Landscape Analysis and Solver Reconfiguration for the Curriculum-Based Course Timetabling

Thomas Feutrier

Université de Lille

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

Ecole Doctorale 631
Mathématiques, Sciences Du Numérique et de leurs Interactions

Spécialité de doctorat : Informatique

Soutenue publiquement le 12/12/2023, par :

Thomas FEUTRIER

Landscape Analysis and Solver Reconfiguration for the Curriculum-Based Course Timetabling

Analyse du paysage et reconfiguration d'algorithmes pour le problème CB-CTT d'emploi du temps universitaire

---

Devant le jury composé de :

François BOULIER Professeur des Universités, Université de Lille	**Examinateur**
Aziz MOUKRIM Professeur des Universités, Université de Technologie de Compiègne	**Rapporteur**
Frédéric SAUBION Professeur des Universités, Université d'Angers	**Rapporteur**
Farouk YALAOUI Professeur des Universités, Université de Technologie de Troyes	**Examinateur**
Marie-Eléonore KESSACI Maîtresse de conférences, Université de Lille	**Directrice de thèse**
Nadarajen VEERAPEN Maître de conférences, Université de Lille	**Co-Encadrant**

Contents

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my two thesis supervisors, Marie-Eleonore Kessaci and Nadarajen Veerapen. Throughout the three years of my thesis, they provided invaluable guidance enabling me to conduct my research and improve as a researcher. I owe much of my progress to their mentorship.

I extend my sincere appreciation to all the members of the thesis jury: François Boulier, serving as the president, and Frédéric SAUBION and Aziz MOUKRIM, who graciously took on the roles of external referees. Additionally, I would like to thank Farouk YALAOUI, who participated as an examiner during my thesis defense. Their collective expertise and attention significantly contributed to validating my work, and I am truly grateful for their dedication.

I am also deeply indebted to the ORKAD team at the CRIStAL laboratory, where I had the privilege to work. I want to acknowledge Laetitia Jourdan, Clarisse Dhaenens, Lucien Mousin, Julien Baste, Olivier Caron, and Julie Jacques for their support and collaboration. Furthermore, I am grateful for the interactions and shared experiences with my fellow Ph.D. students and post-docs over the course of my thesis, including Antoine Castillon, Guillaume Poslednik, Meyssa Zouambi, Amadeu Almeida Coco, Adan Jose-Garcia, Weerapan Sae-Dan, and Mounir Hafsa. Special recognition is due to my two Ph.D. colleagues, Clément Legrand and Metaireau Agathe, with whom I found camaraderie and mutual motivation. They made it possible for us to maintain a social life while answering the demands of our work. I also acknowledge them for the moments of shared humor and camaraderie, even for the non-productive ones.

I would like to express my deep appreciation to my friends who have been by my side since high school: Vincent, Laure, Charlotte, and Eloise. Their unwavering support has been a constant source of strength and encouragement during this thesis.

I am immensely thankful to my entire family for their consistent support throughout my long educational journey. I extend my gratitude to my grandparents, Olivier and Marguerite, Claude and Lucienne, who provided unwavering motivation and encouragement in my pursuit of studies, even when faced with challenges.

About my two brothers, Rémi and Simon, I accept to credit them for some positive things. More seriously, they have always supported me with the ups and downs in my life.

Finally, I would like to thank my two shadow supervisors, my parents: Franck and Cecile. Both of them have mastered most of the work I have done during my thesis. I would like to thank them for everything: the room and board, the attentiveness, and the unlimited support they gave me. Thank you for everything.

Introduction

This thesis presents work in the field of operational research, focusing specifically on combinatorial problems. It focuses on a university timetabling problem called Curriculum-Based Course Timetabling problem (CB-CTT). This thesis was conducted within the ORKAD (Operational Research, Knowledge, And Data) team at the CRIStAL laboratory in Lille. The ORKAD team specializes in solving combinatorial optimization problems by combining established resolution methods from operational research with artificial intelligence techniques, enabling, for example, to predict the performance of one solving method, and consequently to make informed decisions. The ORKAD team area of expertise encompasses a broad range of domains, including single and multi-objective combinatorial optimization problems, resolution methods, algorithm configuration, and graph theory.

Timetabling problems are a well-explored domain in academic research and hold considerable significance for various institutions, notably universities. This thesis centers its attention on the Curriculum-Based Course Timetabling problem. In most cases, solving a combinatorial optimization problem goes through two stages. First, an algorithm builds an initial solution by assessing the problem constraints, ensuring its feasibility. This step, often referred to as the construction phase, employs a variety of problem-specific algorithms to accomplish its task. For CB-CTT, creating an initial solution presents a complex challenge, nevertheless addressed by a highly efficient algorithm. As a result, this thesis concentrates on the subsequent phase of optimization, building upon the initial solution crafted during construction. Optimizing combinatorial problems can be resource-intensive. In the literature, Hybrid Local Search (Müller [2009]) is recognized as the state-of-the-art method and serves as the benchmark for assessing the performance of new algorithms. In the context of CB-CTT, approximate methods are widely favored due to the problem classification as NP-hard, indicating a high level of complexity. Indeed, metaheuristics are favored for their time efficiency and their strong performance. HLS is interesting due to its intricate structure, which incorporates three distinct heuristics. As a metaheuristic, it consistently yields good results. Hence, Hybrid Local Search serves as the foundation upon which this thesis proposes a flexible and tunable framework to solve CB-CTT problems.

Landscape analysis aims at characterizing instances by generating additional data from executions. This process involves new data, generally extracted from a graph that depicts the search landscape. Such data helps in understanding the nature of the problem instance and how the solver performs during the problem-solving process. This graph can be envisioned as an n-dimensional space in which each node represents a feasible solution. Edges connect solutions that are considered close in terms of neighborhood in general. This thesis employs landscape analysis to address the CB-CTT problem, introducing a landscape representation and a set of new features. Various protocols for conducting landscape analysis are explored to identify relevant instance features. The

acquired data is utilized for performance prediction enabling the assessment of protocol efficiency and feature relevance. These models predict the final performance of solvers, such as HLS, on CB-CTT instances.

Metaheuristics are efficient algorithms with many parameters and components. A set of parameters/components is called a configuration. For several years now, tuning techniques have been used to fix these parameters. These techniques explore the configuration space to find configurations that offer the best performance. Numerous techniques are available, using machine learning models, statistical models, or metaheuristics-based models. Concerning HLS, an analysis of its structure showed that it could offer better results by fine-tuning its parameters but also by modifying components of the inner heuristics. In addition, this thesis questions the fixed structure of HLS.

To facilitate the manipulation of HLS components, the thesis introduces the Iterated Sequential Local Search (ISLS) framework. This framework offers structural flexibility and serves as a generalization of local search methods. The original HLS heuristics are encapsulated within ISLS as parameter-specific variants of ISLS. Leveraging ISLS, this thesis employs the irace tuning tool, which is trained on instances that are generated and then selected. The goal is to identify improved configurations by enforcing HLS heuristics during the tuning process. As a result, irace generates HLS-like configurations with specific sub-heuristics disabled. That highlights that the original structure of HLS may not be optimally suited for all real-world instances.

Chapter 1 Chapter 1 provides an overview of the background against which this thesis work is conducted. It begins by introducing combinatorial problems and presenting the solving methods with a focus on metaheuristics divided into two groups nature-inspired algorithms and neighborhood-based algorithms. Then, we provide an overview of the Curriculum-Based Course Timetabling (CB-CTT) problem: its definition, historical context, and components such as curriculum. Additionally, details are given about the inherent constraints of CB-CTT, namely, the hard and soft constraints. Benchmarks of the literature are also presented since they are used for all the experiments detailed in the manuscript. Finally, the Tabu Memetic Algorithm (Abdullah and Turabieh [2012]) and the Hybrid Local Search (Müller [2009]), two metaheuristics designed for the CB-CTT, are presented and described in details.

Chapter 2 Chapter 2 focuses on the works regarding search landscapes. It introduces the concept of search landscapes, providing a formal definition and practical application guidelines. The work aims to efficiently represent the landscape of a given problem instance, identifying specific structures that exhibit distinct behaviors. To achieve this, a protocol has been developed and involves creating a graph representation of the landscape. The chapter elaborates on the methodology employed, including the experimentation with various sampler algorithms under different time budgets to generate multiple graph versions. Furthermore, we explain the process of extracting relevant features while capturing the characteristics of each problem instance. To demonstrate the effectiveness of the protocol, the obtained data is employed to create performance prediction models, with each model using a different graph version. The chapter then compares the performance of these models, determining the best features to incorporate to obtain the best models and the most effective graph version. Ultimately, this analysis helps identify the best protocol to follow in establishing an efficient performance prediction model.

Chapter 3 Chapter 3 concerns a study of all CB-CTT instances. There are three instance benchmarks, one of which is automatically generated. The chapter commences by detailing De Coster et al. [2022] approach to generating new artificial instances, considering instance feature values of the real world. An initial feature-based analysis compares real-world and artificial instances, revealing no significant distinctions. In addition, the work focuses on one particular aspect: the *feasibility* of constructing a first solution. It should be noted that some instances, whether generated or not, may be infeasible, i.e. no initial solution can be found. Examining feasibility by type of instance highlights differences in behavior. However, the differences are not the same depending on the type of instance, real-world and artificial instances. Whether a predictive model trained on artificial instances can predict the feasibility of real-world instances. This means that the two sets are similar. Thus, the development of a predictive model of feasibility serves as a proxy for determining whether artificial data are close enough to real-world data to perform well. The work results in the development of a Selector, enhancing the performance of predictive models trained on selected artificial instances. The entire process, from initial models through the Selector to the final feasibility predictive model, undergoes evaluation for validation and robust performance.

Chapter 4 Chapter 4 focuses on the Iterated Sequential Local Search (ISLS) Framework. Within this chapter, we present an analysis of the performance of simplified versions of the Hybrid Local Search (HLS) based on the embedded heuristics. This analysis revealed that simpler versions of HLS can outperform the original HLS itself. This analysis leads to a reevaluation of the fixed structure of HLS and to the design of the ISLS framework. This chapter includes a detailed explanation

of how, starting from HLS, we abstract layer by layer to obtain the structure of ISLS, and how the local search heuristics of HLS have been encapsulated within a generic local search framework. MH-builder, an ORKAD-developed platform, has been used to implement the ISLS framework and the problem-specific components. Highlights are given on specific code details and optimizations to demonstrate the implementation effectiveness.

Chapter 5 Chapter 5 focuses on the automatic configuration of the ISLS framework. It introduces automatic algorithm configuration (AAC) and explains how this work aligns with existing literature. Furthermore, it presents various widely-used AAC techniques, with a specific focus on irace, the configuration tool utilized in this thesis work. This chapter presents results on the automatic configuration of ISLS. This work used a simplified ISLS and shows the interest of the proposed framework. The automatic algorithm configuration technique manipulates ISLS configurations where the number of heuristics is the same as for HLS and where the heuristics are the same. However, the configurator can deactivate the heuristics and slightly modify their behavior. This chapter validates the earlier observations and provides, as the best ISLS configuration, a Great Deluge-based method where certain neighborhood operators are deactivated. This method consistently outperforms the original HLS and its variants. The investigation into ISLS, constrained to uphold the fixed HLS structure, reveals the necessity of structural flexibility, as individual components can surpass it. This performance assessment employs two methods: average rank per instance and rank based on average fitness. The chapter includes an ablation analysis, highlighting the significance of specific parameters in achieving a performance improvement between HLS and the proposed configuration.

Chapter 1

General Context

Contents

1.1 Introduction

In this chapter, we provide the foundation for the thesis work. The chapter begins by introducing the field of combinatorial problems, presenting their inherent challenges and distinctive characteristics. Then, the chapter presents the methodologies of the literature to efficiently solve combinatorial problems. A focus is made on metaheuristics and local search tenchniques.

Furthermore, the chapter turns its attention to the focal problem addressed in this thesis: the Curriculum-Based Course Timetabling problem. It begins by describing the problem, detailing unique attributes, and situating it within the literature on timetabling and scheduling problems. Concrete examples of algorithms are presented to facilitate the understanding of the problem intricacies. Additionally, the chapter provides an overview of the diverse datasets associated with this problem, highlighting their differences and origins.

Finally, the chapter concludes with a description of the solvers applied to address the Curriculum-Based Course Timetabling problem. These methods encompass various approaches, underscoring the multifaceted nature of problem-solving in this domain.

1.2 Combinatorial problems

Combinatorial optimization problems (COPs) belong to a category of optimization problems characterized by discrete decision variables and a finite search space. However, in most cases, the search space remains significantly vast, making an exhaustive search an impractical choice (Korte et al. [2011]).

