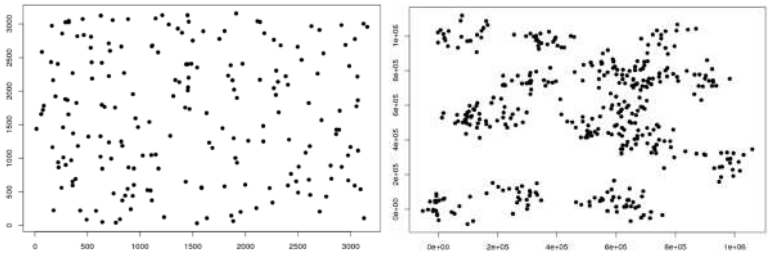
Figure 2.23: Example of the Hamiltonian cycle, in blue, in a TSP for n = 5 cities.

2.5.2.1 Instances

We use the **portgen** and **portcgen** generators of the **8th DIMACS** Implementation Challenge to create two types of random instances respectively: random uniform Euclidean, and random 10-cluster Euclidean instances. An example of a random Euclidean and a cluster Euclidean instance is displayed in Figure 2.24. We consider instances of size 100, 200 and 400 cities. We generate 100 training (tuning) and 10 test instances for each size. The optimal solution for all instances

is computed by the Concorde [1] TSP solver.

For the instance sets, or scenarios, in this thesis, we will consider two of them: one that contains all our Euclidian instances, S_u, and one that contains all our clustered instances, S_c.



(a) Random Uniform Euclidean (b) Cluster Euclidean

Figure 2.24: Example of 2 types of TSP instances

2.5.2.2 Neighborhood Operator

The $k - opt$ family of neighborhoods is widely used for the Traveling Salesman problem. Here we use the `2-opt`, `3-opt`, and `double-bridge` (a specific kind of `4-opt`) neighborhoods. Croes (1958) proposed the `2-opt` neighborhood. As shown in Figure 2.25, a pair of edges, (Π_i, Π_{i+1}) and (Π_j, Π_{j+1}), are removed. They are replaced by two new edges $((\Pi_i, \Pi_j)$ and (Π_{i+1}, Π_{j+1}) where $i \neq j - 1, j, j + 1 \; \forall i, j$, to create a feasible Hamiltonian cycle.

The `3-opt` follows a similar pattern, we delete three edges and create three new edges to bridge the gaps (Figure 2.26).

In many ILS implementations, the so-called `double-bridge` (DB) move is applied as a perturbation, which is a specific kind of `4-opt` neighborhood (Kernighan & Lin, 1970). In this case 4 edges are removed from the round trip. The 4 resulting sections are then rejoined to form a complete round trip (Figure 2.27).

[1] http://www.math.uwaterloo.ca/tsp/index.html

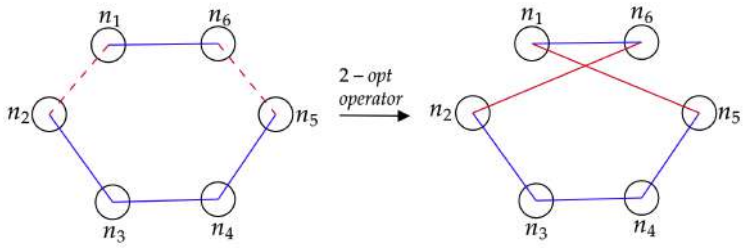


Figure 2.25: The example of 2-opt neighborhood operator. The current solution (n_1, n_2, n_3, n_4, n_5, n_6) used a 2-opt operator removing edges (n_1, n_2) and (n_5, n_6) that obtained a new neighbor (n_1, n_5, n_4, n_3, n_2, n_6)

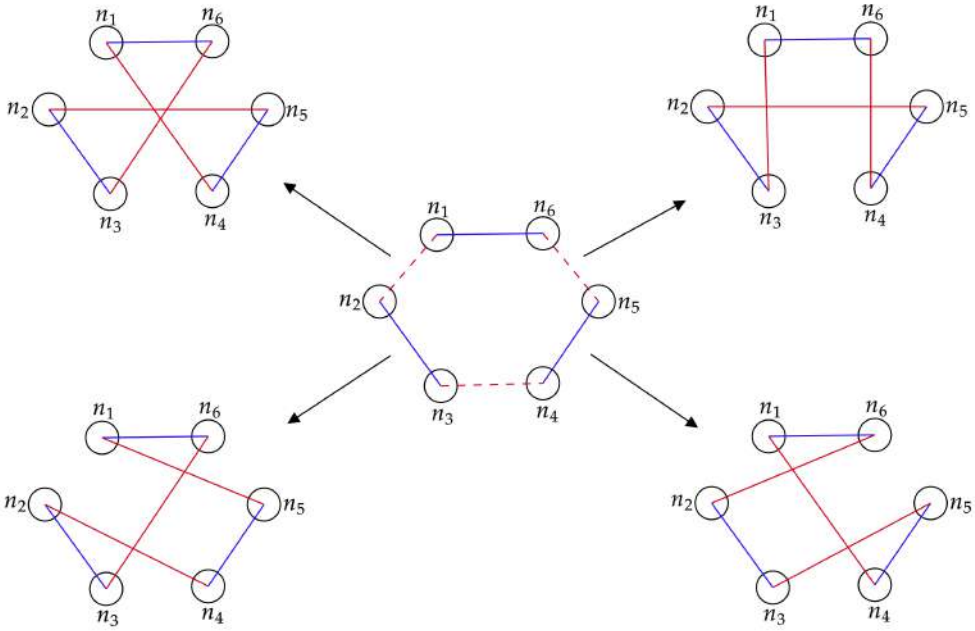


Figure 2.26: Example of 3-opt neighborhood operator. The neighbors of the solution (n_1, n_2, n_3, n_4, n_5, n_6) are (n_1, n_4, n_5, n_2, n_3, n_6), (n_1, n_5, n_4, n_2, n_3, n_6), (n_1, n_3, n_2, n_5, n_4, n_6), and (n_1, n_6, n_2, n_3, n_5, n_4)

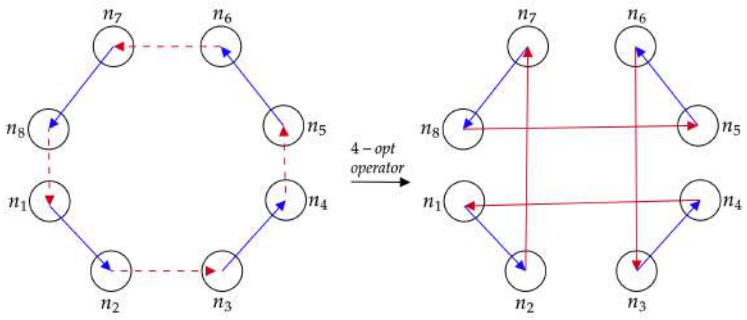


Figure 2.27: Example of Double-Bridge operator. The neighbors of the solution (n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8) used a 4-opt operator removing edges (n_1, n_8), (n_2, n_3), (n_4, n_5), and (n_6, n_7) which is then traversed in the orientation (n_1, n_2, n_7, n_8, n_5, n_6, n_3, n_4)

3 | Baseline ILS Algorithms

Contents

3.1 Introduction

Difficult optimisation problems are often solved by metaheuristics and hyper-heuristics. These frequently have many parameters or strategic components that alter their behaviour. There exists a set of parameter values / strategic components that corresponds to the best configuration for each instance of a problem. The problem of finding the best configuration to achieve the best performance can be approached through parameter tuning and parameter control. Parameter tuning of an algorithm involves optimizing the parameter values and the choice of the strategic components before running the algorithm, while parameter control is about adjusting the parameter values and modifying the strategic components during the execution. Since parameter tuning is performed before the final execution of the algorithm, it allows the evaluation of a large number of configurations. On the other hand, parameter control generally searches into a very few number of configurations, mainly 2 or 3 possible values of parameters of strategic components. In this context, we designed automatic multi-configuration algorithms, presented in next chapters, that modify the configuration during the run where the configurations used have been chosen by tuning in an offline process.

In this chapter we will present our baseline ILS algorithms that will be used to evaluate the performance of our contribution. Section 3.2 presents the framework of the R-ILS (see Chapter 2). It is the baseline for the tuning process. Section 3.3 presents the random multi-configuration ILS where a new configuration is selected after each restart of R-ILS. It is the baseline for the control process. Section 3.4 presents the parameter values and the strategic components used in the manuscript and we give the experimental protocol to evaluate the performance of our two baseline algorithms. Section 3.5 shows the results of our baseline algorithms on two permutation problems, namely the permutation flowshop scheduling problem

and the traveling salesman problem (see Section 2.5).

3.2 Restart-ILS

R-ILS embeds a basic hill-climbing where different exploration strategies can be chosen (see Chapter 2). The perturbation phase of R-ILS aims to escape from local optima. And its restart phase aims to diversify the search and to jump to new region of the search space. R-ILS is a classical framework of local search and so, is our baseline where only one configuration is performed during the whole run (Figure 3.1). For this purpose, a small configuration space will be used for the experiments.

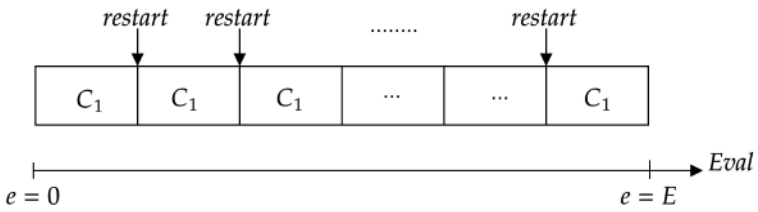


Figure 3.1: Single-configuration Model

3.3 Random Multi-Configuration ILS

The random multi-configuration ILS (R-MC-ILS) is a Restart-ILS where a configuration is randomly chosen among all possible ones after each restart. The diversification is doubled since the restart mechanism naturally leads to other regions of the search space and the search space is visited differently in each region. In effect, this creates c_i potential configurations during the execution of the algorithm as illustrated in Figure 3.2.

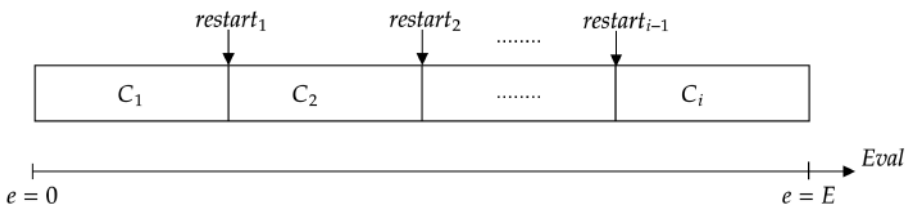


Figure 3.2: Random Multi-configuration Model

3.4 Experimental Protocol

In this section, we experiment the R-ILS and the R-MC-ILS.

3.4.1 Configuration Space

Tables 3.1 and 3.2 report all the strategic components/parameters and their respective values presented in Chapter 2 for PFSP and TSP respectively. It leads to a configuration space of 1200 (2×2×4×3×(1+4)×5) different configurations of PFSP and 3200 (4×2×4×4×(1+4)×5) different configurations of TSP.

With 1200 and 3200 configurations of PFSP and TSP respectively, the size of the space is already too large to exhaustively test all of them within a reasonable time. Therefore, we propose to fix $d, b = 4$ and $k = 3$, been the best parameter values found in the literature respectively, and to fix $stg = 100$, being the middle value in that parameter domain. It drastically reduces the size of the configuration space to 32 and 64 of PFSP and TSP respectively, and enables to perform the exhaustive analysis.

3.4.2 Protocol

The termination criterion of the algorithm is defined as a maximum number of evaluations. We prefer using a maximum number of evaluations instead of a max-

Table 3.1: Configuration Space. Numerical values with a start (*) have been selected for the exhaustive analysis for PFSP

Component/Parameter	**Value**
Neighborhood Operator	{shift, swap}
Neighborhood Order	{random, order}
Exploration Strategy	{IHCfirst, IHCbest, IHCworst, IHClahc}
Perturbation algo	{IG-D/R}
Perturbation strength d	{2, 3, 4*}
Diversification algo	{init, kick}
Diversification strength k	{3*, 4, 5, 10}
Stagnation criterion stg	{0, 50, 100*, 150 , 200}

Table 3.2: Configuration Space. Numerical values with a start (*) have been selected for the exhaustive analysis for TSP

Component/Parameter	**Value**
Neighborhood Operator	{shift, swap, 2-opt, 3-opt}
Neighborhood Order	{random, order}
Exploration Strategy	{IHCfirst, IHCbest, IHCworst, IHClahc}
Perturbation algo	{DB}
Perturbation strength b	{1, 2, 3, 4*}
Diversification algo	{init, kick}
Diversification strength k	{3*, 4, 5, 10}
Stagnation criterion stg	{0, 50, 100*, 150 , 200}

imum run-time because counting the number of evaluations is independent from the source code optimization, and then, the performance assessment is fairer between the algorithms. It is also independent of the machines used and of the load of the machines. This is especially important given that we run the experiments on a cluster with multiple concurrent users and CPUs that have TurboBoost (their frequency changes according to their load). Tables 3.3 and 3.4 report the maximum number of evaluations set for each instance size. This number has been experimentally determined in order to encounter several stagnation periods and therefore trigger several restarts. As metaheuristics are stochastic algorithms, we

perform 30 independent runs for each instance for each algorithm model.

Table 3.3: Maximum number of evaluations per instance size for PFSP.

Size	Evaluation number
50×20	40×10^6
100×10	60×10^6
100×20	100×10^6
200×10	200×10^6
200×20	400×10^6

Table 3.4: Maximum number of evaluations per cities size for both instances of TSP.

Size	Evaluation number
100	50×10^6
200	250×10^6
400	800×10^6

In order to compare the models on different instances, we propose to compute the relative percentage deviation (RPD) as follows:

$$\texttt{RPD} = \frac{C_{max}(\pi) - C_{max}(\pi^*)}{C_{max}(\pi^*)} \times 100 \tag{3.1}$$

where $C_{max}(\pi^*)$ is the best-known values. Then, the Friedman test is used to test the statistical equality of each model considering all instances or each size separately.

The models are implemented using the MH-Builder, a C++ metaheuristic optimization framework currently under development in our research team. The experiments were executed on 80 nodes of four 20-core 2.20GHz Intel Xeon Processor (Skylake, IBRS) with 16MB L3 cache and 500GB RAM, running Ubuntu 18.04.4 LTS.

3.5 Experimental Results

Let us remember that we perform two sets of experiments: first, we conduct an exhaustive analysis of a small configuration space for the R-ILS and second, we statistically compare the performance of R-ILS and R-MC-ILS, the baseline algorithm that modifies the configuration on the fly at each restart.

3.5.1 Results on PFSP

Table 3.5 reports, for each size of instance, the best configurations of R-ILS where the strength of the perturbation and the strength of the diversification have been set to 4 and 3 respectively and the stagnation criterion in 100 in order to enable an exhaustive exploration. The best configurations shown are the ones that are statistically better among the 32 available ones.

Not surprisingly, the `shift` operator is preferred to the `swap` operator to generate the neighbors of the current solutions and the `random` exploration of the neighborhood is chosen in each R-ILS. In these experiments, the hill climbing algorithms based on the `IHCfirst` strategy or the `IHClahc` strategy applied with a `kick` diversification lead to the best performance for all instances.

In the following, we will refer to these best configurations as exh_1 and exh_2 respectively. For instance with 10 machines, the `init` diversification with the late acceptance strategy are also equivalent to the latter. These results are those expected based on the literature. This shows that our protocol is well adjusted to assess the performance of our models to solve the Taillard instances of the PFSP.

In the following, we compare R-ILS with R-MC-ILS. We analyze the resulting

Table 3.5: Best configurations of the R-ILS on Taillard instances for PFSP.

Sizes	ILS	Diversification	stg
50×20	shift, random, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	kick(3)	100
100×10	shift, random, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	init	100
100×20	shift, random, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	kick(3)	100
200×10	shift, random, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	init	100
200×20	shift, random, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	kick(3)	100
All Instances	shift, random, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, random, IHClahc, IG-D/R(4)	kick(3)	100

algorithm configuration on the test set instances. Our finding is that R-ILS clearly outperforms R-MC-ILS for all scenarios and instance sizes of the problem (see also Tables 3.6-3.7).

3.5.2 Results on TSP

For the TSP, two scenarios are considered: the first one with random uniform Euclidean instances (S_u) and the second one with random 10-cluster Euclidean instances (S_c) as detailed in Chapter 2.5.2.

Table 3.8 gives the description of the exhaustive analysis of the R-ILS for the

Table 3.6: Statistical comparison of R-ILS and R-MC-ILS for each scenario on PFSP instances.

Scenario	**single-config.**		**online-multi.**
	exh_1	exh_2	rnd
S_{all}	+	+	−
$S_{N=100}$	+	+	−
$S_{N=200}$	+	+	−
$S_{M=20}$	+	+	−

Table 3.7: Statistical comparison of single-configuration models and online multi-configuration models for each size on PFSP instances.

Size	**single-config.**		**online-multi.**
	exh_1	exh_2	rnd
50×20	+	+	−
100×10	+	+	−
100×20	+	+	−
200×10	+	+	−
200×20	+	+	−

two scenarios (S_u, S_c) where the values of the perturbation and diversification strengths and the stagnation criterion have been fixed to 4, 3 and 100 respectively like for the experiments conducted on the FSP. The `IHCbest` exploration strategy was selected for 100 cities-instances associated with the `order` neighborhood and `2-opt` neighborhood operator, while for 200 cites-instances of sce-

nario S_c the `random` neighborhood order and `3-opt` neighborhood operator was preferred. In this experiment of the S_u scenario and S_c scenario, the two configurations (exh_1 and exh_2) share the three ILS-parameters namely the neighborhood operator (`3-opt`), the neighborhood order (`random`) , and the exploration strategy (`IHCFirst`) and both are associated with the two available diversification (`init` and `kick`).

Then, we compare R-ILS and R-MC-ILS. The results of Tables 3.9 and 3.10 show that R-ILS outperforms R-MC-ILS statistically.

3.6 Conclusion

In this section, we presented basic algorithms namely R-ILS and R-MC-ILS that will be used as baselines for the next experiments . We reduced the configuration space of R-ILS to 32 and 64 for PFSP and TSP respectively. Therefore, we were enabled to perform an exhaustive analysis of the R-ILS configurations assessed on the whole test instance sets. Next, we implemented the R-MC-ILS which chooses a configuration on the fly among 1200 and 3200 configurations for PFSP and TSP respectively. Finally, we compared our baseline algorithms, R-ILS and R-MC-ILS, with each other, on PFSP and TSP.

For PFSP, two configurations exh_1 and exh_2 of R-ILS outperform the 32 other ones. Both configurations use the `shift` operator, the `random` neighborhood order and the `kick(3)` diversification. They differ by the exploration strategy being `IHCfirst` and `IHClahc` respectively. These two strategies are know in the literature to be efficient on the PFSP. For TSP, the R-ILS configurations exh_1 and exh_2 have been selected over the 64 available ones. Both configurations share the same ILS being the `3-opt` operator, the `random` neighborhood order, the `IHCfirst` exploration and the `DB(4)` perturbation. However, the two possible

Table 3.8: Best configurations of R-ILS for TSP.

Sizes	ILS	Diversification	stg
Random Uniform Euclidean Instances			
100	2-opt, order, IHCbest, DB(4)	kick(3)	100
	2-opt, order, IHCbest, DB(4)	init	100
	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
200	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
400	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
S_u	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
Random 10-Clusters Euclidean Instances			
100	2-opt, order, IHCbest, DB(4)	kick(3)	100
	2-opt, order, IHCbest, DB(4)	init	100
	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
200	3-opt, random, IHCbest, DB(4)	kick(3)	100
	3-opt, random, IHCbest, DB(4)	init	100
	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
400	3-opt, random, IHCbest, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100
S_c	3-opt, random, IHCfirst, DB(4)	kick(3)	100
	3-opt, random, IHCfirst, DB(4)	init	100

Table 3.9: Statistical comparison of R-ILS and R-MC-ILS for each scenario on TSP instances.

Scenario	**single-config.**		**online-multi.**
	exh_1	exh_2	rnd
Random Uniform Euclidean Instances			
S_c	+	+	−
Random 10-Cluster Euclidean Instances			
S_u	+	+	−

Table 3.10: Statistical comparison of R-ILS and R-MC-ILS for each size on TSP instances.

Size	**single-config.**		**online-multi.**
	exh_1	exh_2	rnd
Random Uniform Euclidean Instances			
100	+	+	−
200	+	+	−
400	+	+	−
Random 10-Cluster Euclidean Instances			
100	+	+	−
200	+	+	−
400	+	+	−

diversification strategies being `init` and `kick(3)` seem to have less impact on performance. For both PFSP and TSP, R-ILS outperforms R-MC-ILS.

4 | Sequential Multi-Configuration ILS

This chapter presents the work published in international peer-reviewed conferences and workshops:

- Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L.(2020).Time-dependent automatic parameter configuration of a local search algorithm. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* ECADA Workshop, p. 1898-1905.

- Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L. (2020, January). Configuration of Time-Dependent Local Search for the Travelling Salesman Problem. In *ORBEL - The Belgian Operational Research (OR) Society.*

- Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L. (2019, September). Automatic Configuration of a Dynamic Hill Climbing Algorithm. In *SLS - International Workshop on Stochastic Local Search Algorithms*.

Contents

4.1 Introduction

Automatic algorithm configuration (AAC), also known as parameter tuning, provides a single configuration of the target algorithm. On the other hand, adaptive algorithms evolve in the search space and modify on the fly their parameters/strategic components to locally adapt to the search space regions. In Chapter 3, we saw that the adaptation is difficult when a large number of configurations is possible. Our aim is to design algorithms able to locally adapt their configurations by choosing among a large number of parameters/strategic components. In this chapter, we propose a first approach where the configuration of a R-ILS is modified when a number of evaluations is reached. The configurations used during the execution of the R-ILS are tuned using AAC on a large space of configurations. Section 4.2 presents this approach called Sequential multi-configuration ILS since the ordering of the configurations is predefined. Section 4.3 gives the experimental protocol we followed to conduct the experiments on the PFSP and the TSP. Section 4.4 presents the experimental results.

4.2 Sequential Multi-configuration ILS

R-ILS is an iterated local search with a restart mechanism when stagnation (here a certain number of evaluations without improvement) is met. We want to design a R-ILS that modifies its parameter values and strategic components during the execution to better explore the search space. Contrary to adaptive algorithms where the choice of the next configuration is made on the fly, here, we propose to automatically configure a R-ILS to deal with a large configuration space. Our approach is based on the dynamic framework proposed by Pageau

et al. (2019) for multi-objective optimization. The Sequential Multi-configuration ILS (S-MC-ILS) is a framework based on R-ILS and is designed to be easily tuned with AAC configurator. Indeed, considering a maximal number of evaluations E and a number of successive configurations K, we instantiate K successive R-ILS. For each one, the budget e'_k (number of evaluations) has to be defined. Figure 4.1 gives an illustration of our framework. In addition to the configuration space of the original R-ILS, the S-MC-ILS has its own parameters to define being the percentages E_k (with $k \in \{1, ..., k\}$) to split the total number of evaluations into the K configurations. In the following, we consider only 5 possible percentages being $\{10, 25, 50, 75, 90\}$. Then, each budget is calculated as follows: $e_1 = (E \times E_1)/100$; $e_2 = ((E - e_1) \times E_2)/100$; more generally, for $k < K$, $e_k = ((E - \sum_{i=1}^{k-1} e_i) \times E_k)/100$. The final split, $e_K = E - \sum_{i=1}^{K-1} e_i$, uses up the remaining budget. For K splits, we therefore have $\{e_1, e_2, \ldots, e_K\}$ associated evaluation budgets. For example, if $K = 3$, three different splits $\{e_1\}$, $\{e_1, e_2\}$ and $\{e_1, e_2, e_3\}$ respectively (see Figure 4.1). Note that $\{e_1\}$ corresponds to the original R-ILS. For example, if the maximal number of evaluations is 1000 and the configurator sets parameters $E_1 = 25$ % and $E_2 = 50$ % then the budget for each of the the 3 configurations will be $e_1 = 250$, $e_2 = 375$ and $e_3 = 375$ evaluations.

4.3 Experimental Protocol

In this section, we describe the choices that lead to the configuration space used in the experiments for the sequential multi-configuration model to optimise the configuration of the target algorithm on given instances of PFSP and TSP.