The main objective of combinatorial optimization is to identify the best solution based on one or more criteria. These criteria are evaluated using a fitness or objective function, which assigns a score to each solution based on the specified criteria. Thus, the goal is to determine a solution that maximizes or minimizes this objective function, depending on the problem. This optimal solution is generally called *global optimum* and is defined, for the case of minimization, as $s^* \in S$ such that $\forall s \in S$, $f(s^*) \leq f(s)$, where S is the set of all existing solutions.

Combinatorial optimization problems are generally classified according to their complexity. The two main classes are P and NP. In complexity theory (Garey and Johnson [1979]), the class of problems P includes all decision problems that can be solved in polynomial time on a deterministic machine. The NP (Non-deterministic Polynomial) class of problems groups together decision problems that can be solved in polynomial time on a non-deterministic machine. Finally, the NP-hard class is a class of problems that includes some NP problems and other problems that are even more complex to solve. There are many examples of NP-hard combinatorial problems: the traveling salesman problem, the knapsack problem, the permutation flowshop scheduling problem, etc...

As an example, the Traveling Salesman Problem (TSP) consists of finding the shortest route for a salesman to visit all the cities only once and come back to his starting point. In the case of a graph model, the nodes are the cities and the routes connecting them are the edges. These edges are weighted based on the distance. The objective function of this problem refers to the distance covered by the salesman at the end of the journey. In TSP, the objective is thus to discover a Hamiltonian circuit that minimizes the total traveled distance. This problem is known to be NP-hard (Garey and Johnson [1979]). Additionally, the TSP search space is huge, there are a total of $\frac{(n-1)!}{2}$ potential solutions, where n represents the number of cities.

Another well-known NP-hard problem within the field of combinatorial optimization is the Knapsack problem (Martello and Toth [1990]). The Knapsack problem consists in the selection of

elements within the constraints of a weight limit. This weight limit sets the maximum capacity for the knapsack. The goal is to optimize the overall value of the chosen items while ensuring that the cumulative weight does not exceed the specified limit.

1.3 Solving Methods

Combinatorial optimization problems are certainly complex to solve, but there are many methods for solving them efficiently. This section presents the two main types of methods used in combinatorial optimization: exact methods and heuristics. More details are given about heuristics since they are at the center of the manuscript.

1.3.1 Exact methods

An exact method is a method that returns a global optimum. The advantage of this type of method is that we know the result is optimal. Generally, the main criticism directed towards exact methods centers on their potentially substantial time and computational resource consumption. This critique holds particularly true when tackling problems characterized by extensive instances and elevated complexity levels. The most basic and naive example of an exact method is the exhaustive search. This method lists all existing solutions of the search space, then evaluates them, and finally selects the best one or ones. This algorithm is hardly ever used, as the search space is very large in combinatorial problems. Exact resolution methods used in practice are more intelligent and limit the number of solutions to be evaluated. To do this, they use mathematical guarantees that the unexplored parts of the search space cannot contain the optimal global. As a result, these algorithms operate in a way that reduces the search space. Branch and bound algorithms are examples of such an approach (Lawler and Wood [1966]).

These methods consist of two primary components. The first step involves partitioning, where the algorithm divides the main problem into sub-problems. Each of these sub-problems has its own distinct set of solutions, all of which are valid solutions for the original problem. This partitioning process can be recursive, meaning that the sub-problems themselves become the basis for further sub-problems. The outcome is a tree structure, where the root represents the initial problem, and the leaves correspond to specific sub-problems. The branches on this tree represent different categories of problems, with nodes denoting each level of subdivision. The key objective is to ensure that by considering all these leaves collectively, we comprehensively address every aspect of the original problem. This comprehensive coverage is of utmost importance in this approach. The second step is the pruning: if reliable information indicates that the global optimum is not in a particular branch, then the algorithm does not evaluate the solutions in that branch. This is a very simplified summary of the algorithm.

Another noteworthy algorithm is the cutting plane method (Kelley [1960]). This approach involves representing the problem through a set of mathematical constraints. To simplify the problem, certain constraints are initially relaxed. However, in each iteration, the method readjusts these constraints to make progress toward the optimal solution. While the cutting plane method can be effective, it is not frequently used as a standalone method due to its relatively lower efficiency.

In addition to the cutting plane method, other exact methods are recognized in the field. One such method is the branch-and-cut technique (Padberg and Rinaldi [1991]), which combines cutting-plane and branch-and-bound methods to achieve improved results. Similarly, column generation is

employed, which also involves relaxing certain constraints to find an optimal solution. These advanced methods offer more sophisticated approaches to address complex combinatorial optimization problems effectively.

1.3.2 Heuristics

In combinatorial optimization, a heuristic is an approximation algorithm to find a good enough solution in a reasonable amount of time. They represent practical and often domain-specific strategies used to explore and potentially find good solutions for a given problem. In contrast, a metaheuristic is a strategic approach used at a higher level of abstraction to address optimization problems. Metaheuristics are applicable to different problems but may need to implement some problem-specific heuristics or components. There are a large number of very different types of metaheuristics using various strategies, see for example the recent reviews by Sorensen et al. [2017] and Hussain et al. [2019].

The advantage of an approximate method is that it calculates an approximate solution to a problem that is close to optimal in a relatively short time, compared with the time required by exact methods. In most applied cases, industrial applications require solutions that are easy to implement and easy to recalculate in the event of data changes. So an optimal solution that is difficult to achieve is not necessarily the goal if it involves too much computational cost.

The two best-known metaheuristics are probably Hill Climbing and Simulated Annealing, which are local search methods, sometimes referred to as neighborhood-based methods. This thesis focuses mainly on this type of metaheuristic. In the following sections, we define how a local search works in theory, and the neighborhoods it uses. In the end, we also present a type of algorithm called nature-inspired algorithm that includes popular methods like the genetic algorithm type, an evolutionary algorithm. It evolves a population of solutions using mutation and crossover mechanisms.

1.3.2.1 Local Search Methods

A local search is an approximate discrete solving method belonging to metaheuristics. Local search methods are called neighborhood-based methods because they traverse the search space, moving from a solution to a neighboring solution. These neighbors are solutions considered to be close to the current solution and reachable by the use of a neighborhood operator.

This type of algorithm aims to gradually improve the quality of its current solutions and stops when improvement is no longer possible/reachable or some termination criterion is reached. However, there are various strategies for selecting neighbors and accepting neighbors. Below, we give details on how a local search works.

Initialization Phase The initialization phase provides heuristics with a first, initial solution. The initialization phase can be a complex construction method to obtain a good initial solution or it can be a simple random generation algorithm. Local search manipulates a constructed initial solution and then explores the search space to improve it.

The construction phase plays a crucial role prior to the local search process, as many combinatorial problems are inherently challenging to solve, making it difficult to find a feasible solution. Constructors, also known as initializers, are responsible for creating the initial solution, which serves as the starting point for the optimization process.

Various construction methods are employed in combinatorial optimization. One such method is the greedy approach. It involves the step-by-step construction of an initial solution where the best choice at each step is selected, independently of whether it is the best choice in the long run.

Additionally, techniques derived from graph theory are widely used in construction. For instance, the combinatorial problem can be represented as a graph, where each node corresponds to an element and a vertex coloring technique can be employed to produce a feasible solution (Burke et al. [2007]).

Exploration phase A local search is an algorithm that explores different solutions in the search space to find a global optimum. Most local search algorithms follow the same procedure, repeating the same pattern iteratively. During each iteration, it explores new neighbors, and selects one to replace the current solution according to an acceptance criterion. Then, it continues this process until a termination criterion is reached. The local search uses neighborhood operators to define neighborhoods and generate solutions.

There are several local search methods. The best known are probably Hill Climbing, Simulated Annealing (Kirkpatrick et al. [1983]), and Tabu search (Glover and Laguna [1998]).

Algorithm 1 Hill Climbing

```
1: procedure HILL CLIMBING(startSolution)
2:     currentSolution ← startSolution
3:     while true do
4:         L ←NEIGHBORS(currentSolution)
5:         for all x in L do
6:             if EVAL(x) < EVAL(currentSolution) then
7:                 currentSolution ← x
8:             end if
9:         end for
10:    end while
11:    return currentSolution
12: end procedure
```

The Hill Climbing and Simulated Annealing algorithms in a minimization context are detailed in Algorithm 1 and Algorithm 2. Algorithm 1 illustrates the basic steps of Hill Climbing. This Hill Climbing employs the best improvement strategy, meaning it selects the best neighboring solutions from the current solution at each iteration, exhaustively exploring all possible neighbors. Algorithm 1 further demonstrates a version of Hill Climbing with strict acceptance criteria, where it only accepts neighbors that strictly improve upon the current solution.

Algorithm 2 details the Simulated Annealing (SA) algorithm. SA differs from Hill Climbing in that it is willing to accept solutions that do not improve the current solution. It employs the Metropolis metric, which considers the fitness difference and the current temperature to decide whether or not to accept a non-improving solution. An important parameter in Simulated Annealing is the temperature, which is initially set and gradually decreases with each iteration according to a cooling rate. The acceptance of non-improving solutions becomes less likely as the temperature decreases. The SA process concludes when the temperature reaches zero, indicating convergence.

The next paragraphs explain the notions of neighborhoods, operators, and landscapes in order to define the concepts relevant to this manuscript.

Algorithm 2 Simulated Annealing

```
1: procedure SIMULATED ANNEALING(startSolution, initialTemperature, coolingRate)
2:     currentSolution ← startSolution
3:     currentTemperature ← initialTemperature
4:     while currentTemperature > 0 do
5:         neighbor ← GenerateRandomNeighbor(currentSolution)
6:         Δ ← EVAL(neighbor)− EVAL(currentSolution)
7:         if Δ ≤ 0 or Random(0, 1) < e^(−Δ/currentTemperature) then
8:             currentSolution ← neighbor
9:         end if
10:        currentTemperature ← currentTemperature · coolingRate
11:    end while
12:    return currentSolution
13: end procedure
```

Neighborhood In the context of combinatorial optimization, the size of the search space is a function of the number of elements involved in solving a problem. The greater the number, the larger the search space.

The neighborhood relation serves as a way of connecting solutions within this search space. Local search methods commonly utilize neighborhood operators, which are algorithms that make slight modifications to a solution, granting access to neighboring solutions.

Formally, the neighborhood relation is represented as: $N : S \to 2^S$, where S denotes the set of all feasible solutions, and N assigns to each solution a space called the neighborhood, comprising a subset of feasible solutions considered to be its neighbors.

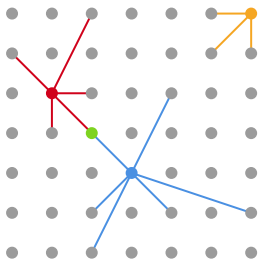


Figure 1.1: Visualization of the neighborhood concept: each point represents a feasible solution. The edges correspond to a neighborhood relation between the two nodes. There are three neighborhoods, of the red, of the blue, and of the orange solution. The green solution is part of two neighborhoods at the same time.

Figure 1.1 provides a simplified partial representation of a search space, illustrating a set of feasible solutions. This figure visualizes the concept of neighborhoods: the red solution is linked to other solutions by edges that represent a neighborhood relation, and the same applies to the blue and orange solutions. Additionally, Figure 1.1 demonstrates that it is possible to be part of several neighborhoods simultaneously, as seen with the green point.

Neighborhood Operator A neighborhood operator is an algorithm that helps the neighborhood-based method to go from the current solution to a neighbor solution. So a neighborhood relation between two solutions exists if there is a neighborhood operator capable of switching from one to the other. The neighborhood of a solution depends entirely on which neighborhood operators the local search uses. Local search methods may employ several operators, each specifically tailored to the problem at hand.

Among the different types of operators for the TSP, there is the 2-opt. The concept is simple: the operator reverses the order of passage between two nodes and picks the best such modification.

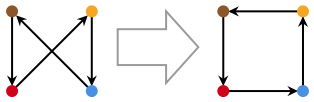


Figure 1.2: Example applying the 2-opt neighborhood operator on a TSP problem. Each node is a city. The starting city is the red point. Arrows represent the path taken by the salesman.

Figure 1.2 shows a 4-city TSP problem where the 2-opt has changed the order of the travel. At the beginning, it was:

$$red \rightarrow yellow \rightarrow blue \rightarrow brown \rightarrow red$$

The 2-opt chose the nodes red and brown, so the order between them is reversed. The result is consequently:

$$red \rightarrow blue \rightarrow yellow \rightarrow brown \rightarrow red$$

Local optimum When exploring solutions, local search methods very often encounter local optima. A local optimum can be defined as a solution $s^* \in S$ such that $\forall s \in N(s^*)$, $f(s^*) \leq f(s)$. Minimization is considered here. Thus, a local optimum is defined in relation to its neighboring solutions, unlike global optima, which are defined as having no strictly better solution in the whole search space (Section 1.2). A plateau is a set of pairwise neighboring solutions with the same fitness value.