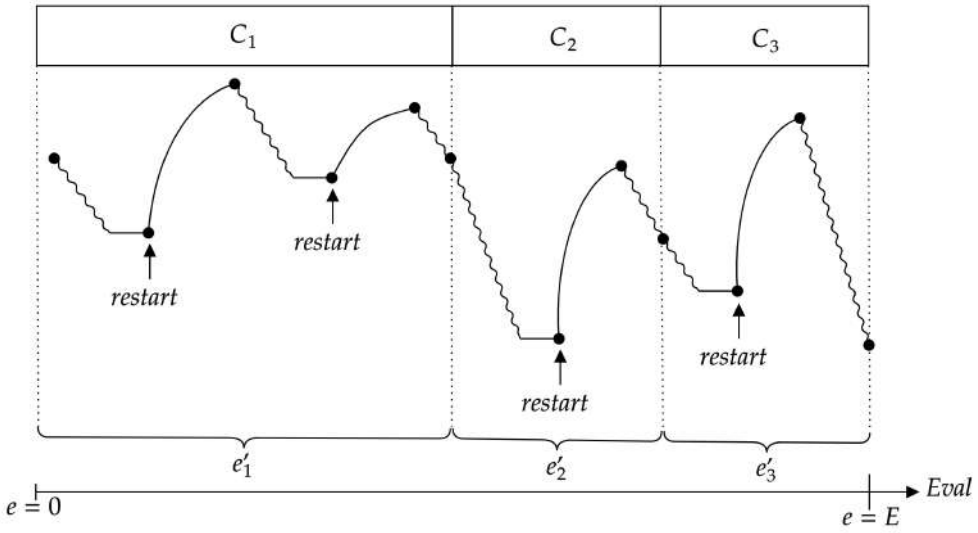


Figure 4.1: Sequential Multi-configuration ILS

4.3.1 Configuration Space for PFSP

For a single time split (K=1), we have 1200 ($2 \times 2 \times 4 \times 3 \times (4 + 1) \times 5$) potential configurations (see Table 3.1) of our algorithm per evaluation split. For a single evaluation split ($K = 1$) therefore we have the same 1200 configurations and the multi-configuration nature of the framework is not exploited. For $K = 2$, or 2 splits, then the fraction of the first evaluation split will be chosen among the 5 different possibilities already presented (|s| = |{10,25,50,75,90}| = 5) and the second and final time split will take the remaining budget. This amounts to approximately $7.20 \times 10^{+6}$ different configurations ($1200+5\times1200^2$). When $K = 3$, or 3 evaluation splits, then we have a total of approximately $4.32 \times 10^{+10}$ different configurations ($1200 + 5 \times 1200^2 + 5^2 \times 1200^3$).

4.3.2 Configuration Space for TSP

For a single evaluation split ($K = 1$), we have 3200 ($4\times2\times4\times4\times(4+1)\times5$) potential configurations (see Table 3.2). For $K = 2$ and $K = 3$, we obtain approximately

$5.12 \times 10^{+7}$ and $8.19 \times 10^{+11}$ different configurations, respectively.

4.4 Experimental Results

In this section, we experiment S-MC-ILS on PFSP and TSP. First, we use the irace configurator with a limited tuning budget of 5000 runs to identify the best configurations of S-MC-ILS for each problem. The comparison with our baseline algorithms presented in Chapter 3 will be presented in Chapter 5.

4.4.1 Results on PFSP

In order to tune S-MC-ILS, we have generated our own training instances. Then, the performance of the best configurations selected by irace is compared on Taillard instance. Table 4.1 reports for the four scenarios S_{all}, $S_{N=100}$, $S_{N=200}$ and $S_{M=20}$ the best configurations returned by irace (seq_i) . These S-MC-ILS have three splits, meaning that three different ILS should be performed successively during the run. For all scenarios, the best configurations seq_i contain the best configurations of R-ILS exh_1 and exh_2 identified in Chapter 3. The numerical values were not available for the exhaustive exploration which were maybe increasing the possible values would have produced some different results. The `IHCbest` strategy is bigger improvements that are found in whole configurations, while `IHCworst` strategy was found to be useful in one of each scenario. This is in keeping with the literature (Tari et al., 2018) that shows that this unlikely operator can be useful in specific situations. Surprising, $S_{N=100}$ in ILS_1 are started with one of the best exhaustive configuration of R-ILS. The parameters of the whole phases are more varied, with the `swap` neighborhood operator, considered quite inefficient by PFSP specialists, being used here. In an overall picture, only the `init` diversification is used to diversify the search for each restart (completed restart)

in seq_2 of $S_{N=200}$ in ILS_3 while the other scenarios used the `kick` diversification, it seems to be better to jump to closer locations of the search space rather than starting a new. The stagnation criteria are at least set to 50.

4.4.2 Results on TSP

irace was performed to determine the best configurations for the TSP (Table 3.2. In brief, Table 4.1 gives, for each scenario (S_u, S_c), the sequential multi-configuration returned over configurations space maximum 10^{+11} for two scenarios. The whole scenario is composed of the best exhaustive configurations exh_1 and exh_2 of R-ILS previously found in Chapter 3.

For the S_u scenario three configurations ($seq_1^{S_u}$, $seq_2^{S_u}$ and $seq_3^{S_u}$) of S-MC-ILS were returned by irace and each configuration use a different maximum split. In addition, it seems that three splits give better results since more configurations have been tested. The `IHCbest` strategy is selected in the whole configuration and selected with a variety of neighborhood operators. On the other hand, the `IHCworst` is selected only one for $seq_3^{S_u}$ in ILS_3 and selected chosen with `order` neighborhood. Moreover, the whole configuration in ILS_3 uses only `kick` diversification, while the other configurations have used the combination of two available diversification components (`kick`, `init`). `init` jumps to a new location in the search space (full restart). The ILS_2 of all configurations is set to only 50 of the stagnation criterion while other configurations are at least set to 100 stagnation.

For S_c scenario, the best four ($seq_1^{S_c}$, $seq_2^{S_c}$, $seq_3^{S_c}$, and $seq_4^{S_c}$) configurations of sequential multi-configurations were returned by irace. It seems that three splits give better results since more configurations have been tested. The parameters of the third phases is more varied with the `swap` operator for $seq_1^{S_c}$ and $seq_2^{S_c}$ in

ILS_3. The neighbors generated with the `swap` neighborhood operator are different from the ones generated with the `shift`. This choice enables new connections between solutions it changes the search landscape. The first, best configuration in ILS_1 used `kick` diversification, while other configurations in ILS_1 used the `init` diversification.

Table 4.1: Best configurations of the S-MC-ILS for PFSP returned by irace.

Size	Conf.	K_1			K_2			K_3			e_1	e_2
		ILS_1	Diversification	stg	ILS_2	Diversification	stg	ILS_3	Diversification	stg		
S_{all}	$seq_1^{S_1}$	shift, random, IHCfirst, IG-D/R(4)	kick(3)	200	shift, random, IHClahc, IG-D/R(2)	kick(3)	50	swap, order, IHCfirst, IG-D/R(2)	kick(4)	100	75	75
	$seq_2^{S_1}$	shift, random, IHCfirst, IG-D/R(4)	kick(5)	150	shift, order, IHCbest, IG-D/R(3)	kick(3)	200	shift, order, IHCworst, IG-D/R(2)	kick(4)	100	90	25
	$seq_3^{S_1}$	shift, order, IHCbest, IG-D/R(4)	kick(3)	200	shift, random, IHCfirst, IG-D/R(3)	kick(4)	150	shift, random, IHClahc, IG-D/R(4)	kick(4)	150	90	25
$S_{N=100}$	$seq_1^{S_2}$	shift, random, IHCfirst, IG-D/R(3)	kick(3)	200	swap, random, IHCworst, IG-D/R(4)	kick(5)	150	swap, order, IHCbest, IG-D/R(4)	kick(4)	50	90	10
	$seq_2^{S_2}$	shift, random, IHCfirst, IG-D/R(3)	kick(10)	150	shift, order, IHClahc, IG-D/R(4)	kick(3)	50	swap, random, IHCbest, IG-D/R(3)	kick(5)	150	90	25
	$seq_3^{S_2}$	shift, random, IHCfirst, IG-D/R(4)	kick(5)	200	swap, order, IHCbest, IG-D/R(4)	kick(3)	100	shift, random, IHClahc, IG-D/R(3)	kick(5)	150	75	25
$S_{N=200}$	$seq_1^{S_3}$	shift, random, IHCfirst, IG-D/R(4)	kick(10)	100	swap, order, IHCfirst, IG-D/R(4)	kick(3)	100	shift, order, IHClahc, IG-D/R(4)	kick(10)	50	90	10
	$seq_2^{S_3}$	shift, random, IHClahc, IG-D/R(4)	init	200	shift, random, IHCfirst, IG-D/R(3)	init	200	swap, order, IHCbest, IG-D/R(3)	init	100	10	75
	$seq_3^{S_3}$	shift, random, IHCfirst, IG-D/R(4)	kick(4)	100	swap, order, IHCbest, IG-D/R(4)	kick(10)	100	shift, random, IHCworst, IG-D/R(4)	kick(4)	150	90	75
	$seq_4^{S_3}$	swap, random, IHClahc, IG-D/R(4)	kick(10)	200	shift, random, IHCfirst, IG-D/R(3)	kick(3)	200	shift, order, IHCbest, IG-D/R(4)	kick(4)	150	25	75
$S_{M=20}$	$seq_1^{S_4}$	shift, random, IHCfirst, IG-D/R(3)	kick(3)	200	shift, order, IHCbest, IG-D/R(4)	kick(3)	150	swap, order, IHClahc, IG-D/R(4)	kick(4)	50	75	75
	$seq_2^{S_4}$	shift, random, IHCfirst, IG-D/R(3)	kick(4)	150	shift, order, IHCworst, IG-D/R(4)	kick(10)	100	swap, order, IHCbest, IG-D/R(2)	kick(10)	200	90	75
	$seq_3^{S_4}$	shift, random, IHClahc, IG-D/R(3)	kick(4)	200	shift, order, IHCfirst, IG-D/R(4)	kick(10)	200	swap, random, IHCbest, IG-D/R(3)	kick(10)	50	90	75

Table 4.2: Best configurations of the S-MC-ILS for TSP returned by irace.

Size	Conf.	K_1			K_2			K_3			e_1	e_2
		ILS_1	Diversification	stg	ILS_2	Diversification	stg	ILS_3	Diversification	stg		
S_u	$seq_1^{S_u}$	3-opt, random, IHCfirst, DB(2)	kick(3)	150	shift, random, IHCbest, DB(2)	init	50	3-opt, random, IHCbest, DB(4)	kick(4)	100	90	75
	$seq_2^{S_u}$	3-opt, order, IHCbest, DB(1)	init	200	3-opt, random, IHCfirst, DB(3)	kick(3)	50	2-opt, order, IHCbest, DB(2)	kick(10)	200	75	10
	$seq_3^{S_u}$	3-opt, random, IHCfirst, DB(3)	init	200	3-opt, order, IHCbest, DB(3)	init	50	shift, order, IHCworst, DB(1)	kick(4)	200	75	10
S_c	$seq_1^{S_c}$	3-opt, order, IHCworst, DB(2)	kick(3)	200	3-opt, random, IHCfirst, DB(1)	kick(3)	200	swap, random, IHCbest, DB(1)	kick(10)	200	75	75
	$seq_2^{S_c}$	3-opt, random, IHCfirst, DB(1)	init	200	3-opt, order, IHCbest, DB(1)	kick(10)	100	swap, random, IHClahc, DB(3)	init	150	50	75
	$seq_3^{S_c}$	3-opt, random, IHCfirst, DB(2)	init	50	shift, random, IHCbest, DB(1)	init	50	3-opt, random, IHCfirst, DB(4)	kick(4)	150	75	10
	$seq_4^{S_c}$	3-opt, random, IHCbest, DB(2)	init	200	3-opt, random, IHCfirst, DB(1)	kick(10)	50	3-opt, order, IHCfirst, DB(1)	init	50	75	90

4.5 Conclusion

In this Chapter, we presented the Sequential Multi-configuration ILS that modifies successively the configuration when a number of predefined number of evaluations is reached. The configurations are also tuned before the run using AAC, and more precisely the irace configurator. The S-MC-ILS framework is a first approach to benefit from both parameter tuning and parameter control. Indeed, the AAC of S-MC-ILS enables to consider a large search space of parameters that parameter control approach cannot. In the experiments, we allowed the use of up to three configurations in sequence to solve instances of PFSP or TSP. We showed that the best S-MC-ILS use 3 successive configurations. However, the main drawback of this approach is the fixed number of modification during the run which depends on the number of evaluations. In the next chapter, we present a more flexible approach based on probabilistic models.

5 | Probabilistic Multi-Configuration ILS

This chapter presents the work published in HIS, an international peer-reviewed conference:

- Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L.(2021, December). Automatic Algorithm Multi-Configuration Applied to an Optimization Algorithm. In *21st International Conference on Hybrid Intelligent Systems (HIS 2021)*, p. 160-170.

Contents

5.1 Introduction

In this chapter, we present an other approach to use both parameter tuning and parameter control to improve the performance of the R-ILS. Contrary to the Sequential Multi-Configuration ILS presented in Chapter 4 where the configuration is modified when a number of evaluations is reached, this new approach is able to modify the configuration of the R-ILS after each restart when stagnation is met. Then we propose two models: the first one leads to a fixed order decision while the second one uses roulette selection to choose the next configuration to apply.

Section 5.2 presents this approach called Probabilistic Multi-Configuration ILS and the fixed and the roulette models designed to select the next configuration after each restart. Section 5.3 gives the experimental protocol we followed to conduct the experiments on the PFSP and the TSP. Section 5.4 presents the experimental results of the two models of the Probabilistic Multi-Configuration ILS. Section 5.5 reports and analyzes the results of the comparison between our two Multi-Configuration ILS, namely Sequential and Probabilistic, and the baseline algorithms, namely R-ILS and R-MC-ILS (see Chapter 3).

5.2 Probabilistic Multi-configuration ILS

In this chapter, we consider scenarios where offline tuning potentially determines which algorithm configuration to apply initially, but also after a restart of some algorithm. Here, we instantiate this multi-configuration model on an Iterated Local Search as Probabilistic Multi-configuration ILS (P-MC-ILS) whose components are detailed in Chapter 2. A restart provides a fairly intuitive point at which it makes sense to potentially switch configuration. The type of restart operator (including if it is a full or partial restart) is a parameter in itself, as is the condition

that triggers the restart.

When the ILS algorithm is stagnant, it continues to run and therefore loses time in the search process as the current solution cost does not improve, i.e., the search stagnates. The deciding criterion for a restart is often a number of stagnation steps. In this thesis, we opt for the simplest option of letting the configurator select a maximum number of stagnation steps among a discrete set of values that triggers a restart.

Aside from the parameters related to the restart mechanism, the remaining ones are for the ILS and its building blocks, as well as the initialization mechanism. These are changed according to two types of scenarios that will be detailed further: fixed multi-configuration and roulette multi-configuration.

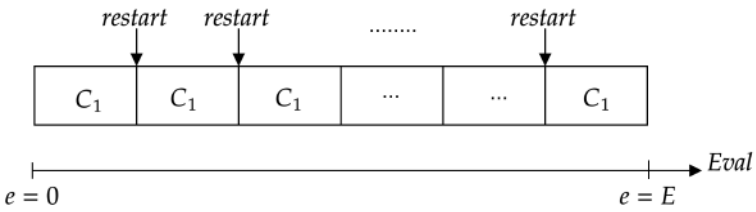
While automatic algorithm configuration is very powerful, increasing the number of parameters requires additional tuning time. For this reason, as well as for the sake of simplicity, we restrict ourselves to a maximum of three different configurations of P-MC-ILS that can be returned by the configurator.

The P-MC-ILS consists of two model: (1) fixed model iterates tuned ILS in a predefined order and (2) the roulette model selects, after each restart, one of the tuned ILS following a predefined probability.

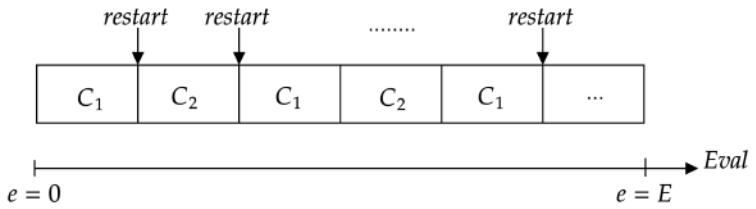
5.2.1 Fixed Model

In fixed multi-configuration the configurations are applied in a predefined sequence that loops around until the total execution budget is used up as illustrated in Figure 5.1. There can be a sequence of two or three configurations. The trivial single-configuration variant is also there to serve as a baseline and the same configuration (c_1) is used after each restart until the stopping criterion is reached (Figure 5.1(a)). In the two- and three-configuration versions, we start with some initial

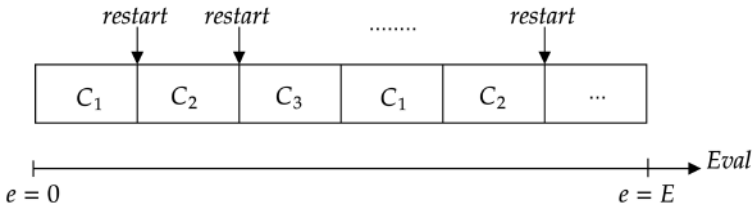
configuration (c_1), then switch to c_2 after the restart. These two configurations are then used sequentially until the stopping criterion is met for the two-configuration version (Figure 5.1(b)). While the three-configuration version naturally employs a third configuration (Figure 5.1(c)).



(a) Single-configuration model



(b) Two-configuration model



(c) Three-configuration model

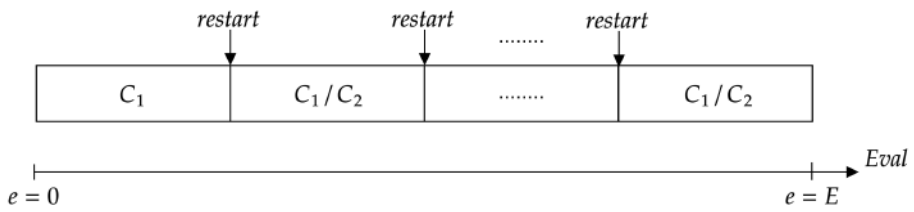
Figure 5.1: Fixed multi-configuration Model

5.2.2 Roulette Model

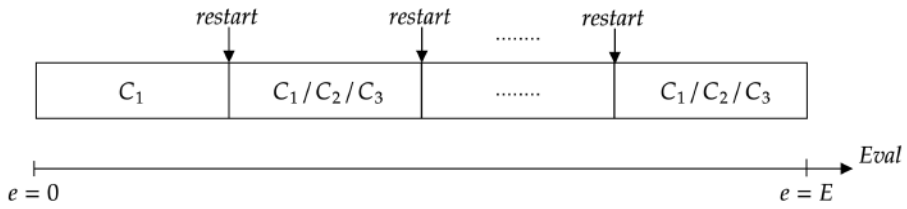
For roulette multi-configuration ILS, we consider two and three configurations with roulette wheel selection, and dispense from using a single trivial configuration as this is equivalent to the fixed single-configuration.

In this setting, the probability of applying each configuration is a parameter op-

timized by the configurator. The application of each configuration is illustrated in Figure 5.2. We always start with configuration (c_1). Then, after each restart, each configuration can be chosen with some fixed probability. Configuration c_1 is assigned some percentage value p_1 that will trivially correspond to probability P_1. In the two-configuration case, c_2 then automatically has probability $P_2 = 1 - P_1$. In the three-configuration case, c_2 is assigned some percentage value p_2 of the remaining $1 - P_1$, which gives a probability $P_2 = p_2 \times (1 - P_1)$ for c_2. Finally c_3 automatically has probability $P_3 = 1 - (P_1 + P_2)$. As an example, if 50 % is chosen for both c_1 and c_2, then the probability of c_1 will be 0.5, c_2 will be 0.25 (50 % of the remaining 0.5) and c_3 will be 0.25 $(1 - (0.5 + 0.25))$.



(a) Two-configuration model



(b) Three-configuration model

Figure 5.2: Roulette multi-configuration Model

5.3 Experimental Protocol

Tables 3.1 and 3.2 report all the search components/parameters and their respective values presented in the sections above for the PFSP and TSP respectively. It leads to a configuration space of 1200 different configurations (2×2×4×3×(1+4)×5)

of PFSP and 3200 different configurations (4×2×4×4×(1+4)×5) of TSP. This configuration space is used by the single-configuration model of fixed P-MC-ILS.

5.3.1 Configuration Space for PFSP

The other model of fixed P-MC-ILS, two configurations sequential, we have around 2.89×10^5 different configurations ($240 \times 5 + 240^2 \times 5$) and three configurations sequential, then we have around 6.94×10^7 different configurations ($240 \times 5 + 240^2 \times 5 + 240^3 \times 5$).

The roulette P-MC-ILS uses a probability P to select the next configuration to run. We set $P = \{25, 50, 75\}$ and allow only three values to avoid a combinatorial explosion of the configuration space. In the two-configurations case we have around 8.64×10^5 different configurations ($240^2 \times 5 \times 3$) and three-configurations case we have around $6.23 \times 10^{+8}$ different configurations ($240^2 \times 5 \times 3 + 240^3 \times 5 \times 3^2$).

5.3.2 Configuration Space for TSP

For two and three configurations case of our fixed multi-configuration model, we have approximately 2.05×10^6 different configurations ($640 \times 5 + 640^2 \times 5$) and 1.31×10^9 different configurations ($640 \times 5 + 640^2 \times 5 + 640^3 \times 5$) respectively.

We also have 6.14×10^6 different configurations ($640^2 \times 5 \times 3$) and 1.18×10^{10} different configurations ($640^2 \times 5 \times 3 + 640^3 \times 5 \times 3^2$) for two and three configurations case respectively of our roulette P-MC-ILS.

5.4 Experimental Results

In this section, we focus on the fixed and roulette P-MC-ILS. First, we present the best configurations returned by irace among configuration spaces larger than

10^7. Then, the performance of the best configurations is compared to problem instances of PFSP and TSP. A configuration is then composed of a maximum of three differently tuned ILS.

5.4.1 Results on PFSP

Result of Fixed Model. Table 5.1 reports the three (S_{all}, $S_{N=200}$, $S_{M=20}$) and four ($S_{N=100}$) best configurations returned by irace for the fixed P-MC-ILS. The first P-MC-ILS (fix_1) of $S_{N=200}$ is composed of three tuned ILS only, while the other first P-MC-ILS are composed of two tuned ILS. However, the last two P-MC-ILS (fix_2, fix_3) in $\{S_{all}, S_{N=200}, S_{M=20}\}$ and three P-MC-ILS (fix_2, fix_3, fix_4) in $\{S_{N=100}\}$ are composed of three tuned ILS. All scenarios contain the best single-configuration presented in the previous section, except for the strength of the perturbation algorithms and the stagnation criterion. We recall that all the numerical values were not available for the exhaustive exploration, and maybe increasing the possible values would have produced some different results. However, the fixed model configuration with only one ILS was in the configuration space of irace and no such configuration has been returned. This shows that having at least two ILS seems to lead to better performance. Surprisingly, the $S_{N=200}$ used uses the `swap` operator in whole tuned ILS (fix_2 in ILS_1, fix_3 in ILS_2, and fix_1, fix_3 in ILS_3), while the other scenarios are used in only ILS_3 (fix_2, fix_3 of S_{all}, fix_3, fix_4 of $S_{N=100}$, and fix_3 of $S_{M=20}$). The neighbors generated with the `swap` operators are different from the ones generated with the `shift` neighborhood operator. This choice enables new connections between solutions as it changes the search landscape. The `IHCfirst` strategy is selected in whole scenarios with the ordered exploration of the neighborhood. This is logical since this strategy evaluates all the neighbors of the current solution to select the best among them. Moreover, only the `kick` diversification is used to diversify the search for each restart (a

partial restart) and the stagnation criteria is at least set to 100.

Table 5.1: Best configurations of the fixed P-MC-ILS for PFSP returned by irace.