Figure 1.3 illustrates local optima, a global optimum, and a plateau in a 2D-representation of search space.

1.3.2.2 Nature-Inspired Algorithm

A nature-inspired algorithm is a computational technique that emulates principles or behaviors observed in natural systems to solve complex problems. There are various nature-inspired algorithms available, such as genetic and evolutionary algorithms (GA) (Goldberg and Holland [1988]), particle swarm optimization (PSO) (Kennedy and Eberhart [1995]), ant colony optimization (ACO) (Dorigo and Di Caro [1999]), and genetic programming (GP) (Koza [1994]).

PSO simulates particles adjusting positions based on their experiences and neighbors to find optimal solutions. ACO emulates ant foraging behavior, using pheromones to find the best paths in a search space. Genetic programming is based on evolving computer programs or expressions to discover adaptable solutions for various problems.

An evolutionary algorithm is a nature-inspired optimization technique that uses principles of natural selection, mutation, and recombination to evolve a population of potential solutions. A

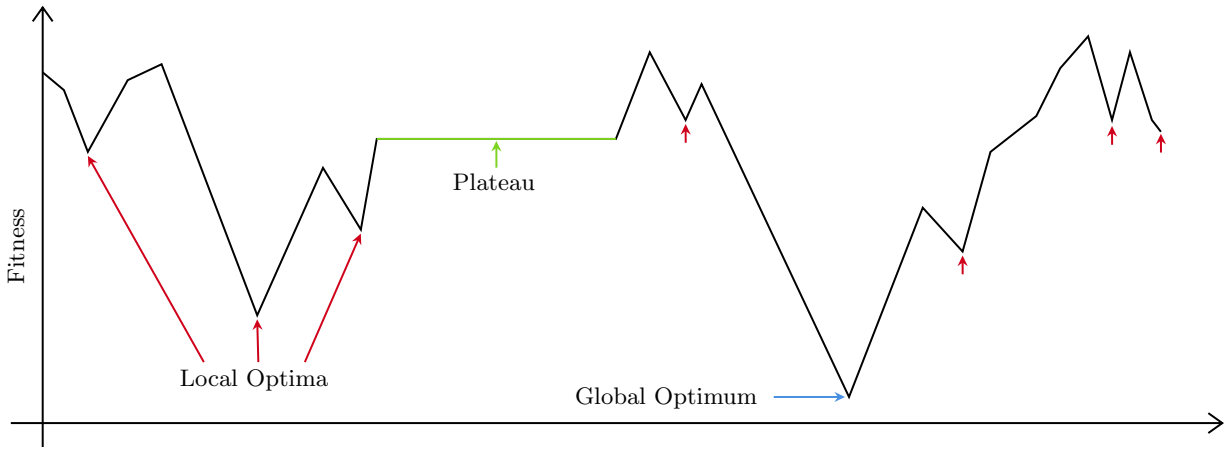


Figure 1.3: Example of Local Optima, Global Optimum, and a Plateau. All red arrows indicate a local optimum.

genetic algorithm (GA) is a specific type of evolutionary algorithm that uses genetic operators like mutation and crossover to evolve a population of candidate solutions.

Genetic Algorithm Genetic algorithm is the most popular evolutionary algorithm which draws inspiration from the evolution of the genome.

The algorithm is modeled on the evolution of a population over generations. That implies that the fittest individuals, according to the value of the objective function, survive beyond generations.

Furthermore, as time progresses, the fittest individuals within the population cross-breed, giving rise to a new generation that inherits traits from both parents. This process, known as crossover in genetic algorithms, involves taking two solutions and combining them to produce children who inherit parts of each parent solution. To ensure that all children are still feasible and adhere to constraints when accepting a solution, it is often necessary to apply a repair algorithm. For instance, in the context of the Traveling Salesman Problem (TSP), a child may adopt the path taken by the salesman for the first 4 towns from one parent and the remaining path from the other parent. This specific type of crossover is referred to as a single-point crossover. Figure 1.4 shows that the repairing algorithm added new arrows to make it work.

Similar to the theory of evolution, genetic algorithms incorporate the concept of mutation. Mutations occur randomly in DNA, contributing to the population diversification. In the genetic algorithm, mutations are introduced to enable new variations along with crossover. Mutation typically affects only a relatively small percentage of the new generation and is typically minimal in magnitude. The parallel can be drawn with local search: mutation can correspond to selecting randomly a neighboring solution.

By incorporating crossover and mutation, the genetic algorithm explores and exploits several potential solutions, gradually converging towards improved solutions over successive generations.

The generic Genetic Algorithm loops through the steps listed below.

1. Initial Generation: Generate an initial population using a construction algorithm.

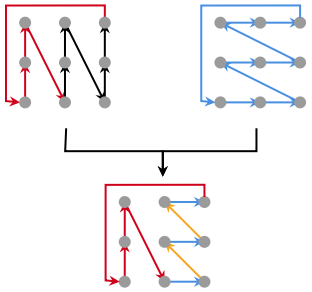


Figure 1.4: Crossover Process. Red Arrows are the ones chosen to be inherited first. Then it adds the blue ones if it is possible. Orange ones are the ones added.

2. Evaluation: Each solution is evaluated using the objective function.
3. Selection: The selection process is employed to keep the most promising solutions, ideally maintaining some diversity in the population.
4. Crossover and Repair: The crossover is applied first, creating a new generation with new solutions. The population then has the selected solutions and the new ones generated and repaired by a repair function.
5. Mutation: The mutation is applied to solutions according to a predetermined probability. This operator slightly modifies solutions to diversify the population. The probability of mutation can vary based on the problem being addressed.
6. Repeat: The algorithm returns to the selection and repeats the steps iteratively until the allotted time or stop criterion is reached.

The genetic algorithm often requires many generations to converge. In our work, we have not used this type of algorithm because we decided to focus on local search during this thesis. However, some methods of this type are effective on the problem addressed.

1.4 Curriculum-Based Course Timetabling

This section discusses the Curriculum-Based Course Timetabling problem. We explain the specific details of this problem, including its constraints and its objective function, while also situating it within the existing literature.

Additionally, we provide information about the state-of-the-art solving method that is the basis for much of the work in this thesis. This includes an explanation of the process, heuristics, and strategies used by the solver. Lastly, we present the benchmark used in this thesis. Our aim is to provide all the necessary information to understand this thesis work, including the problem, the solving method, and benchmarks.

1.4.1 Problem

This thesis focuses on a problem called Curriculum-Based Course Timetabling, abbreviated as CB-CTT. This is a difficult problem categorized as NP-Hard (Burke et al. [2010b]). In practice,

solving the Curriculum-Based Course Timetabling problem cannot be done in polynomial time. Curriculum-Based Course Timetabling has been widely studied in the literature (Abdullah and Turabieh [2012], Lü and Hao [2010], Bellio et al. [2013], Cacchiani et al. [2013]). Many different solving methods have been developed, and research is still being conducted on this problem. CB-CTT remains an important problem to solve because of scientific interest and out of practical interest for better timetables for institutions, staff, and students.

The history of CB-CTT starts with an event: the International Timetabling Competition in 2007 (ITC 2007). The latter is a competition organized by PATAT, International Conference on the Practice and Theory of Automated Timetabling. ITC aims to stimulate research around timetabling problems. ITC 2002, the first edition, focused on only one timetabling problem. The second edition, ITC 2007, had three different tracks, i.e. three timetabling problems with their specificities each. The first one is an Examination Timetabling Problem, while the second is referred to as the Post-Enrollment-based Course Problem. In this latter problem, students submit a list of their desired courses, and the timetable is then created based on these selections. There is no pre-made package of lectures used in this one. The third track concerned the Curriculum-Based Course Timetabling.

ITC 2007 formalized the CB-CTT problem and thus imposed specificities, constraint weights, objectives to optimize, etc. The following sections focus on the aspects and specificities of the problem that must be respected to solve it according to the rules set.

In order to approach more easily what the problem consists of, it is necessary to situate CB-CTT in the field of timetabling problems.

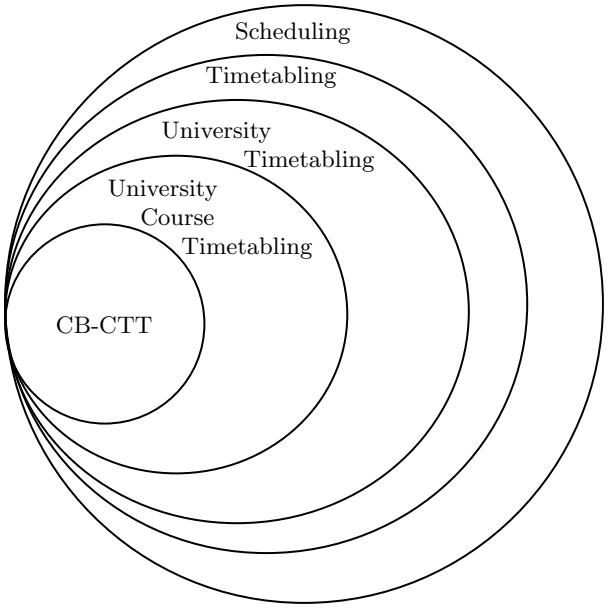


Figure 1.5: Hierarchy of University Course Timetabling.

Figure 1.5 shows how to situate CB-CTT in the plethora of scheduling and timetabling problems (Schaerf [1999]). Indeed, Wren [1995] demonstrated in 1996 that timetabling problems are complex scheduling problems.

1.4.1.1 Definition

The Curriculum-Based Course Timetabling is basically a complex problem of assigning resources to events. In this context of university timetabling, the events are class hours, called lectures. All lectures last the same amount of time, usually one hour. There is no way in CB-CTT to force explicitly two specific lectures to follow each other, to do two hours in a row for example. Lectures are grouped by course. The notion of course comes from the University Course Timetabling, or UCTT, problem class, Figure 1.5 highlights this legacy. UCTT is a problem type where lectures are grouped by subject to stick as close as possible to concrete problems. Each course has one teacher who teaches each lecture of that course. Thus, a course corresponds to a set of lectures taught by a single teacher, for example, data science for group A taught by Mr. Stevenson. If group B attends the data science course with another teacher, then it is a different course, because a course is defined by its teacher.

An important notion that diverges from common UCTT problems is the curriculum, a package of courses. A Curriculum-Based Course Timetabling problem has several curricula that include courses. A curriculum represents a group of students where each individual attends the same courses. A student can only enroll in one curriculum. However, it is acceptable for the same course to be included in two or more curricula. In this case, all students in the curricula have classes at the same time. This is a shared course. Teachers can perform an unlimited number of lectures and have no maximum number of hours. In a CB-CTT problem, the teachers are linked to courses in the statement. That means that the algorithm does not have the flexibility to change teachers if they are not available. This represents an additional constraint and increases the difficulty. Figure 1.6 shows a graph representation of a simple CB-CTT problem. For example, course 1 is attended by students of curricula 1 and 2. Each instance of CB-CTT includes a list of courses with the number of lectures, the teacher linked, and curricula.

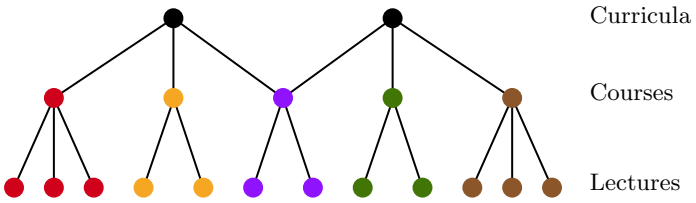


Figure 1.6: Graph representation of a CB-CTT example problem. Each color corresponds to the teacher in charge. Each edge means a link of belonging.

Now that we have presented the layout and structures of the events or lectures, we must add and detail the resources to be planned in the CB-CTT problem. The problem consists in assigning a timeslot and a room for each lecture. Each room has a capacity, which is the maximum number of students it can accommodate. Curriculum-Based Course Timetabling is a weekly problem. Therefore, it is divided into days which are themselves divided into timeslots. One timeslot often corresponds to one hour. CB-CTT has an unavailability mechanism. That means the problem offers the possibility to the teachers, and only them, to have unavailable timeslots. Consequently, teachers have certain timeslots where they cannot teach because they have unavailabilities. The problem neglects the possibility of room unavailability for other reasons than those used by lectures.

The solution representation to a CB-CTT problem can be considered as a matrix where each cell is either empty or filled by an event. Each row corresponds to one room, and the columns are

timeslots. Figure 1.7 shows a representation of a solved toy problem. Each color is a course, there is no curriculum to simplify. Names correspond to the name of the course.

	Day 0				Day 1			
Room	t0	t1	t2	t3	t4	t5	t6	t7
A	Data				Com			
B			Com				Data	
C			Prog	Data	Prog			
D	SQL			SQL		Prog		

Figure 1.7: Example of a solution for a CB-CTT toy problem. Each color and name represents a course: Data Science, Communication, SQL database, and Programming. Each row corresponds to a room and each column to a timeslot. Timeslots are ordered chronologically through the week.

1.4.2 Constraints

In Curriculum-Based Course Timetabling, a solution consists of scheduling lectures in timeslots and available rooms following hard and soft constraints. These constraints have been proposed by ITC 2007 (Di Gaspero and Schaerf [2006]).

1.4.2.1 Hard

The first type of constraint is called a hard constraint, and each hard constraint must be respected. These rules define whether a solution is feasible or not. The ITC 2007 has set four hard constraints that ensure the schedule is feasible.

The four hard constraints for CB-CTT are listed below.

1. **Conflicts:** Two lectures from the same course or the same curriculum or with the same teacher cannot take place at the same time.
2. **Room occupancy:** Only one lecture can take place at a time in a room.
3. **Availability:** The lectures of a teacher must be scheduled when he is available.
4. **Lectures:** All lectures must be scheduled.

The last hard constraint prevents getting a partial solution. The others focus on avoiding aberrant conflicts. For example, they prevent a group of students from having two lectures at the same time, a room having two different lectures at the same time, and teachers teaching when they are not there or two lectures at the same time. To summarize, a timetable is said to be feasible when all the hard constraints are met.

1.4.2.2 Soft

Soft constraints represent the second type of constraints. Contrary to hard constraints, they are optional and generally represent targets to strive for. The violations of each soft constraint are represented as a function to minimize. Thus, soft constraints represent the desired aspects of the

solution. The more the soft constraints are respected, the better the timetable is. The four soft constraints are as follows.

1. *RoomCapacity*: One violation for each student without a seat during one lecture.
2. *MinWorkingDays*: A course has lectures that should be scheduled within a minimum number of days in order to avoid students having all lectures of one course over one day, which would be inconvenient. The number of violations corresponds to the difference between the given minimum number and the real one.
3. *CurriculumCompactness*: Within each curriculum, lectures on the same day should be consecutive, more precisely if there is more than one lecture of one curriculum on the same day, it is preferable for them to immediately precede or follow one another. One violation is counted for each isolated lecture, which means a lecture without neighbor lectures within the same curriculum.
4. *RoomStability*: Lectures of a course should be in the same room; one violation for every extra room used.

Regarding the MinWorkingDays constraint, each course has a predefined value of MinWorkingDays given in the problem file. Interestingly, RoomCapacity is a soft constraint. That implies solutions can have lectures in rooms too small to accommodate all the students. It is a CB-CTT specificity, since, for example, Post Enrollment-based Course Timetabling, which was one of the three tracks in the ITC 2007, considers the number of seats as a hard constraint to be respected. This choice increases the number of feasible solutions to the same problem and offers algorithms an additional degree of freedom. The latter can, for example, schedule a lecture to avoid a gap in the timetable, but a place is missing in the new room.

While hard constraints must be obeyed, soft constraints may be violated and only indicate some preferred outcome. The fewer the violations, the better the timetable. This is the baseline for the objective function that drives the optimization process.

1.4.3 Objective function

For a given CB-CTT problem, there may exist plenty of feasible solutions that respect hard constraints. The objective is to get a timetable that respects the soft constraints as much as possible, thus minimizing the violations. Recall that CB-CTT is a single-objective problem, thus, with only one objective function. For CB-CTT, ITC 2007 chose to consider aggregation of the soft constraints corresponding to the weighted sum of the violation of each soft constraint:

$$\begin{aligned} Fitness(s) =& RoomCapacity(s) * \omega_{rc} && (1.1)\\ &+ MinWorkingDays(s) * \omega_{mw} && (1.2)\\ &+ CurriculumCompactness(s) * \omega_{cc} && (1.3)\\ &+ RoomStability(s) * \omega_{rs} && (1.4)\\ &\text{where } s \text{ is a Timetable} \end{aligned}$$

Thus, the CB-CTT objective function to optimize is a weighted sum of the constraint violations where s represents a timetable. The weights, as used for ITC 2007, are set to 1, 5, 2, and 1. The weights are different because the violations are not computed in the same way and have

different domains. For example, $RoomCapacity(s)$ returns the number of missing seats by lecture. Consequently, the returned value is high. On the contrary, $MinWorkingDays(s)$ returns the difference between the desired number of days and the actual number for each run. However, the problem is weekly, so the value returned by the course can not exceed 4.

Consequently, CB-CTT is a single-objective problem, because solvers only optimize one objective function. However, it is relevant to understand that initially, the CB-CTT problem is multi-objective where each objective function corresponds to one soft constraint. However, the choice has been made to aggregate all soft constraints into a single fitness function. That lets us use methods designed for single-objective problems.

1.5 Benchmarks

Benchmarks offer the opportunity to evaluate the performance of different algorithms on instance sets. In this thesis, we use three groups of instances. This section explains the origin of the instances.

1.5.1 ITC

The most used benchmark in the literature about Curriculum-Based Course Timetabling was provided at the ITC 2007 and is called *ITC* in this manuscript. It is composed of various CB-CTT instances used to compare the solvers for the competition. This benchmark includes only 21 instances, it is thus a small set of instances. These problem instances come from real-world problems encountered by the University of Udine, Italy. Despite its small size, the *ITC* set is the most commonly used in papers about CB-CTT until recently. Our work primarily used this set because it was the only one available until 2022.

1.5.2 New Real-world

The benchmark called *New Real-world* is an additional set of instances proposed by De Coster et al. [2022]. This team gathered 61 new real-world CB-CTT instances, from different institutions. For example, there are new problems from Udine and Erlangen Universities. Some instances of this benchmark are very different from *ITC* because the number of lectures to schedule is much higher. Thus, *New Real-world* diversifies the types of problems encountered. New real-world instances offer the opportunity to use machine learning to develop smarter solving methods as the number of instances becomes larger.

1.5.3 Artificial

The last set of instances also comes from De Coster et al. [2022]. They have been automatically generated. Chapter 3 details how they have been generated. The main idea consists in mimicking the two previous sets. In any case, this set includes 6942 different problems. That amount of instances represents the main benefit of the artificial instance automatic generation because it offers such a large amount of data. Chapter 3 is devoted to the analysis of these benchmarks in order to verify that they are usable for our work.

1.6 CB-CTT Solving Methods

This section introduces two solving methods for the Curriculum-Based Course Timetabling problem. The first method is a hybrid approach, primarily based on a genetic algorithm. The second method is a metaheuristic that involves three local search algorithms, and it is used throughout this thesis.

The objective here is to demonstrate that there exist various approaches and strategies for effectively solving CB-CTT instances. Each method offers unique characteristics and can be adapted to tackle different challenges within the problem domain.

1.6.1 Tabu-based Memetic Algorithm

The Tabu-based Memetic Algorithm was proposed by Abdullah and Turabieh [2012] and designed to solve the Curriculum-Based Course Timetabling problem.

First of all, memetic algorithms can be described as population-based metaheuristics that are composed of an evolutionary system and include a local search algorithm, which are executed during the cycle of the framework (Neri and Cotta [2012]).

The Tabu-based Memetic Algorithm has demonstrated high efficiency to solve ITC instances. The primary objective of this section is to present how this method works, highlighting the various approaches to solving diverse problems.

This method involves employing the mechanisms found in population-based methods, specifically genetic algorithms (Section 1.3.2.2). These mechanisms include mutation and crossover operators. However, the method also incorporates a tabu search mechanism that enables the focusing of the search in a specific region of the search space, promoting intensification.

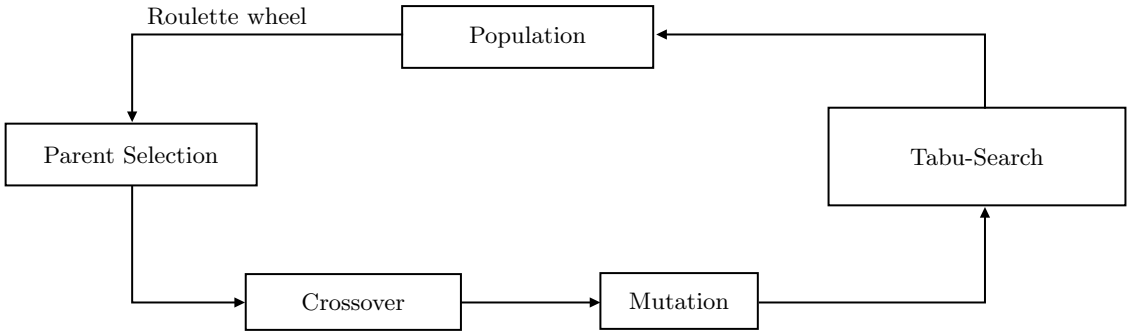


Figure 1.8: Composition of Tabu-based memetic algorithm.

Crossover The crossover involves taking two parent solutions and generating one or more child solutions, it is adapted from an other work (Cheong et al. [2009]). This particular method uses two parents to create two children.

The crossover operator initially copies parent 1 to create child 1. Then, in parent 2, it identifies a fixed timeslot and determines which lectures are scheduled there and in which room. Subsequently, the lectures scheduled for this timeslot in child 1 are removed, and the corresponding timeslot from parent 2 is inserted in their place.

To ensure that all hard constraints are still satisfied, a repair function is applied to Child 1, which checks for any violations and resolves any instances of double-scheduled lectures. The same steps are then performed for child 2, but with parent 1 and parent 2 reversed.

The advantage of this crossover method is its fast execution speed. Unlike crossovers that divide and merge jobs, which often require time-consuming repair algorithms iterating through all possibilities, this approach significantly reduces processing time.

Mutation In this method, the mutation phase involves randomly selecting a neighborhood structure and applying it to the current solution. Then, the algorithm accepts a new solution regardless of its quality. Abdullah and Turabieh [2012] uses neighborhood structures that define a way of changing the solution. These neighborhood structures are detailed later.

Tabu Search In this local search algorithm, during the first iteration, a neighborhood operator is randomly selected from all available neighborhood structures. In subsequent iterations, the choice of the neighborhood operator depends on the outcome of the previous iteration. If the selected neighborhood led to an improving neighbor, that particular operator is saved and reused in the next iteration to continue exploiting its promising results. However, if the operator generates a poor-quality solution, it is not accepted, and the operator is placed in a tabu list to avoid reusing it immediately. In the next iteration, a random neighbor is selected from the operators that are not present in the tabu list.

To prevent the tabu list from growing excessively, its size is limited to be smaller than the number of available neighborhood structures. If the list becomes full, the First-In-First-Out (FIFO) rule is employed to free up space by removing the oldest elements from the list.

So in this algorithm, only strictly better solutions are accepted. It considers that as long as a structure allows the solutions to be improved, we keep it. Otherwise, we change it. Tabu search is applied to each new solution after the crossover and mutation phases.

1.6.1.1 Neighborhood structures

The mutation operator and the tabu search both use neighborhood structures as named by Abdullah and Turabieh [2012]. They are listed below:

- Nbs1: Select two events at random and swap timeslots.
- Nbs2: Choose a single event at random and move to a new random feasible timeslot.
- Nbs3: Randomly select two timeslots. Then, perform a simple swap, exchanging all the events in one timeslot with all the events in the other timeslot.
- Nbs4: Randomly select two timeslots, denoted as t_i and t_j (where $j > i$), from an ordered set of timeslots $t_1, t_2, t_3, \ldots, t_p$. First, reallocate all events from timeslot t_i to timeslot t_j. Next, move the events that were originally in t_{j-1} to t_{j-2}, and continue this process iteratively until all events that initially belonged to t_{j+1} are allocated to t_i.
- Nbs5: Select a random 10% subset of events, and from this subset, identify the event with the highest penalty. Then, move this high-penalty event to a randomly selected feasible timeslot.
- Nbs6: Carry out the same process as in Nbs5 but with 20% of the events.

- Nbs7: Move the highest penalty event from a random 10% selection of the events to a new feasible timeslot that can generate the lowest penalty cost.
- Nbs8: Carry out the same process as in Nbs7 but with 20% of the events.
- Nbs9: Select two timeslots, t_i and t_j, based on the maximum number of enrolled lectures. Next, identify the most conflicted lecture in t_i and t_j, and then apply a Kempe chain, as proposed by Thompson and Dowsland [1996].

The first three neighborhood structures consist of simple kick operators that only move a single element.

The objective of Nbs4 is to shift a sub-part of the timetable by one day, effectively moving all events scheduled for Thursday to Wednesday.

Nbs5-8 are operators that analyze a sub-part of the problem and then move the event that presents the most penalty cost, making them simple guided kicks. Nbs5-6 perform random moves, while Nbs7-8 attempt to minimize the impact of the move on the overall schedule. All Nbs5-8 operators exclusively change timeslots.