Size	**Conf.**	ILS_1	ILS_2	ILS_3	Diversification	stg
S_{all}	$fix_1^{S_1}$	shift, random, IHCfirst, IG-D/R(4)	shift, random, IHClahc, IG-D/R(3)	–	kick(3)	200
	$fix_2^{S_1}$	shift, random, IHCfirst, IG-D/R(3)	shift, random, IHClahc, IG-D/R(4)	swap, order, IHCbest, IG-D/R(3)	kick(3)	200
	$fix_3^{S_1}$	shift, random, IHClahc, IG-D/R(3)	shift, random, IHCfirst, IG-D/R(4)	swap, random, IHClahc, IG-D/R(4)	kick(5)	100
$S_{N=100}$	$fix_1^{S_2}$	shift, random, IHClahc, IG-D/R(3)	shift, random, IHCfirst, IG-D/R(4)	–	kick(5)	200
	$fix_2^{S_2}$	shift, random, IHCfirst, IG-D/R(4)	shift, random, IHClahc, IG-D/R(3)	shift, random, IHCbest, IG-D/R(2)	kick(3)	200
	$fix_3^{S_2}$	shift, random, IHCfirst, IG-D/R(3)	shift, random, IHClahc, IG-D/R(2)	swap, random, IHClahc, IG-D/R(4)	kick(3)	200
	$fix_4^{S_2}$	shift, random, IHClahc, IG-D/R(4)	shift, random, IHCfirst, IG-D/R(3)	swap, order, IHCbest, IG-D/R(2)	kick(4)	150
$S_{N=200}$	$fix_1^{S_3}$	shift, random, IHClahc, IG-D/R(4)	shift, random, IHCfirst, IG-D/R(2)	swap, order, IHClahc, IG-D/R(2)	kick(4)	200
	$fix_2^{S_3}$	swap, random, IHClahc, IG-D/R(2)	shift, random, IHCfirst, IG-D/R(4)	shift, order, IHCbest, IG-D/R(3)	kick(5)	150
	$fix_3^{S_3}$	shift, random, IHClahc, IG-D/R(4)	swap, random, IHCworst, IG-D/R(3)	swap, random, IHCfirst, IG-D/R(3)	kick(5)	150
$S_{M=20}$	$fix_1^{S_4}$	shift, random, IHCfirst, IG-D/R(3)	shift, random, IHCbest, IG-D/R(4)	–	kick(5)	200
	$fix_2^{S_4}$	shift, random, IHCfirst, IG-D/R(4)	shift, random, IHClahc, IG-D/R(4)	shift, random, IHCbest, IG-D/R(3)	kick(3)	200
	$fix_3^{S_4}$	shift, random, IHCfirst, IG-D/R(4)	shift, random, IHCworst, IG-D/R(2)	swap, random, IHClahc, IG-D/R(4)	kick(5)	150

Result of Roulette Model. Table 5.2 reports the three ($S_{M=20}$) or four (S_{all}, $S_{N=100}$, $S_{N=200}$) best configurations (rlt_1, rlt_2, rlt_3 and rlt_4), returned by in the irace for the roulette P-MC-ILS with a maximum of three different ILS in $\{S_{all}$, $S_{M=20}\}$. However, the rlt_3 and rlt_4 of $\{S_{N=100}, S_{N=200}\}$ are composed the three tuned ILS, while first and second P-MC-ILS are composed two tuned ILS. Even if they are all composed in part of the best single-ILS has seen in the previous section, the `IHCbest` and the `IHCworst` strategies are more represented. In deeps, the `IHCbest` strategy is set three tuned ILS (rlt_1, rlt_4 of S_{all} in ILS_2, rlt_2 of $S_{N=100}$ in ILS_2, rlt_2 of $S_{N=200}$ in ILS_1, and rlt_3 of $S_{M=20}$ in ILS_3), while the `IHCworst` is preferred for ILS_3 (rlt_3, rlt_4 of S_{all}, rlt_3 of $S_{N=100}$, and rlt_1 of $S_{M=20}$) and ILS_2 (rlt_4 of $S_{N=200}$). Contrary to the fixed model, the `IHCworst` is set in only ILS_2. Moreover, the `shift` neighborhood operator is always selected in ILS_1 except rlt_2 of $S_{N=200}$. For the roulette model, the stagnation criterion is at least set to 150. This implies that the configurations leave time to the perturbation `IG-D/R` to find a better adjacent region before restarting. In this multi-configuration model, the probability of choosing among the three ($S_{M=20}$) or four (S_{all}, $S_{N=100}$, $S_{N=200}$) ILS is also tuned automatically. The four best configurations of S_{all} give 75% of chance to select ILS_1, while the other scenarios give the variety of probability $p_1 = \{25, 50, 75\}$. Moreover, the three best configurations of $S_{M=20}$ give 75% of chance to select ILS_2 that section 5.2.2 is assigned some percentage value p_2. It seems that ILS_2 and ILS_3 aim at introducing diversity in the search as well in changing the exploration strategy as modifying the definition of the neighborhood.

Table 5.2: Best configurations of the roulette P-MC-ILS for PFSP returned by irace.

Size	**Conf.**	ILS_1	ILS_2	ILS_3	Diversification	stg	P_1	P_2
S_{all}	$rlt_1^{S_1}$	shift, random, IHClahc, IG-D/R(3)	shift, random, IHCbest, IG-D/R(4)	shift, random, IHCfirst, IG-D/R(4)	kick(5)	150	75	25
	$rlt_2^{S_1}$	shift, random, IHClahc, IG-D/R(4)	shift, random, IHCfirst, IG-D/R(4)	swap, random, IHClahc, IG-D/R(3)	kick(3)	200	75	75
	$rlt_3^{S_1}$	shift, random, IHClahc, IG-D/R(4)	shift, random, IHCfirst, IG-D/R(2)	shift, random, IHCworst, IG-D/R(2)	kick(3)	150	75	75
	$rlt_4^{S_1}$	shift, random, IHCfirst, IG-D/R(3)	shift, random, IHCbest, IG-D/R(4)	shift, random, IHCworst, IG-D/R(3)	kick(5)	150	75	25
$S_{N=100}$	$rlt_1^{S_2}$	shift, random, IHCfirst, IG-D/R(4)	swap, random, IHClahc, IG-D/R(2)	–	kick(5)	150	25	–
	$rlt_2^{S_2}$	shift, random, IHCfirst, IG-D/R(2)	shift, random, IHCbest, IG-D/R(2)	–	kick(3)	150	50	–
	$rlt_3^{S_2}$	shift, random, IHClahc, IG-D/R(2)	shift, random, IHCfirst, IG-D/R(4)	shift, random, IHCworst, IG-D/R(4)	kick(4)	100	50	25
	$rlt_4^{S_2}$	shift, random, IHClahc, IG-D/R(2)	swap, order, IHCfirst, IG-D/R(2)	shift, random, IHCfirst, IG-D/R(4)	kick(4)	200	50	25
$S_{N=200}$	$rlt_1^{S_3}$	shift, random, IHClahc, IG-D/R(4)	shift, random, IHCfirst, IG-D/R(2)	–	kick(5)	200	50	–
	$rlt_2^{S_3}$	swap, random, IHCbest, IG-D/R(3)	shift, random, IHCfirst, IG-D/R(2)	–	kick(5)	150	50	–
	$rlt_3^{S_3}$	shift, random, IHCfirst, IG-D/R(2)	swap, random, IHClahc, IG-D/R(4)	swap, random, IHClahc, IG-D/R(4)	kick(3)	150	25	25
	$rlt_4^{S_3}$	shift, random, IHClahc, IG-D/R(2)	swap, order, IHCworst, IG-D/R(2)	swap, order, IHCfirst, IG-D/R(2)	kick(4)	150	50	75
$S_{M=20}$	$rlt_1^{S_4}$	shift, random, IHCfirst, IG-D/R(2)	swap, random, IHClahc, IG-D/R(3)	swap, random, IHCworst, IG-D/R(4)	kick(3)	200	50	75
	$rlt_2^{S_4}$	shift, random, IHCfirst, IG-D/R(4)	swap, order, IHCfirst, IG-D/R(2)	shift, random, IHClahc, IG-D/R(3)	kick(4)	150	75	75
	$rlt_3^{S_4}$	shift, random, IHCfirst, IG-D/R(4)	swap, random, IHClahc, IG-D/R(3)	swap, random, IHCbest, IG-D/R(2)	kick(4)	150	50	75

5.4.2 Results on TSP

Result of Fixed Model. We present the best configurations returned by irace over configuration space 10^9. Table 5.3 show the four best configurations of $\mathbf{S}_u$ ($fix_1^{S_u}$, $fix_2^{S_u}$, $fix_3^{S_u}$, $fix_4^{S_u}$) scenario returned by irace for fixed P-MC-ILS and three tuned ILS only were performing better on training instances. It is contained the best single-configuration at present before. It appears that three ILS leads to better performance because the one and two ILS was in the configuration space which has not been returned. In ILS_3 for $fix_1^{S_u}$, $fix_3^{S_u}$ the `swap` neighborhood operator is preferred with `order` neighborhood order. Surprising, `IHClahc` exploration is used twice ($fix_3^{S_u}$ in ILS_2 and $fix_1^{S_u}$ in ILS_3) which is in keeping with the literature. However, all configurations also share the stagnation criterion set to 150, while the `kick` and `init` diversification are used in $\{fix_1^{S_u}, fix_4^{S_u}\}$ and $\{fix_2^{S_u}, fix_3^{S_u}\}$ respectively.

For $\mathbf{S}_c$ scenario, the best four configurations have been returned by irace. All have three tuned ILS. Analyzing the fixed P-MC-ILS, we notice that the only different $fix_3^{S_c}$ between the other configurations are the neighborhood operator of ILS_1 that one include not the set of parameters corresponding to the best exhaustive configuration exh of R-ILS (see chapter 3). In $fix_2^{S_c}$ used the `kick` diversification, while the other configurations used the `init` diversification where is so far from the current one in the search space. Moreover, $fix_1^{S_c}$ used the stagnation criterion set to 150, while the other configurations are share to set to 200 stagnation criterion.

Result of Roulette Model. Table 5.4 shows that the best four algorithm configurations of $\mathtt{S}_u$ ($rlt_1^{S_u}$, $rlt_2^{S_u}$, $rlt_3^{S_u}$, $rlt_4^{S_u}$) are obtained from the irace over configuration space 10^{10}. There are three tuned ILS only and also included the best single-configuration model. Surprising, the `shift` neighborhood operator for $rlt_2^{S_u}$ in ILS_1 and $rlt_2^{S_u}$, $rlt_3^{S_u}$ in ILS_3 enable to roulette P-MC-ILS. This model

selects, after each restart, one of the tuned with 50% probability for all algorithm configurations except of $rlt_1^{S_u}$ gives 25% for P_1, while the probability are inverted between P_1 and P_2. Moreover, the $fix_3^{S_u}$ used stagnation criterion sets to 150, the other ones share to set to 200. It seems that the `init` diversification is one of the best for applying to diversification and it is better for a satisfying assignment anew from a new location so far in the search space.

For $\mathtt{S}_c$ scenario, the four best configurations ($rlt_1^{S_c}$, $rlt_2^{S_c}$, $rlt_3^{S_c}$, $rlt_4^{S_c}$) are achieved by irace and there are three tuned ILS. All algorithm configurations are contained the best single-configuration model when training across all instances. The `IHCbest` strategy is used, usually considered inefficient. It is beneficial to keep a variety of operations. The `kick` diversification (partial restart) is used for only $rlt_1^{S_c}$, the other ones are used `init` diversification. Moreover, we observe the probability of first three parameter configurations ($fix_1^{S_c}$, $fix_2^{S_c}$, $fix_3^{S_c}$) where are inverted between P_1 and P_2 of first quartile and third quartile, while last one algorithm configuration ($fix_4^{S_c}$) is only used 50% for P_1 and P_2.

Table 5.3: Best configurations of the fixed P-MC-ILS for TSP returned by irace.

Size	**Conf.**	ILS_1	ILS_2	ILS_3	Diversification	stg
S_u	$fix_1^{S_u}$	3-opt, random, IHCfirst, DB(1)	shift, random, IHCfirst, DB(4)	swap, order, IHClahc, DB(1)	kick(5)	150
	$fix_2^{S_u}$	3-opt, order, IHCfirst, DB(3)	2-opt, random, IHCbest, DB(4)	3-opt, random, IHCfirst, DB(2)	init	150
	$fix_3^{S_u}$	3-opt, random, IHCfirst, DB(2)	shift, random, IHClahc, DB(1)	swap, order, IHCworst, DB(3)	init	150
	$fix_4^{S_u}$	2-opt, random, IHCbest, DB(1)	shift, random, IHCworst, DB(2)	3-opt, random, IHCfirst, DB(4)	kick(4)	150
S_c	$fix_1^{S_c}$	3-opt, random, IHCfirst, DB(1)	shift, order, IHCbest, DB(3)	swap, random, IHCbest, DB(4)	init	150
	$fix_2^{S_c}$	3-opt, random, IHCfirst, DB(2)	shift, random, IHCbest, DB(4)	2-opt, order, IHCbest, DB(1)	kick(5)	200
	$fix_3^{S_c}$	2-opt, random, IHCbest, DB(4)	shift, order, IHCbest, DB(4)	3-opt, random, IHCfirst, DB(2)	init	200
	$fix_4^{S_c}$	3-opt, random, IHCfirst, DB(1)	shift, random, IHClahc, DB(2)	swap, order, IHCfirst, DB(1)	init	200

Table 5.4: Best configurations of the roulette P-MC-ILS for TSP returned by irace.