Finally, Nbs9 relocates a set of scheduled events connected by hard constraints. This last neighborhood structure is more complex because it utilizes the Kempe chain. We now explain the principles of these operators as applied to our specific problem. Kempe chain has been proposed to be used on timetabling problems by Thompson and Dowsland [1996] and Burke et al. [2010a].

The variation in the number of neighborhood structures offers an intriguing perspective on the diverse methods for altering a solution. In this thesis, we did not employ these structures. Nonetheless, the Kempe chain process represents one of the directions for exploration with more sophisticated neighborhood operators.

Kempe chain operator The concept of a Kempe chain comes from the graph coloring problem. However, CB-CTT uses it as a neighborhood operator.

A Kempe chain is a chain that includes all connected nodes in a graph. In CB-CTT, nodes are lectures, and the edges represent conflicts arising from hard constraints. Specifically, two nodes are connected if the two lectures cannot be scheduled at the same time. That occurs when the lectures belong to the same curriculum or course, or have the same teacher.

Representing all kempe chains of an entire timetable would be a complex task and not useful for later. So, Kempe Chains neighborhood operator focuses on two timeslots and one lecture. To apply the Kempe chain operator: a starting lecture is fixed, and the corresponding graph represents only the Kempe chain of events scheduled in the two selected timeslots.

After repairing the Kempe chain, a simple swap of timeslots is performed for the events included in the chain. This swap is swift because most constraints have already been checked, and it allows multiple events to be moved simultaneously. The remaining constraints are the required number of rooms and teacher unavailability.

Overall, using Kempe chains as a neighborhood operator offers the advantage of fast swap operations while maintaining adherence to critical constraints.

1.6.2 The Hybrid Local Search

This section focuses on the state-of-the-art solver for Curriculum-Based Course Timetabling (Müller [2009]). This solver is used in each chapter of this thesis and it is thus necessary to know its design

and the details of its components. After a brief presentation of this solver, a deep description is provided.

1.6.2.1 Context

The literature on CB-CTT problem encompasses a wide range of methods aimed at its resolution, spanning from genetic algorithms to heuristics. Among these methods, hybrid approaches have emerged as particularly effective, surpassing other techniques in performance. Notably, Müller [2009] developed this solver for the ITC 2007 competition and emerged as the winner of CB-CTT track of the competition.

This solver consists of two main components. The first part involves the construction phase, where an algorithm utilizes graph coloring techniques described by Müller [2005] to generate an initial solution. The second part focuses on an optimization algorithm that iteratively refines the solutions, and returns improving solutions in terms of fitness. This thesis focuses on the optimization of a solution and not construction.

The optimization method employed in this solver is called Hybrid Local Search (HLS) in this thesis, which derives from the utilization of three local search algorithms operating successively in a loop. HLS has demonstrated its efficacy specifically in addressing the CB-CTT problem. Furthermore, the solver is partially open-source, facilitating a better understanding of its inner workings. The developer of HLS continues to update the solver, and it has also found applications in other timetabling problems, including the ITC 2019 competition. The name of the solver currently developed is Unitime[1], an Apereo Sponsored Project[2]. This versatility offers promising prospects for employing the solver in tackling novel timetabling challenges.

Our work strategically uses the version of HLS that won the ITC 2007 competition. In addition to the availability of more recent versions, we deliberately rely on this version, highlighting its efficiency and robust performance. Finally, its open-source code offers an easier and clearer understanding of all the mechanisms implemented.

1.6.2.2 Neighborhood Operators

Local search algorithms operate by iteratively moving from an initial solution to subsequent solutions, exploring the search space based on a defined neighborhood relation. The Hybrid Local Search method, specifically, incorporates six distinct neighborhood operators. In this thesis, we have made the deliberate choice to retain these six operators, as they are widely recognized and have been extensively studied by practitioners and researchers in the field (Song et al. [2021]).

The neighborhoods considered are listed below, and it is worth noting that some of them may be referred to by different names in existing literature.

Time Move: A lecture and a timeslot are selected. The lecture is assigned to the timeslot.

Room Move: A lecture and a room are selected. The lecture is assigned to the room.

Lecture Room Move or Lecture Move: A lecture, a room, and a timeslot are selected. The lecture is assigned to the room and timeslot. If the room is already used on this timeslot, the timeslot and room of the selected lecture are swapped with those of the conflicting lecture.

[1] See more details in https://www.unitime.org/
[2] https://www.apereo.org/projects/unitime

Room Stability Move: A course and a room are selected, then each lecture of the course is assigned to the room. If the room is not available, the lectures of the course swap rooms with the conflicting lectures.

MinWorkingDays Move: A course with a MinWorkingDays violation is selected. Lectures for that course, on days with more than one lecture, are moved to another timeslot to minimize MinWorkingDays violations. The room can be changed if it is already used by an other lecture.

CurriculumCompactness Move: A curriculum with a CurriculumCompactness penalty is selected. An isolated lecture within the curriculum is assigned to a different timeslot so that it becomes adjacent to other curriculum lectures. Similar to other moves, the room can be changed if it is not available.

These six operators represent various strategies or operations used within the timetabling problem to enhance the overall solution. They can be divided into two groups: the random operators include the Time Move, Room Move, and Lecture Move, while the remaining three belong to the specific group dealing with soft constraints. That is because they create a neighbor that aims to improve the solution by reducing the violation of a particular soft constraint.

1.6.2.3 Heuristics of Hybrid Local Search

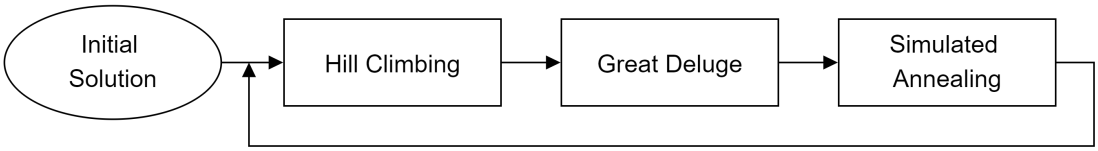


Figure 1.9: Composition of the HLS method.

The Hybrid Local Search (HLS) consists of three distinct local search heuristics (Hill Climbing, Great Deluge, and Simulated Annealing) executed sequentially and iteratively, as illustrated in Figure 1.9. To ensure efficient execution, a fixed time limit of 5 minutes, as defined in the ITC 2007 literature, is set as the termination criterion. Subsequent work has used this 5-minute limit even with more powerful computers, effectively increasing the original computational budget nowadays (De Coster et al. [2022], Song et al. [2021]). HLS ends immediately when the time limit is reached, without waiting for the current local search iteration to conclude naturally. Additionally, HLS can terminate its execution upon achieving a perfect solution that adheres to all soft constraint violations, resulting in a fitness value of zero. In the subsequent sections, we provide concise descriptions of each heuristic employed within the HLS framework, highlighting their functioning and distinctive characteristics.

Hill Climbing Hill Climbing is the best-known local search algorithm in the literature (Skalak [1994]). It is also the simplest to implement. Hill Climbing is an iterative method that generates a new solution by applying a neighborhood operator. If this new solution is better than the previous one, then it becomes the new current solution. Hill Climbing in Hybrid Local Search has a

specificity: it accepts any neighbor that has an equal or better score, this criterion is called neutral acceptance. The choice is motivated by the recurring neutrality of timetabling problems, which results in a large number of plateaus (Section 1.3.2.1). Thus, accepting equivalent solutions helps to increase diversity and may lead to improvement.

Given the acceptance criterion and the high neutrality of the CB-CTT problem, HC has a special ending criterion. In the literature, HC usually stops when there is no strictly local improving neighbor. CB-CTT is very neutral, which means it has a lot of connected solutions for the same score value. Thus, an HC with neutral acceptance would run in a loop because it would accept all these equivalent neighbors. The termination criterion uses a fixed number of explored solutions. This maximum number is fixed at 50,000 and is called $maxIdle$. That means Hill Climbing ends if it does not find a global improving solution after 50,000 neighbors explored consecutively.

Algorithm 3 HLS Hill Climbing

```
1: procedure HILL CLIMBING(Timetable)
2:     Current ← Timetable
3:     MaxIdle ← 50,000
4:     IterWithoutImprovment ← 0
5:     while IterWithoutImprovment < MaxIdle do
6:         Op ← Random(Ops)
7:         Neighbor ← Neighbors(Op, Current)
8:         if EVAL(Neighbor) ≤ EVAL(Current) then
9:             Current ← Neighbor
10:            if EVAL(Current) ≤ EVAL(BestFitnessHLS) then
11:                IterWithoutImprovment ← −1
12:            end if
13:        end if
14:        IterWithoutImprovment + 1
15:    end while
16:    Return Current.
17: end procedure
```

Algorithm 3 corresponds to the pseudo-code of the Hill Climbing used in Hybrid Local Search and begins with an existing solution. Then, the process continues until the number of iterations without strict improvement is reached comparing with the best solution found during the HLS run, not the previous current solution as commonly. In Algorithm 3, one iteration corresponds to one neighbor evaluated.

During each iteration, the Hill Climbing algorithm selects a neighborhood operator, Op among all and uses it to generate a new neighbor solution. The quality of this new solution, evaluated using the objective function (referred to as EVAL), is compared to the score of the current solution.

If the neighbor solution is better than the current one, it replaces the current solution. On the other hand, if the neighbor solution is not better, the current solution remains unchanged, If the new current solution is better than the best solution found by HLS, then the iteration count without improvement is reset to -1. The count of iterations without improvement is always incremented by one.

Great Deluge The Great Deluge algorithm is a local search method originally proposed by Dueck [1993]. In the literature, it is often considered as an alternative to Simulated Annealing, as it shares the characteristic of accepting locally non-improving solutions.

The Great Deluge algorithm sets an initial Bound value that is greater than the fitness of the best solution found during the execution of the Hybrid Local Search algorithm. Throughout the execution, only solutions with a score strictly lower than Bound are accepted as the current solution. The Bound value progressively decreases at each iteration. The execution of the Great Deluge algorithm concludes when the Bound reaches a precomputed value called Lower Bound, determined by the overall best fitness.

As a result, the Great Deluge algorithm gradually accepts fewer deteriorating solutions as the Bound value decreases. Additionally, if the algorithm discovers a new best solution, the lower bound value is adjusted accordingly, which extends the exploration process.

Algorithm 4 is a pseudocode representation of the Great Deluge algorithm in the Hybrid Local Search. Parameter values have been tuned by Müller for ITC 2007.

Algorithm 4 HLS Great Deluge

```
1: procedure GREAT DELUGE(Timetable)
2:     Current ← Timetable
3:     UpperBoundRate ← 1.15
4:     LowerBoundRate ← 0.9
5:     CoolingRate ← 0.9999999999874
6:     At ← HLSInformationValue
7:     if First Execution in HLS then
8:         Bound ← BestFitnessFound * UpperBoundRate
9:     end if
10:    while Bound > BestFitnessFound * LowerBoundRate^at do
11:        Op ← Random(Ops)
12:        Neighbor ← Neighbors(Op, Current)
13:        if EVAL(Neighbor) ≤ Bound then
14:            Current ← neighbor
15:        end if
16:        Bound ← Bound * CoolingRate
17:    end while
18:    Bound ← BestFitnessFound * UpperBoundRate^at
19:    Return Current.
20: end procedure
```

Simulated Annealing Simulated Annealing is a widely recognized and effective method of local search, initially proposed by researchers at IBM (Kirkpatrick et al. [1983]). This metaheuristic took inspiration from the process of metallurgy, specifically the technique of cooling a solid material to achieve a state of minimum energy.

Simulated Annealing (SA) has gained prominence as a powerful approach for solving NP-Hard problems. Unlike some other optimization methods, SA incorporates a probabilistic acceptance criterion, which allows it to explore and potentially accept new solutions that do not strictly improve

the current solution. This flexibility enables SA to navigate complex problem spaces efficiently, making it particularly well-suited for challenging optimization tasks.

The temperature-decreasing scheme utilized in SA plays a crucial role in the exploration-exploitation trade-off. By gradually reducing the temperature, the algorithm effectively balances the exploration of new solution areas at higher temperatures with the exploitation of promising regions at lower temperatures. This adaptive nature contributes to the algorithm ability to find high-quality solutions in complex problem domains.

In summary, Simulated Annealing proved to be a powerful optimization method, leveraging its probabilistic acceptance criterion and temperature-decreasing scheme to tackle NP-Hard problems effectively. Its ability to navigate diverse solution spaces makes it a valuable tool for addressing complex optimization challenges.

Algorithm 5 HLS Simulated Annealing

```
1: procedure SIMULATED ANNEALING(Timetable)
2:     Current ← Timetable
3:     CoolingRate ← 0.82
4:     MaxIterCooling ← InstanceDependentValue1
5:     MaxIterWithoutImprovment ← InstanceDependentValue2
6:     IterWithoutImprovment ← 0
7:     if First Execution in HLS then
8:         Temperature ← 2.5
9:     end if
10:    while MaxIterWithoutImprovment < IterWithoutImprovment do
11:        Op ← Random(Ops)
12:        Neighbor ← Neighbors(Op, Current)
13:        if EVAL(Neighbor) ≤ EVAL(current) then
14:            Current ← neighbor
15:            if EVAL(Neighbor) ≤ EVAL(current) then
16:                IterWithoutImprovment ← −1
17:            end if
18:        else if Uniform(0, 1) < exp^(EVAL(current)−EVAL(Neighbor))/Temperature
19:            Current ← Neighbor
20:        end if
21:        IterWithoutImprovment + 1
22:        if IterWithoutCooling > MaxIterCooling then
23:            IterWithoutCooling ← 0
24:            Temperature ← Temperature ∗ CoolingRate
25:        end if
26:    end while
27:    Temperature ← Reheat(Temperature)
28:    Return Current.
29: end procedure
```

Algorithm 5 shows the Simulated Annealing version within HLS, with parameters set by Müller. Notably, this method customizes parameters based on the specific CB-CTT instance, calculated by summing lecture slots during construction.

Both SA and HC share a common termination criterion: they stop when the limit of the maximum explored solution is reached without improving the best HLS-found solution. With SA, the maximum value is an instance-dependent complexity value (Müller [2009]).

Upon algorithm completion, SA adjusts its temperature for the next call by multiplying it with an instance-dependent coefficient, effectively warming up the temperature.

1.7 Conclusion

The objective of this chapter was to establish the context for this thesis work. We defined combinatorial optimization problems and explored their classification into distinct groups.

Subsequently, we focused on the two families of methods commonly employed to address these optimization challenges and elaborated on their respective working principles. Particular emphasis was placed on metaheuristics, which excel at efficiently solving highly intricate problems, even though the optimality of the results is not guaranteed.

The second section has provided a comprehensive overview of the Curriculum-Based Course Timetabling (CB-CTT) problem and its significance within the field of timetabling problems.

CB-CTT is a single-objective minimization optimization problem, where the objective function evaluates a solution based on the weighted sum of soft constraint violations. These constraints capture the desired characteristics of the schedules, emphasizing the need for efficient optimization techniques.

Furthermore, the presentation of different benchmarks, in the third section, highlights the extensive research that has been conducted on CB-CTT, indicating the breadth of work that can be undertaken in this area.

Finally, the focus was on how to solve CB-CTT and detailed neighborhood structures and genetic processes. Then a detailed explanation of the Hybrid Local Search (HLS) followed. Detailed explanations were provided regarding the strategies, heuristics, and overall functioning of HLS, aiming to facilitate a deeper understanding of this state-of-the-art method. This method won the ITC 2007 and still represents a reference of performance for the work on the CB-CTT. Furthermore, its complex structure of metaheuristics paves the way for several studies to improve the performance.

Overall, this section has set the stage for the subsequent work and analysis, providing the necessary background and understanding of the CB-CTT problem and the selected optimization methods.

Chapter 2

Solver Performance Prediction

Publications:

- Thomas Feutrier, Nadarajen Veerapen, and Marie-Éléonore Kessaci. "Exploiting landscape features for fitness prediction in university timetabling." GECCO'22 Companion: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM 2022.
- Thomas Feutrier, Marie-Éléonore Kessaci, and Nadarajen Veerapen. "Analysis of Search Landscape Samplers for Solver Performance Prediction on a University Timetabling Problem." PPSN 2022: International Conference on Parallel Problem Solving from Nature. Springer International Publishing, 2022.

Contents

2.1 Introduction

A growing number of works focus on landscape analysis to describe search spaces in the literature (Ochoa et al. [2014], Thomson et al. [2020], Verel et al. [2010]). Some works leverage the features from this analysis to make predictions or guide problem-solving strategies (Liefooghe et al. [2019], Ochoa et al. [2009]).

Landscape analysis understanding how solving algorithms navigate the search space. This approach employs algorithms to explore the problem search space and collects data on, for example, solution quality, neighborhood relations, and local optima solutions (Chapter 1). This thesis centers on the Curriculum-Based Course Timetabling (CB-CTT) problem as detailed in Chapter 1. The analysis in this Chapter primarily covers a subset of instances from the International Timetabling Competition (ITC) 2007, aiming to represent a diverse set of real-world instances. This chapter explores the landscape of CB-CTT instances and introduces performance prediction. Being able to predict algorithm performance, especially final fitness, based on features computed in a short time budget (e.g. 5 seconds) may help in choosing an appropriate solver or solving strategy. The longer allowable time budget (e.g. 5 minutes) may then be spent on the promising choice.

The chapter presents landscape analysis protocols for some CB-CTT instances. It introduces the Fitness Network model, representing the landscape using information given from exploration algorithms. The chapter details the features extracted from this search space representation. It analyzes the models and identifies three distinct zones within the search spaces, each with its unique characteristics. Finally, the chapter develops prediction models for three efficient resolution algorithms: HLS, a variant of HLS, and SA of HLS. Features are selected, and models are trained with different protocol setups, including various samplers and exploration times. These models provide valuable performance predictions.

2.2 Search Landscape

This section provides context for our work on landscapes. It begins by introducing the definitions necessary for understanding of the results that follow. In addition, it details the concepts used to motivate our work.

2.2.1 Definitions

Landscape analysis can provide valuable assistance in understanding the nature of the search space as it helps us characterize the ruggedness of the landscape and identify connectivity patterns. By computing various relevant metrics at both global and local levels, this analysis offers valuable insights that serve different purposes. For instance, it allows for an *a posteriori* analysis to enhance our understanding of solving algorithm behavior or enables performance prediction, as demonstrated in this paper.

The concept of landscape closely relates to the algorithm used to explore the search space, as illustrated by the following definitions. While exhaustive exploration provides a perfect model of the landscape, other sampling methods only offer approximations of the landscape. That means our representation and results could vary if the sampler or exploration algorithms used are not the same. In our specific case, conducting exhaustive exploration proves computationally infeasible because of the number of feasible solutions. Hence, we rely on Iterated Local Search as our chosen sampling

method because it has already demonstrated a high exploration capacity when successfully applied by Ochoa et al. [2017].

The following definitions span a spectrum, ranging from broader, more general concepts to more specific ones.

Landscape A landscape (Stadler [2002]) may be formally defined as a triplet (S, N, f) where

- S is a set of solutions, or search space,
- $N : S \longrightarrow 2^S$, the neighborhood structure, is a function that assigns, to every $s \in S$, a set of neighbors $N(s)$
- $f : S \longrightarrow \mathbb{R}$ is a fitness function.

The landscape metaphor considers the fitness of a solution as the height of the landscape, thus forming surfaces that can exhibit various characteristics, such as smoothness or ruggedness. Plateaus within these surfaces can contribute to the search difficulty.

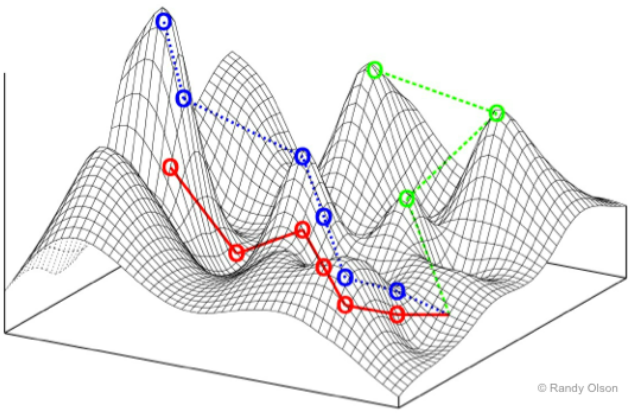


Figure 2.1: A hypothetical search landscape (image credit: Randy Olson) for a two-variable search space (x and y axes). The z-axis is the fitness value. The colored lines indicate potential trajectories of some algorithms in the search space.

Figure 2.1 illustrates a hypothetical landscape, where each intersection represents a solution with a fitness value equivalent to its height. There are obvious limits to the metaphor since any non-trivial landscape is actually highly multidimensional.

In our work, exhaustive exploration of the landscape is not feasible due to computational constraints. Hence, we employ the Iterated Local Search (ILS) algorithm to sample the search space. Specifically, our focus lies on local optima (Ochoa et al. [2017]).

Algorithm 6 Iterative Local Search

```
1: procedure ILS(Timetable, MaxTime)
2:     Current ← Timetable
3:     start(Time)
4:     while Time < MaxTime do
5:         Current ←Hill Climbing(current)                ▷ Algorithm 3
6:         Current ← Perturbation(current)
7:     end while
8:     Return Current.
9: end procedure
```

Algorithm 6 outlines the ILS procedure, where the *Current* variable represents the current timetable being evaluated. The algorithm iteratively applies Hill Climbing (Algorithm 3) for exploitation and a perturbation for exploration. The ILS sampler records every local optimum it encounters (including fitness) as well as the sequence in which the local optima were encountered.

Plateau A plateau, sometimes called a neutral network, is usually defined as a set of connected solutions with the same fitness value. Two solutions s_1 and s_2 are linked if they are neighbors, i.e., $s_2 \in N(s_1)$. Depending on the neighborhood function, we may also have $s_1 \in N(s_2)$.

In our work, plateaus are identified as consecutive sequences of solutions that exhibit an identical fitness value. However, due to the trajectory-based nature of the sampling approach we use, it poses a considerable challenge to say whether two distinct sequences, both characterized by the same fitness values, truly belong to the same plateau. The inherent difficulty arises from the lack of complete information about the landscape structure and the limitations imposed by the trajectory-based exploration method.

Ruggedness of landscape Solving combinatorial optimization problems aims to find the global optima, i.e., the solution(s) with the lowest score in the case of minimization. However, with approximate methods, the exploration of the search space relies on neighborhood operator-based moves, and the acceptance of a new neighbor as the current solution is a choice made by the algorithm.

The outcome obtained from these methods depends on the problem and the relationships between the neighbors. This difficulty in finding the best solution is determined by the structure of the search landscape. To understand this structure, various studies have explored the search landscape, leading to the classification of landscapes into two classes: smooth and rough. In a smooth landscape, the metaphor of a valley is perfect to explain the idea. The optimal solution is located at the bottom of the valley, and the method needs to go down to reach it. Figure 2.2 shows a representation of a smooth landscape with only 4 local optima, this type of landscape describes generally an easy problem to solve.

On the other hand, in a rough landscape, there are many peaks and valleys and, therefore, numerous local optima. Finding a good solution means having to go through worse neighbors before reaching an even better solution. Considering the structure of the landscape is crucial because it directly relates to the choice of neighborhood operators that allow for efficient exploration of the search landscape.

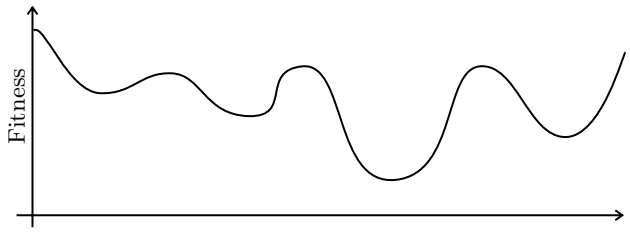


Figure 2.2: Smooth search landscape.

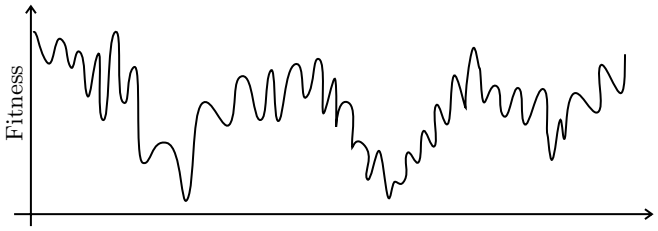


Figure 2.3: Rugged search landscape with a large number of local optimal.

Local Optima Network (LON) LONs (Tomassini et al. [2008]) provide compressed graph models of the search space, where nodes are local optima and edges are transitions between them according to some search operator. LONs for neutral landscapes have been studied before by Verel et al. [2010]. The latter work introduces the concept of *Local Optimum Neutral Network* that considers that a neutral network is a local optimum if all the configurations of the neutral network are local optima. Another approach to neutrality in LONs is by Ochoa et al. [2017] who develop the notion of *Compressed LONs* where connected LON nodes of equal fitness are aggregated together. For our purposes, we consider two slightly different kinds of networks: Timeout Plateau Networks and Fitness Networks.