Size	**Conf.**	ILS_1	ILS_2	ILS_3	Diversification	stg	P_1	P_2
S_u	$rlt_1^{S_u}$	3-opt, random, IHCfirst, DB(3)	swap, random, IHCfirst, DB(4)	2-opt, random, IHCbest, DB(1)	init	200	25	50
	$rlt_2^{S_u}$	shift, order, IHCfirst, DB(2)	3-opt, random, IHCfirst, DB(3)	shift, order, IHCworst, DB(4)	init	200	50	25
	$rlt_3^{S_u}$	3-opt, random, IHCfirst, DB(3)	2-opt, random, IHCbest, DB(3)	shift, random, IHCfirst, DB(1)	init	150	50	25
	$rlt_4^{S_u}$	2-opt, order, IHCbest, DB(3)	3-opt, random, IHCfirst, DB(3)	swap, order, IHCworst, DB(3)	init	200	50	25
S_c	$rlt_1^{S_c}$	2-opt, random, IHCbest, DB(3)	3-opt, random, IHCfirst, DB(1)	2-opt, random, IHClahc, DB(4)	kick(4)	200	25	75
	$rlt_2^{S_c}$	3-opt, random, IHCfirst, DB(1)	3-opt, random, IHCbest, DB(3)	swap, order, IHCbest, DB(2)	init	200	75	25
	$rlt_3^{S_c}$	3-opt, order, IHCbest, DB(1)	3-opt, random, IHCfirst, DB(2)	shift, random, IHCworst, DB(3)	init	50	25	75
	$rlt_4^{S_c}$	3-opt, random, IHCfirst, DB(2)	2-opt, random, IHCbest, DB(3)	shift, order, IHClahc, DB(2)	init	150	50	50

5.5 Comparisons of the Automatic Multi-Configuration ILS models

In Chapter 3, we presented our baseline algorithms, the R-ILS and the random multi-configuration ILS (R-MC-ILS). In Chapter 4, we presented a first model of automated design of multi-configuration ILS, the Sequential Multi-Configuration ILS (S-MC-ILS). In this chapter, we presented a second model of automated design of multi-configuration ILS, the Probabilistic Multi-Configuration ILS (P-MC-ILS) with two instances being the fixed and the roulette models. Experiments have been conducted with a maximum of three configurations of ILS returned by irace. In this section, we compare our three automated multi-configuration ILS models with our two baseline algorithms.

5.5.1 Experimental Protocol

First, we compare the fixed P-MC-ILS and the roulette P-MC-ILS in order to select the best configurations. Then, we compare of the two automated multi-configuration ILS models, the S-MC-ILS with and P-MC-ILS. Finally, we compare the automated multi-configuration ILS with the two baselines algorithms, namely the R-ILS and the random multi-configuration ILS. We give the comparison considering four scenarios of the PFSP (S_{all}, $S_{N=100}$, $S_{N=200}$, $S_{M=20}$) and two scenarios of TSP (S_u, S_c) and the comparison per size of the problem instances.

5.5.2 Experimental Results

The experiments were executed on 80 nodes of four 20-core 2.20GHz Intel Xeon Processor (Skylake, IBRS) with 16MB L3 cache and 500GB RAM, running Ubuntu 18.04.4 LTS.

5.5.2.1 Results on PFSP

In order to compare our two models of P-MC-ILS, namely fixed (fix) and roulette (rlt), we run at least six configurations of two multi-configuration models presented above on the Taillard instances. We average the 30 RPD per instances and compute the Friedman test using the 10 instances per size to rank the configurations. Table 5.5 presents the result of the statistical comparison between the configurations. The configurations marked with '+' statistically outperform the ones with '-' for the considered instances. First, we give the comparison considering all the scenarios. Clearly, the roulette multi-configuration model is more robust than the fixed model. Indeed, if we focus on all instances (S_{all}) as we recall that the tuning has been performed considering all sizes, and $S_{M=20}$ found that the four best configurations and the three best configurations known for the roulette P-MC-ILS respectively are best ranked while fix_3 is less efficient. Moreover, four best configurations of fixed P-MC-ILS are best ranked in S_{100}, while S_{200} scenario of both models are less efficient at least once.

With Table 5.6, we showed the comparison per size of the problem for different scenario tested, the irace gives better configurations when training used similar instance (sharing the same number of jobs or machines). If we look at the results for each size, fix_1 of fixed P-MC-ILS and rlt_1 of roulette P-MC-ILS are always best ranked. In these multi-ILS configurations, not only the best R-ILS configurations are represented. This shows the importance of putting in the configuration space used by the configurator all the possible components known for a problem with no *a priori* knowledge.

In the following, we will compare the sequential multi-configuration ILS with the probabilistic multi-configuration ILS. We select the best configuration to represent the fixed P-MC-ILS and roulette P-MC-ILS where are the configurations marked

with '+' statistically outperform.

Table 5.5: Statistical comparison of fixed P-MC-ILS and roulette P-MC-ILS for each scenario on Taillard instances.

Scenario	**Fixed**				**Roulette**			
	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
S_{all}	+	+	−		+	+	+	+
$S_{N=100}$	+	+	+	+	+	+	−	+
$S_{N=200}$	+	+	−		+	−	+	+
$S_{M=20}$	+	+	−		+	+	+	

Table 5.6: Statistical comparison of fixed P-MC-ILS and roulette P-MC-ILS for each size on Taillard instances.

Size	Fixed													Roulette												
	S_{all}			$S_{N=100}$				$S_{N=200}$			$S_{M=20}$			S_{all}				$S_{N=100}$				$S_{N=200}$				$S_{M=20}$		
	fix_1	fix_2	fix_3	fix_1	fix_2	fix_3	fix_4	fix_1	fix_2	fix_3	fix_1	fix_2	fix_3	rlt_1	rlt_2	rlt_3	rlt_4	rlt_1	rlt_2	rlt_3	rlt_4	rlt_1	rlt_2	rlt_3	rlt_4	rlt_1	rlt_2	rlt_3
50×20	+	+	−								+	+	+	+	+	−	−								+	+	+
100×10	+	+	−	+	+	+	+							+	+	+	+	+	+	−	+							
100×20	+	+	−	+	−	+	+				+	+	−	+	−	+	−	+	+	−	+				+	−	+
200×10	+	+	−					+	+	−				+	−	+	+					+	−	+	+			
200×20	+	+	−					+	+	−	+	+	−	+	+	+	+					+	−	+	+	+	+	−

Table 5.7 shows the comparison between S-MC-ILS with the best configurations returned by irace and P-MC-ILS with the best configurations. Obviously, P-MC-ILS is more robust than S-MC-ILS. Indeed, the performance $S_{N=100}$ scenario of S-MC-ILS is equal to P-MC-ILS, while the other scenarios of P-MC-ILS are more robust than S-MC-ILS.

Table 5.8 summarizes, for each size of PFSP, the result of the statistical tests between the S-MC-ILS and the two models of P-MC-ILS for different size tested. Certainly, based on these experiments, the configurations are marked with the plus sign (+) statistically of each scenarios per size of instance that we regard the results for each size, seq_2 of S_{all} and seq_1 of $S_{N=100}$ and seq_1, seq_4 of $S_{N=200}$ are best ranked, while the two configurations (seq_1, seq_2) of $S_{M=20}$ as shared number of machines scenario, are best ranked for S-MC-ILS. Meanwhile, the fix_1 and rlt_1 are always best ranked for P-MC-ILS.

In the following, we select the best configurations statistically (plus sign (+)), for the automated multi-configuration ILS models to compare with the baseline algorithms.

Table 5.7: Statistical comparison of S-MC-ILS and P-MC-ILS for each scenario on Taillard instances.

Scenario	**Sequential**				**Probabilistic**							
	seq_1	seq_2	seq_3	seq_4	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
S_{all}	+	+	−		+	+			+	+	+	+
$S_{N=100}$	+	+	+		+	+	+	+	+	+		+
$S_{N=200}$	+	−	−	+	+	+			+		+	+
$S_{M=20}$	+	+	−		+	+			+	+	+	

Table 5.8: Statistical comparison of S-MC-ILS and P-MC-ILS for each size on Taillard instances.

Size	Sequential														Probabilistic																							
	S_{all}			$S_{N=100}$			$S_{N=200}$				$S_{M=20}$				S_{all}						$S_{N=100}$							$S_{N=200}$							$S_{M=20}$					
	seq_1	seq_2	seq_3	seq_1	seq_2	seq_3	seq_1	seq_2	seq_3	seq_4	seq_1	seq_2	seq_3		fix_1	fix_2	rlt_1	rlt_2	rlt_3	rlt_4	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_4	fix_1	fix_2	rlt_1	rlt_3	rlt_4	fix_1	fix_2	fix_3	rlt_1	rlt_2	rlt_3
50×20	+	+	+								+	+	−		+	+	+	+															+	+	−	+	+	+
100×10	+	+	−	+	−	+									+	+	+	+	+	−	+	+	+	−	+	+	+											
100×20	+	+	−	+	+	−					+	+	−		+	−	+		+		+		+		+	+	−						+	−		+		+
200×10	−	+	−				+	−	−	+					+	+	+		+	+								+	−	+	+	+						
200×20	−	+	−				+	−	−	+	+	+	−		+	−	+	+	+	+								+	+	+	+	+	+	+		+	+	

We compare our automated multi-configuration ILS models with baseline algorithms, the random multi-configuration model and the R-ILS on the Taillard instances. Table 5.9 gives, for each scenario, the result of the statistical test those. For all scenarios, the automatic configuration model outperforms the random multi-configuration and single-configuration model. For this problem, we designed different experiment scenarios in order to see whether it is better to train irace on rather all instances or instances sharing the same number of jobs or machines.

Table 5.10 summarises, for each size of PFSP, the results the statistical tests compared of automated design of multi-configuration model always outperforms the random multi-ILS and the best configurations previously obtained in the exhaustive analysis of the single-configuration model.

Table 5.9: Statistical comparison of the automated multi-configuration models with the baseline algorithms for each scenario on Taillard instances.

Scenario	**W/O Automatic Conf.**			**Automatic Configuration**											
	Single-config.		**Random**	**Sequential**				**Fixed**				**Roulette**			
	exh_1	exh_2	rnd	seq_1	seq_2	seq_3	seq_4	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
S_{all}	−	−	−	+	+			+	+			+	+	+	+
$S_{N=100}$	−	−	−	+	+	+		+	+	+	+	+	+		+
$S_{N=200}$	−	−	−	+			+	+	+			+		+	+
$S_{M=20}$	−	−	−	+	+			+	+			+	+	+	

Table 5.10: Statistical comparison of the automated multi-configuration models with the baseline algorithms for each size on Taillard instances.

Size	W/O Automatic Conf.			Automatic Configuration																																			
	Single-config.		Random	S_{all}										$S_{N=100}$										$S_{N=200}$								$S_{M=20}$							
	exh_1	exh_2	rnd	seq_1	seq_2	seq_3	fix_1	fix_2	rlt_1	rlt_2	rlt_3	rlt_4	seq_1	seq_2	seq_3	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_4	seq_1	seq_4	fix_1	fix_2	rlt_1	rlt_3	rlt_4	seq_1	seq_2	fix_1	fix_2	rlt_1	rlt_2	rlt_3
50×20	–	–	–	+	+	+	+	+	+	+																			+	+	+	+	+	+	+
100×10	–	–	–	+	+		+	+	+	+	+		+		+	+	+	+		+	+	+													
100×20	–	–	–	+	+		+		+		+		+	+		+		+	+	+	+									+	+	+		+		+
200×10	–	–	–		+		+	+	+		+	+											+	+	+		+	+	+								
200×20	–	–	–		+		+		+	+	+	+											+	+	+	+	+	+	+	+	+	+	+	+	+		

5.5.2.2 Results on TSP

Tables 5.11 and 5.12 report the results of the statistical Friedman test between the performance of S-MC-ILS and P-MC-ILS on Taillard instances presented by scenario or size respectively. The results of Table 5.11 show that the roulette P-MC-ILS performs best on statistically test (mark '+') in terms of four (rlt_1, rlt_2, rlt_3, rlt_4) of S_u and three (rlt_1, rlt_2, rlt_3) of S_c configurations are best ranked while rlt_4 of S_c is less efficient. Moreover, fix_2 and fix_4 are best ranked for two scenarios, nevertheless the fix_1 of S_c is additionally best ranked. Then, Table 5.12 shows three different instance sizes including 100, 200, and 400 cities for each model. If we look at the results for each size, fix_2, rlt_1, and rlt_3 of S_u have always best ranks. The best ranks, not only the best single-configuration model is presented previously.For S_c, fix_2, fix_4, rlt_1, rlt_2 and rlt_3 are best ranked.

In the following, we will compare the S-MC-ILS with the best configurations of P-MC-ILS with fixed or roulette models.

Table 5.11: Statistical comparison of fixed P-MC-ILS and roulette P-MC-ILS for each scenario on TSP instances.

Scenario	**Fixed**				**Roulette**			
	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
Random Uniform Euclidean Instances								
S_c	−	+	−	+	+	+	+	+
Random 10-Cluster Euclidean Instances								
S_u	+	+	−	+	+	+	+	−

Next, we analyzed the resulting algorithm configuration on the test set instances. Our first finding is that the P-MC-ILS clearly outperforms the S-MC-ILS for six configurations of S_u scenario and five configurations of S_c where rlt_4 is eliminated

Table 5.12: Statistical comparison of fixed P-MC-ILS and roulette P-MC-ILS for each size on TSP instances.

Size	**Fixed**				**Roulette**			
	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
Random Uniform Euclidean Instances								
100	+	+	+	+	+	+	+	+
200	−	+	−	−	+	−	+	+
400	−	+	−	+	+	+	+	−
Random 10-Cluster Euclidean Instances								
100	+	+	−	+	+	+	+	+
200	+	+	−	+	+	+	+	−
400	−	+	−	+	+	+	+	−

previously, while seq_2 of two scenarios, seq_1 of only S_u and seq_4 of S_c are best ranked of sequential multi-configuration model in the Table 5.13. Thus, we focus the results of each size (Table 5.14), we find the fix_2 and rlt_1 which are always best ranked of the P-MC-ILS for S_u, while the $fix_2, fix_4, rlt_1, rlt_2$, and rlt_3 are always best ranked for S_c. On the other hand, the best ranks of S-MC-ILS are not found because of the 400 cities of the two scenarios are not found the best solution statistically.

Finally, we compare the automated multi-configuration models with our baseline algorithms.

The results of Table 5.15 and 5.16 show that the automated design of the multi-configuration model outperform the other ones statistically. Interestingly, it seems

Table 5.13: Statistical comparison of S-MC-ILS and P-MC-ILS for each scenario on TSP instances.

Scenario	**Sequential**				**Probabilistic**						
	seq_1	seq_2	seq_3	seq_4	fix_1	fix_2	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
Random Uniform Euclidean Instances											
S_c	+	+	−		+	+	+	+	+	+	
Random 10-Cluster Euclidean Instances											
S_u	−	+	−	+	−	+	+	+	+	+	

Table 5.14: Statistical comparison of S-MC-ILS and P-MC-ILS for each instance on TSP instances.

Size	**Sequential**				**Probabilistic**							
	seq_1	seq_2	seq_3	seq_4	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
Random Uniform Euclidean Instances												
100	−	+	−		+	+	+	+	+	+	−	+
200	+	+	−		+				+		+	+
400	+	−	−		+		+	+	+	+	+	
Random 10-Cluster Euclidean Instances												
100	+	+	+	−	+	+		+	+	+	+	+
200	−	+	−	+	−	+		+	+	+	+	
400	−	−	−	+		+		+	+	+	+	

that, here, only the three tuned ILS of the multi-configuration ILS are preferable for these scenarios and explored to a greater extent to the rest of the scenarios across the 3 problems.

Table 5.15: Statistical comparison of the automated multi-configuration models with the baseline algorithms for each scenario on TSP instances.

Scenario	W/O Automatic Conf.			Automatic Configuration								
	Single-config.		Random	Sequential			Fixed		Roulette			
	exh_1	exh_2	rnd	seq_1	seq_2	seq_4	fix_2	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
Random Uniform Euclidean Instances												
S_u	−	−	−	+	+		+	+	+	+	+	+
Random 10-Cluster Euclidean Instances												
S_c	−	−	−		+	+	+	+	+	+	+	

Table 5.16: Statistical comparison of the automated multi-configuration models with the baseline algorithms for each size on TSP instances.

Size	W/O Automatic Conf.			Automatic Configuration											
	Single-config.		Random	Sequential				Fixed				Roulette			
	exh_1	exh_2	rnd	seq_1	seq_2	seq_3	seq_4	fix_1	fix_2	fix_3	fix_4	rlt_1	rlt_2	rlt_3	rlt_4
Random Uniform Euclidean Instances															
100	−	−	−		+			+	+	+	+	+	+		+
200	−	−	−	+	+				+			+		+	+
400	−	−	−	+					+		+	+	+	+	
Random 10-Cluster Euclidean Instances															
100	−	−	−	+	+	+		+	+		+	+	+	+	+
200	−	−	−		+		+		+		+	+	+	+	
400	−	−	−				+		+		+	+	+	+	

5.6 Conclusion

In this Chapter, we presented the Probabilistic Multi-configuration ILS that modifies the configuration of a R-ILS at each restart. The choice of the next configuration can be done following either a fixed model where a sequence of tuned configurations are applied, or a roulette model where tuned configurations are applied according to the roulette wheel execution. The tuned configurations are

selected using AAC, and more precisely the irace configurator. The P-MC-ILS framework is then a second approach to benefit from both parameter tuning and parameter control. In the experiments, we allowed to use up to three configurations to solve instances of PFSP or TSP. The results are equivalent for both problems. We showed that both fixed and roulette models best performed using more than 2 tuned configurations. The roulette model is also more robust than the fixed one. The comparison with the Sequential Multi-Configuration ILS shows that the Probabilistic Multi-Configuration ILS leads to better performance with both fixed and roulette models. Finally, the comparison with our baseline algorithms, R-ILS and random multi-configuration ILS, shows the advantage of using an automated multi-configuration ILS.

6 | Conclusion

Contents

Automatic algorithm design and in particular automatic algorithm configuration has been more and more studied in the literature for the last ten years. In the majority of the cases, the tools provide a unique configuration that should be used during the execution of the algorithm. This thesis aimed to bridge the gap between parameter tuning and parameter control by allowing the alteration of parameter values and the modification of strategic components during the execution. Such mechanisms could allow for a better balance between exploitation and exploration in local search metaheuristics when tackling complex problems.

6.1 Contribution Summary

In this thesis, we proposed automated multi-configuration models that find the best performing hyper-parameters of a given algorithm and adapt its parameters depending on the problem. The target algorithm chosen in this thesis is a restart iterated local search algorithm (R-ILS). The restart mechanism is applied when stagnation is reached. We illustrated our work on two well-known combinatorial problems: Permutation Flowshop Problem (PFSP) and Travelling Salesman Problem (TSP). We defined for the R-ILS and for both problems a configuration space with five strategic components and three parameters leading to 1200 and 3200 possible configurations respectively. We made multiple campaigns of experiments that showed the interest of the multi-configuration ILS. The performance was analyzed with the help of Friedman's test on different scenarios or instance sizes separately.

6.1.1 Automated Multi-Configuration ILS

We presented two algorithms to be our baselines for the experiments. We defined a small configuration space for R-ILS by fixing the parameter values to reduce the

size to 32. The parameter values have been fixed according to the better values known for each problem in the literature. Therefore, we were able to exhaustively evaluate the performance of the 32 configurations of R-ILS. The best configurations gave the first baseline. We also defined a random multi-configuration model applied with R-ILS where a configuration is randomly selected in the original search space of R-ILS after each restart. This algorithm represented our second baseline. We showed that the best configurations of R-ILS outperformed the random multi-configuration ILS. Moreover, the experimental results showed that the automated multi-configuration ILS models presented in this thesis statistically outperformed the baseline algorithms.

6.1.2 Sequential and Probabilistic Frameworks

We presented two different frameworks of automated multi-configuration ILS. These multi-configuration models were tuned using AAC and more specifically the irace configurator, without loss of generality. The combinatorics of possible configurations is explosive with the number of tuned R-ILS. Therefore, for all experiments conducted in this thesis, we allowed up to three different tuned R-ILS configurations to keep a *reasonable* configuration space (about 10^{11} possible configurations with 3 tuned R-ILS for the TSP). The sequential multi-configuration model alters the configuration of a R-ILS when a number of evaluation with the same configuration is reached while the Probabilistic Multi-Configuration model alters the configuration as soon as the restart is applied. The first model executes the tuned configurations successively and once for each while the second model is more flexible and is able to execute multiple times each tuned configurations. The experiments showed that the probabilistic multi-configuration ILS statistically outperformed the sequential multi-configuration ILS for both tackled problems. Therefore, it seems better to modify the configuration after each restart

like for the baseline random multi-configuration ILS but it is essential to reduce the choice using parameter tuning.

6.1.3 Fixed and Roulette Models

We presented two models, namely fixed and roulette, to select the R-ILS configuration after each restart of the Probabilistic Multi-Configuration ILS. With the fixed model, the probabilistic Multi-Configuration ILS iterates successively the tuned configurations in the same order during the whole run. With the roulette model, the tuned configurations are selected following a roulette wheel mechanism to better handle intensification and exploration of the search space. For example, with two tuned configurations, it would be possible to have one R-ILS configured to intensify the search in a region while the second R-ILS configured would be more explorative. The experiments showed that the performance of both models are comparable but the roulette model seems to give more robust performance than ones obtained with the fixed model.

Consequently, we investigated the benefit of using parameter tuning, through automatic algorithm configuration, and parameter control simultaneously in the design.

6.2 Future Research

The contributions detailed in this manuscript have been presented in international workshops and conferences. In spite of this, we have not been able to study everything that was initially planned. For example, we only considered a restart mechanism based on the stagnation while more complex ones could be used. Moreover, we could have studied other combinatorial problems or metaheuristics. In the following, we propose some other future works.

6.2.1 Increase the number of tuned R-ILS

The automated multi-configuration model enables the use of multiple tuned R-ILS during a whole execution. However, the number of tuned configurations exponentially increases the size of the associated configuration space. That is why, we bounded this number and we allowed a maximum of three tuned R-ILS. In future work, the number of configurations should be increased. Experiments would allow to analyze if our multi-configurations are still interesting with more tuned configurations.

6.2.2 Analysis of the multi-configuration ILS algorithms

The configuration space of R-ILS was defined from strategic components with more or less good efficiency. In the experiments, we saw the importance of keeping all the component strategies in the configuration space since some automated multi-configuration ILS used strategies known to be less efficient in the literature. In future work, we would like to deeper analyze the components selected during the parameter tuning in order to understand their role and impact in the global or local performance of our multi-configuration ILS algorithms.

References

Adenso-Díaz, B., & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, *54*(1), 99–114.

Allahverdi, A., & Aldowaisan, T. (2002). New heuristics to minimize total completion time in m-machine flowshops. *International Journal of Production Economics*, *77*(1), 71-83. doi: 10.1016/S0925-5273(01)00228-6

Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., & Tierney, K. (2015). Model-based genetic algorithms for algorithm configuration. In *Proceedings of the 24th International Conference on Artificial Intelligence* (p. 733–739). AAAI Press.

Ansótegui, C., Pon, J., Sellmann, M., & Tierney, K. (2021). Pydgga: Distributed GGA for automatic configuration. In C. Li & F. Manyà (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings* (Vol. 12831, pp. 11–20). Springer.

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In I. P. Gent (Ed.), *Principles and Practice of Constraint Programming - CP 2009* (pp. 142–157). Berlin, Heidelberg: Springer Berlin Heidelberg.

Balaprakash, P., Birattari, M., & Stützle, T. (2007). Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein et al. (Eds.), *Hybrid Metaheuristics* (pp. 108–122). Berlin, Heidelberg: Springer Berlin Heidelberg.

Barahona, F., Grötschel, M., Jünger, M., & Reinelt, G. (1988). An application of combinatorial optimization to statistical physics and circuit layout design.

Operations Research, *36*(3), 493–513.

Bartz-Beielstein, T., Lasarczyk, C., & Preuss, M. (2010). The sequential parameter optimization toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, & M. Preuss (Eds.), *Experimental methods for the Analysis of Optimization Algorithms* (pp. 337–362). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-02538-9_14

Bazargani, M., & Lobo, F. G. (2017). Parameter-less late acceptance hill-climbing. In *Proceedings of the Genetic and Evolutionary Computation Conference* (p. 219–226). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3071178.3071225

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., & Lindauer, M. (2020). Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *Proceedings of the twenty-fourth European Conference on Artificial Intelligence (ECAI'20)* (p. 427-434).