Timeout Plateau In order to explore the landscape, this work employs two variants of Iterated Local Search (ILS) to compare the resulting data. The first ILS variant we utilize, called $\text{ILS}_{\text{neutral}}$, incorporates a hill-climber that ceases its exploration if it remains on the same plateau for an extended period of time, specifically 50,000 consecutive evaluations with identical fitness. Additionally, the Hill-Climber in this variant accepts the first non-deteriorating move, which can be an improving or a neutral neighbor. We refer to the plateaus discovered in this manner as *timeout plateaus* since the hill-climber is unable to escape from them within a specified number of iterations. However, it is important to note that these *timeout plateaus* are not necessarily sets of local optima. Through exploratory analysis, we found that at least 1% of these solutions were not actual local optima. To do this, we used data from the protocols presented in Section 2.2.2.2. Then, for each solution of all the identified timeout plateaus, we looked at each of their neighbors using the six neighborhood operators used in HLS. If there is a strictly best neighbor, the solution is not a local optimum.

The second variant, $\text{ILS}_{\text{strict}}$, shares the same components as $\text{ILS}_{\text{neutral}}$, but with the crucial difference that it only accepts strictly improving solutions. Consequently, the *timeout plateaus* identified in $\text{ILS}_{\text{strict}}$ trivially consist of a single solution.

Timeout Plateau Network A Timeout plateau network is a graph where each node is a contracted timeout plateau, where an edge represents a transition between two such plateaus. In practice, this transition is an ILS perturbation followed by a Hill Climbing. Timeout Plateau Networks are a set of independent chains, where each represents one ILS run. Therefore our Timeout Plateau Networks do not exhibit much connectivity.

Fitness Network This is a simplification of Timeout Plateau Networks where all nodes with the same fitness are contracted together. This provides a graph structure with much higher connectivity than a Timeout Plateau Network. While it is not meant as an accurate representation of the landscape, several different metrics related to connectivity between fitness levels can be computed. Note that this is an even greater simplification than *Compressed LONs* (Ochoa et al. [2017]) which only aggregate nodes sharing the same fitness that are connected together at the LON level.

2.2.2 Experimental Protocol

Experiments use 19 of the 21 instances proposed for ITC 2007. Instances 01 and 11 are set aside since they are very easy to solve. The latter are optimally solved by all methods we tested. Thus, predicting their fitness is too easy.

2.2.2.1 Performance Protocol

The objective of this protocol is to compute the performance of three solvers: Hybrid Local Search (HLS), iterated Simulated Annealing (SA), and a variation of HLS called $\text{HLS}_{\text{strict}}$.

HLS combines the algorithms Hill Climbing (HC), Great Deluge, and Simulated Annealing in an iterated sequence. In our experiments, we employ two versions of HLS. The default version, referred to as HLS, accepts solutions that are equal to or better than the current solution during the Hill Climbing phase. On the other hand, $\text{HLS}_{\text{strict}}$ uses a strict acceptance criterion for Hill Climbing.

Additionally, we test the performance of iterated Simulated Annealing (SA). This involves reusing the SA component from HLS and incorporating it into a loop that terminates when the runtime budget is reached. Only these three solving algorithms are considered, as preliminary work has shown that they are both efficient and significantly different on the ITC instances.

All executions of the solvers were conducted on an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz. Each solver was given a time budget of 5 minutes and tested 100 times on each instance.

By following this protocol, we aim to assess the performance of HLS, $\text{HLS}_{\text{strict}}$, and iterated SA. The experiments provide valuable insights into the behavior and capabilities of these solvers in solving the given ITC instances.

2.2.2.2 Sampling the Search Space

To explore our search space, we employ one of two Iterated Local Search (ILS) algorithms that are based on algorithmic components from the Hybrid Local Search.

Both ILS variants include an Iterative Bounded Perturbation (IBP) technique (Müller [2009]). The HC phase terminates when it reaches a local optimum or after evaluating 50,000 solutions without any strict improvement. Figure 2.4 shows the process followed by ILS samplers.

The distinction between the two ILS samplers lies in their acceptance criteria. The first sampler, $\text{ILS}_{\text{neutral}}$, adopts the same strategy as HLS, accepting solutions that are equal to or better than the

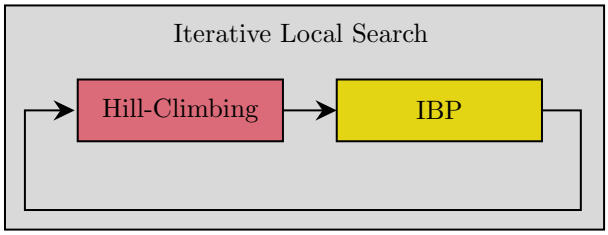


Figure 2.4: Iterated Local Search sampler

current solution. On the other hand, the second sampler, $\text{ILS}_{\text{strict}}$, follows the strategy of $\text{HLS}_{\text{strict}}$ and only accepts strictly improving solutions.

The IBP technique operates by taking a baseline fitness value, denoted as $Fit_{FirstSol}$, which corresponds to the fitness of the first solution obtained after the initial construction phase. It then deteriorates the final solution found by the HC phase, denoted as $Fit_{LastSol}$, in order to reach a solution with a fitness value equal to $Bound = Fit_{LastSol} + 0.1(Fit_{FirstSol} - Fit_{LastSol})$.

By utilizing these ILS algorithms and incorporating the IBP technique, we aim to effectively explore the search space. The different acceptance criteria in $\text{ILS}_{\text{neutral}}$ and $\text{ILS}_{\text{strict}}$ allow us to compare the performance and efficiency of these approaches in finding high-quality solutions.

Due to memory constraints, $\text{ILS}_{\text{neutral}}$ only records fitness and size of all timeout plateaus met. $\text{ILS}_{\text{strict}}$ just saves the fitness of the last solution of each HC, its timeout plateaus have a size of 1. Both ILS samplers are run 100 times with a time budget of {5, 10, 20, 30} seconds per run on each instance. This budget was set in order to obtain enough predictive information on the landscape. Each (time budget, sampler) pair produces one network, so we have 8 networks for each instance.

Timeout Plateau Network Each Timeout Plateau Network is constructed using data collected from 100 individual runs. In the network, each node represents a timeout plateau, while each directed edge corresponds to a transition involving a perturbation followed by a hill-climber step. Initially, the weight assigned to each directed edge is set to 1.

Figure 2.5 shows a graphical representation of the timeout plateau obtained using $\text{ILS}_{\text{neutral}}$ with a time budget 30 seconds. At this stage, the individual trajectories of the ILS sampler run have no connectivity, one run corresponds to one chain in a graph representation (Figure 2.5 shows 100 independent chains), and further information extraction requires a contraction step. This step aims to consolidate and enhance the understanding of the network by identifying connections and relationships between different timeout plateaus.

Fitness Network Given the high level of neutrality observed in the problem, we have made the broad assumption that all solutions with the same fitness value belong to a single wide plateau. This hypothesis enables us to create a connected network more easily, capturing the relationships between different plateaus.

The contraction process of the Timeout Plateau Network into a Fitness Network retains all the necessary data for computing the metrics discussed in Section 2.2.3.2. The weights assigned to the directed edges in the Fitness Network are determined by summing the weights of the contracted directed edges.

To provide a visual representation of the Fitness Network, Figure 2.6 depicts the graph of a

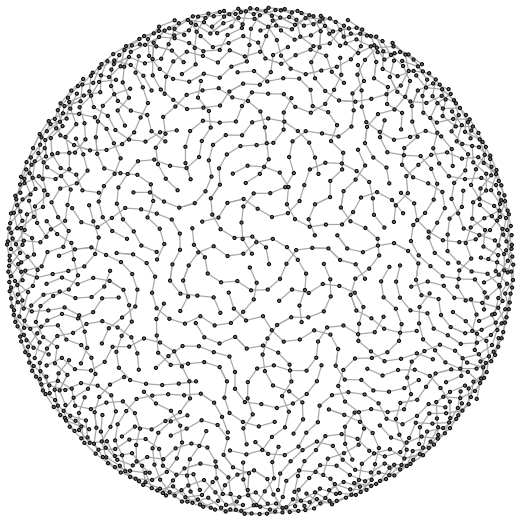


Figure 2.5: Timeout Plateau Network of Instance 12 built by $\text{ILS}_{\text{neutral}}$ with 30 seconds.

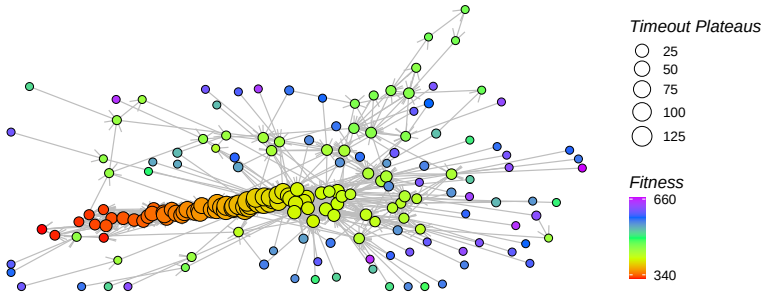


Figure 2.6: Fitness Network of Instance 12 built by $\text{ILS}_{\text{neutral}}$ with 30 seconds.

fitness network. This graph showcases the interconnectedness between plateaus, illustrating the transitions and pathways between different levels of fitness.

2.2.3 Data

This section serves as an introduction to the features used to describe each instance of CB-CTT in the ITC dataset. Our goal is to provide an overview of the various features extracted from each instance, which encompass both landscape analysis metrics and instance-specific characteristics.

2.2.3.1 Instance features

The instance features encompass various aspects of the CB-CTT problem, providing valuable descriptive information and quantifying its complexity. These features serve to characterize each problem instance and offer insights into the scheduling requirements and constraints. Basic descriptive data includes:

- Courses: the number of courses in the problem instance.
- Rooms: the number of rooms available for scheduling lectures.
- Constraints: the number of unavailability constraints placed on courses or rooms.
- Days: the number of days in the scheduling horizon.
- Periods Per Day: the Maximum Number of Periods available within each day for scheduling courses.
- Teachers: the number of teachers available for teaching courses.
- Curricula: the number of Curricula, which are groups of related courses.
- Lectures: the total number of lectures that need to be scheduled.
- Min|Mean|Max RoomCapacity: the Minimum, Average, and Maximum Room Capacities across all available rooms.
- Min|Max Student: the Minimum and Maximum Number of Students enrolled in each course.
- Min|Mean|Max Course by Curriculum: the Minimum, Average, and Maximum Number of Courses associated with each curriculum. This helps assess the balance of courses across curricula.
- Co_cu Courses divided by Curricula: the ratio of Courses to Curricula. The value differs from the Mean Course by Curriculum because some courses may belong to multiple curricula, resulting in a different average value.
- Min|Max mwDays: the Minimum and Maximum Minimum Working Days required for scheduling each course. These features reveal how courses are distributed across the week.
- Lecture Occupancy: This metric divides Lectures by the number of timeslots, i.e. Days multiplied by Periods Per Day. The idea is to represent what the readings might occupy if everyone had a different timeslot and only one room was available.
- Min|Mean|Max LecturesbyCourse: the Minimum, Maximum, and Average Number of Lectures required for each course in the problem instance. This metric illustrates the variation in lecture requirements among different courses.
- TeachersNeeded: the Average number of teachers required for a single timeslot, calculated by dividing the total number of lectures by the number of available timeslots. This feature provides insights into teacher allocation demands and resource utilization.

In total, there are 24 distinct instance features, collectively capturing various aspects of CB-CTT problem. These features enable an assessment of each instance characteristics and complexity.

2.2.3.2 Landscape metrics

The LON model model introduces metrics derived from graph theory, which enable the characterization of the structure of combinatorial landscapes. These metrics, originally designed for LONs, can also be effectively applied to our Fitness Networks.

By leveraging these metrics, we gain valuable insights into the organization and properties of the search space, facilitating a deeper understanding of the combinatorial optimization problem at hand.

Node-Level Metrics Node-level metrics include two sets of metrics. The first one relates to what the nodes represent.

- Plateaus: Number of plateaus that have been contracted to form the node.
- Size: Sum of the number of accepted solutions in the contracted plateaus. This shows whether solutions of this fitness are often found or not.
- Fit: Fitness represented.
- Loops: Number of consecutive loops on the same fitness. Increase by 1 each time that, after a disruption phase and an intensification phase, an ILS sampler still returns the same fitness. It is an estimator for attraction power.

The metric "Plateaus" corresponds to the size of nodes for Figure 2.6.

A second set of metrics relates to the connectivity of the nodes within the network.