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th annual Conference on Genetic and Evolutionary Computation* (p. 11–18). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Blot, A., Hoos, H., Jourdan, L., Marmion, M.-É., & Trautmann, H. (2016). MO-ParamILS: A Multi-objective Automatic Algorithm Configuration Framework. In *Learning and Intelligent Optimization* (Vol. 10079, p. 32-47). Ischia, Italy. doi: 10.1007/978-3-319-50349-3_3

Blot, A., Hoos, H. H., Kessaci, M.-É., & Jourdan, L. (2018). Automatic configuration of bi-objective optimisation algorithms: Impact of correlation between objectives. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)* (p. 571-578). doi: 10.1109/ICTAI.2018.00093

Böttcher, S., Doerr, B., & Neumann, F. (2010). Optimal fixed and adaptive

mutation rates for the leadingones problem. In R. Schaefer, C. Cotta, J. Kołodziej, & G. Rudolph (Eds.), *Parallel Problem Solving from Nature, PPSN xi* (pp. 1–10). Berlin, Heidelberg: Springer Berlin Heidelberg.

Burke, E. K., & Bykov, Y. (2017). The late acceptance hill-climbing heuristic. *European Journal of Operational Research*, *258*(1), 70-78. doi: 10.1016/j.ejor.2016.07.012

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, *176*(1), 177-192. doi: 10.1016/j.ejor.2005.08.012

Cheng, C.-Y., Li, S.-F., & Lin, Y.-C. (2019). Self-adaptive parameters in differential evolution based on fitness performance with a perturbation strategy. *Soft Comput.*, *23*(9), 3113–3128. doi: 10.1007/s00500-017-2958-z

Clay, S., Mousin, L., Veerapen, N., & Jourdan, L. (2021). Clahc - custom late acceptance hill climbing: First results on tsp. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (p. 1970–1973). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3449726.3463129

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, *6*(6), 791–812.

DaCosta, L., Fialho, A., Schoenauer, M., & Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (p. 913–920). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1389095.1389272

de Moraes Barbosa, E. B., & Senne, E. L. F. (2017). Improving the fine-tuning of metaheuristics: An approach combining design of experiments and racing

algorithms. *Journal of Optimization*, *2017*, 1-7.

Doerr, B., & Doerr, C. (2018, June). Theory of Parameter Control for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices. *arXiv:1804.05650 [cs]*.

Doerr, B., Witt, C., & Yang, J. (2018). Runtime analysis for self-adaptive mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference* (p. 1475–1482). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3205455.3205569

Dong, X., Chen, P., Huang, H., & Nowak, M. (2013). A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers & Operations Research*, *40*(2), 627-632. doi: 10.1016/j.cor.2012.08.021

Eiben, A., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *3*(2), 124–141. doi: 10.1109/4235.771166

Fan, Q., & Yan, X. (2016). Self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies. *IEEE Transactions on Cybernetics*, *46*(1), 219-232. doi: 10.1109/TCYB .2015.2399478

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. M. (2017). 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), 707-721. doi: 10.1016/j.ejor.2016.09.055

Feutrier, T., Kessaci, M.-E., & Veerapen, N. (2021). Investigating the landscape of a hybrid local search approach for a timetabling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (p. 1665–1673). New York, NY, USA: Association for Computing Machin-

ery. doi: 10.1145/3449726.3463175

Fialho, Á., Da Costa, L., Schoenauer, M., & Sebag, M. (2008). Extreme Value Based Adaptive Operator Selection. In *10th International Conference on Parallel Problem Solving From Nature (PPSN X)* (Vol. 5199/2008, p. 175-184). Dortmund, Germany. doi: 10.1007/978-3-540-87700-4_18

Framinan, J. M., & Leisten, R. (2008). Total tardiness minimisation in permutation flow shops: a simple approach based on a variable greedy algorithm. *International Journal of Production Research*, *46*(22), 6479-6498. doi: 10.1080/00207540701418960

Framinan, J. M., Leisten, R., & Ruiz-Usano, R. (2005). Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers & Operations Research*, *32*(5), 1237-1254. doi: 10.1016/j.cor.2003.11.002

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533-549. (Applications of Integer Programming) doi: 10.1016/0305-0548(86)90048-1

Glover, F. (1989). Tabu search—part i. *ORSA Journal on Computing*, *1*(3), 190-206. doi: 10.1287/ijoc.1.3.190

Glover, F. (1990). Tabu search—part ii. *ORSA Journal on Computing*, *2*(1), 4-32. doi: 10.1287/ijoc.2.1.4

Guizzo, G., Vergilio, S. R., Pozo, A. T., & Fritsche, G. M. (2017). A multi-objective and evolutionary hyper-heuristic applied to the integration and test order problem. *Appl. Soft Comput.*, *56*(C), 331–344. doi: 10.1016/j.asoc .2017.03.012

Henderson, D., Jacobson, S. H., & Johnson, A. W. (2003). The theory and practice of simulated annealing. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 287–319). Boston, MA: Springer US. doi: 10.1007/0-306-48056-5_10

Hevia Fajardo, M. A., & Sudholt, D. (2021). Self-adjusting population sizes for non-elitist evolutionary algorithms: Why success rates matter. In *Proceedings of the Genetic and Evolutionary Computation Conference* (p. 1151–1159). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3449639.3459338

Hoos, H. H., & Stützle, T. (2005). Stochastic local search: Foundations and applications. In H. H. Hoos & T. Stützle (Eds.), . San Francisco: Morgan Kaufmann. doi: 10.1016/B978-155860872-6/50021-4

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2012). Parallel Algorithm Configuration. In Y. Hamadi & M. Schoenauer (Eds.), *Learning and Intelligent Optimization* (Vol. 7219, pp. 55–70). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-34413-8_5

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stuetzle, T. (2009, October). ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research*, *36*, 267–306. doi: 10.1613/jair.2861

Jungnickel, D. (1999). *Graphs, Networks and Algorithms* (Vol. 5; E. Becker, M. Bronstein, H. Cohen, D. Eisenbud, & R. Gilman, Eds.). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-662-03822-2

Kalender, M., Kheiri, A., Özcan, E., & Burke, E. K. (2013). A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing*, *17*(12), 2279–2292. doi: 10.1007/s00500-013-1096-5

Karafotias, G., Hoogendoorn, M., & Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, *19*(2), 167-187. doi: 10.1109/TEVC.2014.2308294

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, *49*(2), 291-307. doi:

10.1002/j.1538-7305.1970.tb01770.x

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680. doi: 10.1126/science.220.4598.671

Lässig, J., & Sudholt, D. (2011). Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms* (p. 181–192). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1967654.1967671

Lobo, F. G., Bazargani, M., & Burke, E. K. (2020). A cutoff time strategy based on the coupon collector's problem. *European Journal of Operational Research*, *286*(1), 101-114. doi: 10.1016/j.ejor.2020.03.027

Lourenço, H. R., Martin, O. C., & Stützle, T. (2010). Iterated local search: Framework and applications. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 363–397). Boston, MA: Springer US. doi: 10.1007/978-1-4419-1665-5_12

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58. doi: 10.1016/j.orp.2016.09.002

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097-1100. doi: 10.1016/S0305-0548(97)00031-2

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91-95. doi: 10.1016/0305-0483(83)90088-9

Ochoa, G., Verel, S., & Tomassini, M. (2010). First-Improvement vs. Best-

Improvement Local Optima Networks of NK Landscapes. In R. Schaefer, C. Cotta, J. Kołodziej, & G. Rudolph (Eds.), *Parallel Problem Solving from Nature, PPSN XI* (pp. 104–113). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-15844-5_11

Oliveto, P. S., Lehre, P. K., & Neumann, F. (2009). Theoretical analysis of rank-based mutation - combining exploration and exploitation. In *2009 IEEE Congress on Evolutionary Computation* (p. 1455-1462). doi: 10.1109/CEC.2009.4983114

Pageau, C., Blot, A., Hoos, H. H., Kessaci, M.-É., & Jourdan, L. (2019). Configuration of a Dynamic MOLS Algorithm for Bi-objective Flowshop Scheduling. In K. Deb et al. (Eds.), *Evolutionary Multi-Criterion Optimization* (pp. 565–577). Cham: Springer International Publishing. doi: 10.1007/978-3-030-12598-1_45

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization: Algorithms and complexity*. USA: Prentice-Hall, Inc.

Rajaraman, K., & Sastry, P. (1996). Finite time analysis of the pursuit algorithm for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *26*(4), 590-598. doi: 10.1109/3477.517033

Rajendran, C. (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, *29*(1), 65-73. doi: 10.1016/0925-5273(93)90024-F

Rowe, J. E., & Sudholt, D. (2014). The choice of the offspring population size in the (1,lambda) evolutionary algorithm. *Theoretical Computer Science*, *545*, 20-38. (Genetic and Evolutionary Computation) doi: 10.1016/j.tcs.2013.09.036

Ruiz, R., & Stützle, T. (2006). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of*

Operational Research, *177*, 2033–2049.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*(3), 2033-2049. doi: 10.1016/j.ejor.2005.12.009

Sabar, N. R., Ayob, M., Qu, R., & Kendall, G. (2011). A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, *37*, 1-11.

Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L. (2020). Time-dependent automatic parameter configuration of a local search algorithm. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (p. 1898–1905). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3377929.3398107

Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L. (2021, December). Automatic Algorithm Multi-Configuration Applied to an Optimization Algorithm. In *21st International Conference on Hybrid Intelligent Systems (HIS 2021).* online, United States.

Stützle, T., & Hoos, H. H. (2002). Analysing the Run-Time Behaviour of Iterated Local Search for the Travelling Salesman Problem. In *Essays and Surveys in Metaheuristics* (pp. 589–611). Boston, MA: Springer US. doi: 10.1007/978-1-4615-1507-4_26

Szczepanski, N., Audemard, G., Jourdan, L., Lecoutre, C., Mousin, L., & Veerapen, N. (2021). A hybrid CP/MOLS approach for multi-objective imbalanced classification. In *Proceedings of the Genetic and Evolutionary Computation Conference* (p. 723–731). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3449639.3459310

Szczepanski, N., Mousin, L., Veerapen, N., & Jourdan, L. (2020). Automatic configuration of multi-thread local search: Preliminary results on bi-objective

tsp. In *2020 ieee 32nd International Conference on Tools with Artificial Intelligence (ICTAI)* (p. 1241-1248). doi: 10.1109/ICTAI50040.2020.00187

Taillard, E. (1993, January). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285. doi: 10.1016/0377-2217(93)90182-M

Tari, S., Basseur, M., & Goëffon, A. (2018). Worst Improvement Based Iterated Local Search. In A. Liefooghe & M. López-Ibáñez (Eds.), *Evolutionary Computation in Combinatorial Optimization* (pp. 50–66). Cham: Springer International Publishing. doi: 10.1007/978-3-319-77449-7_4

Tari, S., Hoos, H., Jacques, J., Kessaci, M.-E., & Jourdan, L. (2020). Automatic configuration of a multi-objective local search for imbalanced classification. In T. Bäck et al. (Eds.), *Parallel Problem Solving from Nature – PPSN XVI* (pp. 65–77). Cham: Springer International Publishing.

Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation* (p. 1539–1546). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1068009.1068251

Valente, J. M. S., & Alves, R. A. F. S. (2008). Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. *Comput. Oper. Res.*, *35*(7), 2388–2405. doi: 10.1016/j.cor.2006.11.004

Whitley, D., Howe, A., & Hains, D. (2013). Greedy or not? best improving versus first improving stochastic local search for maxsat. In *Proceedings of the twenty-seventh AAAI Conference on Artificial Intelligence* (p. 940–946). AAAI Press.

Yang, T., Zhang, S., & Li, C. (2021). A multi-objective hyper-heuristic algorithm based on adaptive epsilon-greedy selection. *Complex & Intelligent Systems*,

7(2), 765–780. doi: 10.1007/s40747-020-00230-8

Zamli, K. Z., Din, F., Kendall, G., & Ahmed, B. S. (2017). An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation. *Information Sciences*, *399*, 121-153. doi: 10 .1016/j.ins.2017.03.007

Zhang, T., Georgiopoulos, M., & Anagnostopoulos, G. C. (2013). S-race: A multi-objective racing algorithm. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation* (p. 1565–1572). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/ 2463372.2463561