- Degree_in: Number of different nodes that lead to the current node. If the degree is low, it means that this fitness is rarely found. The opposite is true. In the other case, many samplers arrive at this solution.
- Degree_out: Number of different nodes the current node leads to. The idea is close to Degree_in but concerns the moment when the samplers have moved from the current fitness to another value.
- Weight_out: Number of times through all runs where we escape from this fitness (without loops)
- Weight_in: Number of times through all runs where exploration went to this fitness (without loops)

We also consider two variants of weight and degree metrics. For some given node, the *better* (resp. *worse*) variant only considers arcs between this node and better (resp. *worse*) nodes.

The above metrics are computed for each node and five points corresponding to the quartiles (Q1, median, and Q3) and the 10th and 90th percentiles of the distribution are used as features for our models.

Network Metrics Another set of metrics called Network Metrics describes the networks themselves, including:

- Mean fitness.
- Number of timeout plateaus.
- Number of nodes: The number of different fitnesses among the timeout plateaus found by samplers.
- Number of edges: This corresponds to the number of different changes from one fitness to another made by the samplers.
- Mean of the edge weights.
- Number of sink nodes: Sink nodes are defined as nodes that lack outgoing arcs to nodes with better fitness, indicating that they represent local optima in the search space.
- Coefficient of assortivity: a measure of similarity between linked nodes (Ochoa et al. [2014]), plays a significant role in characterizing the connectivity patterns within the network. The coefficient of assortivity increases as the number of connected nodes with similar attributes becomes more abundant.
- The transitivity coefficient or the clustering coefficient (Ochoa et al. [2014]): it quantifies the probability of a link existing between neighboring vertices when one vertex is selected. It provides insights into the level of local clustering or interconnectedness within the network.

By examining these metrics, we can gain a deeper understanding of the structural properties and characteristics of the network, shedding light on the dynamics of the optimization process.

Finally, we use the PageRank centrality to analyze the networks since Herrmann et al. [2018] showed that related metrics were useful when working with LONs. The higher the PageRank, the more important and attractive the node concerned. Three measures summarize page rank into features:

- The mean PageRank over nodes,
- The PageRank of best fitness node,
- The mean PageRank over sink nodes (nodes without arcs leading to a better fitness).

The landscape metrics are summarized in Table 2.1.

2.3 Results of Search Landscape Exploration

The experimental results can be divided into three main parts. Firstly, we show how our representations allow us to highlight three major fitness groups. Secondly, we explore the impact of the choice of sampler and the allocated exploration time on the portion of the landscape that is explored. Finally, we leverage the data extracted from our representations and instance features to construct a predictive model for the performance of three solvers on ITC instances in an efficient manner.

Table 2.1: Landscape metrics summary

Type	Landscape Metric	Description
Node-Level Q1|Q2|Q3|10th|90th	Plateaus	Number of plateaus.
	Size	Number of accepted solutions.
	Fit	Fitness represented.
	Loops	Number of edges that loop.
	Degree_in$_{better\|worse}$	Number of different in-going arcs.
	Degree_out$_{better\|worse}$	Number of different outgoing arcs.
	Weight_out$_{better\|worse}$	Sum of weights of outgoing arcs.
	Weight_in$_{better\|worse}$	Sum of weights of in-going arcs.
Network-Level	Plateaus	Number of plateaus.
	$\overline{Fit}$	The average fitness value within the network.
	Nodes	Number of nodes in the network.
	Edges	Number of edges
	Sink	Number of nodes without improving outgoing arcs.
	Assortivity	Measures similarity between linked nodes.
	Transitivity	Provides insights into local clustering.
	Mean PageRank	Average PageRank centrality over all nodes.
	PageRankBest	PageRank of the node with the best fitness.
	Mean PageRank$_{\text{Sink}}$	Average PageRank centrality over sink nodes

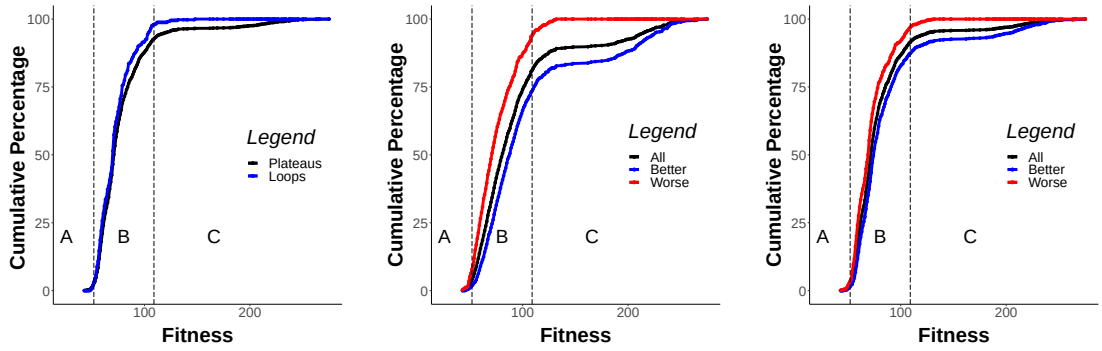
2.3.1 Identifying Major Sections of the Landscape

This section aims to identify the most relevant regions within our representation of the search landscape. Referring back to Figure 2.6, we can observe a rough indication of three distinct groups of nodes. The first is a group of rarely visited nodes with poor fitness. Then there is a group of highly interconnected nodes with very diverse fitness. Finally, we can consider that the best nodes form their own small group: this third group is connected to the highly-connected second group but connections within the group are more rare and the size of nodes is smaller because they are visited less often. However, this is a visual analysis, we now turn to a more rigorous analysis based on landscape metrics. By analyzing the data, we can obtain factual insights that complement our visual observations.

In order to identify and measure the most and least promising regions of the network, we have plotted the cumulative distribution of several key features as a function of fitness. This allows us to pinpoint low and high fitness regions with greater precision. Figure 2.7 presents the plots for a representative instance, illustrating our approach.

Figure 2.7a plots the cumulative percentage distribution of the sum of plateaus and loops. For the former, one node corresponds to the number of distinct timeout plateaus with the corresponding same fitness value. Loops is a node-level metric presented in Section 2.2.3.2. It represents the number of times the exploration algorithm, consecutively finds two timeout plateaus with the same fitness. Figure 2.7b and Figure 2.7c respectively show the cumulative percentage, as a function of node fitness, of the out-going degree of nodes and the weights of the out-going arcs connected to nodes. These two features are measures of the density over the networks.

From the sigmoid shape of the distributions, we can identify three groups of nodes :



(a) Size (b) Degree (c) Weight

Figure 2.7: Cumulative percentage of the sum of node size, degree, out-going edge weights as a function of fitness for 30 seconds Instance 12 Fitness Network with $\text{ILS}_{\text{neutral}}$. Vertical lines represent fitness limits between groups.

- **Group A:** The first sub-network has a low local density. Its fitness values are little visited and are the best found.

- **Group B:** This set of nodes represents a big part of network. Nodes correspond to intermediate fitness values, and solving methods often find them. Vertices are inter-connected and arcs are frequently traveled, with high weights.

- **Group C:** The nodes in this group are almost all of size one. They represent the worst fitness values found. They are not connected to each other because their arcs lead only to vertices belonging to Group B.

The distributions across all instances display a consistent sigmoid shape, suggesting similar behaviors and the presence of the same number of distinct groups.

This work focuses on ITC benchmark instances, which, at present, consist of a limited dataset. Nevertheless, the objective is to design a methodology that exhibits adaptability to a broader range of instances. Therefore, it becomes essential to streamline and automate the operations to the greatest extent possible. To automate the grouping of nodes, we employ a method that involves identifying two points of inflection, or kinks, on the curve. These kinks are determined based on the distribution of the sum of plateaus, represented by the black curve in Figure 2.7a.

The first kink is determined by setting a threshold of 1% difference between consecutive fitness values. If the difference exceeds this threshold, the corresponding fitness values are classified as part of Group A, while the subsequent values are assigned to Group B. The second kink is identified when the difference drops below 1%, and the remaining fitness values are classified as Group C.

In order to analyze the sub-networks A and B, we compute several features including the mean fitness, number of nodes, number of plateaus, and the number of sink nodes. These features provide insights into the characteristics of each sub-network. Overall, there are 8 features that are relevant to the analysis of the sub-networks and are described in Table 2.2.

Table 2.2: Eight additional features computed for sub-networks. X can either represent group A or group B.

Feature	Description
$\overline{Fit}_X$	Mean of the fitness nodes in the sub-network
Node_X	Number of nodes in the sub-network
Plateaus_X	Number of plateaus in the sub-network
Sink_X	Number of sink nodes in the sub-network

2.3.2 Effects of Sampler Choice on Sampled Fitness Values

In this analysis, we investigate the impact of different (sampler, time budget) pairs on the distribution of fitness. The two samplers under consideration are $\text{ILS}_{\text{strict}}$ and $\text{ILS}_{\text{neutral}}$, which employ different strategies for accepting neighboring solutions, namely strict or neutral acceptance.

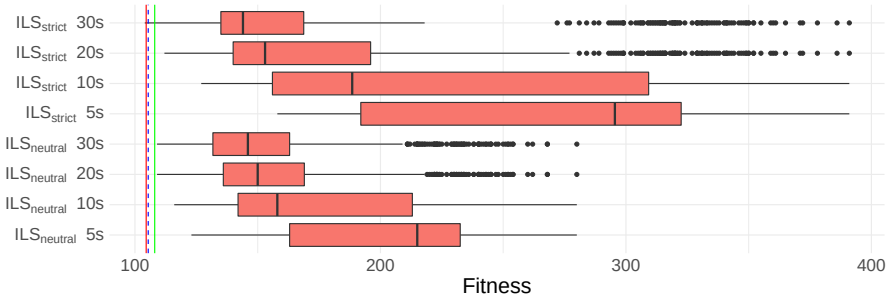


Figure 2.8: Fitness distribution for each (sampler, time budget) pair on instance 21 is depicted here. Vertical lines indicate the mean fitness values achieved by each solver in 5 minutes, with red representing $\text{HLS}_{\text{strict}}$, blue for HLS, and green for SA.

Figure 2.8 shows the fitness distribution of the timeout plateaus obtained by each sampler on one representative instance as boxplots. As might be expected, we can observe a clear relationship between the time budget and the skew of the distribution towards better solutions. This stands for both samplers. The best solutions found with 20 and 30 second sampling runs reach, or are very close to, the mean fitness value obtained by the solvers when run for 5 minutes. In addition, it stands to reason that as time increases, timeout plateaus are added but those found within shorter time budgets remain, only they represent a smaller proportion. The tighter time budgets also, naturally, sample fewer solutions.

The fact that some sampling scenarios reach, or are very close to, the mean solver fitness likely indicates that any feature that encodes some information about the best sample fitness is very important to the model. In our case, this is exactly what $\overline{Fit}_{\text{A}}$, the mean fitness of Group A, does. This is investigated further in Section 2.4.

The boxplots also highlight the distinct behaviors of the two samplers. It is evident that $\text{ILS}_{\text{neutral}}$ consistently discovers better solutions in terms of fitness within the same time frame. Notably, Figure 2.8 demonstrates that as the time decreases, the disparity between the performances of the two ILS samplers becomes more pronounced. This discrepancy can be attributed to the neutral acceptance policy employed by $\text{ILS}_{\text{neutral}}$, which leads to the exploration of plateaus

located further down the landscape. These observations suggest that $\text{ILS}_{\text{neutral}}$ is not only more efficient but also more robust in finding improved solutions when faced with limited time constraints. The neutral strategy enhances the performance of ILS and improves the sampling effectiveness specifically in the most promising regions of the search space.

2.4 Performance Prediction

In the previous sections, we have focused on the analysis of the landscape of the Curriculum-Based Course Timetabling (CB-CTT) problem, utilizing our representation and various landscape metrics. This analysis has provided us with a rich dataset encompassing extensive information about each instance of the problem. We have gained insights into the fitness distributions.

Building upon this understanding of the problem landscape, the focus of this section shifts towards a crucial objective: predicting the performance of three solvers on the ITC instances. By developing prediction models, we aim to estimate the final fitness achieved by each solver using a subset of the available data.

The motivation behind performance prediction goes beyond its immediate practical implications. While accurate predictions can be immensely beneficial in guiding solver selection and decision-making processes, they also serve as a means to validate the effectiveness of our landscape exploration. By successfully predicting solver performance, we substantiate the notion that our exploration and analysis of the landscape have uncovered meaningful patterns and insights.

To accomplish this task, we employ a data-driven approach, leveraging the information extracted from our representation and instance features. Through careful feature selection and model development, we aim to uncover the key factors that influence solver performance and ascertain the most influential features in determining solution quality. Moreover, we investigate the impact of different sampler configurations and time budgets on the performance prediction models, assessing their effectiveness in accurately predicting solver performance for CB-CTT problem.

The outcomes of our prediction models offer some insights into the relationship between problem characteristics, solver behavior, and fitness solutions. Moreover, they validate the usefulness of exploring the landscape, and not only considering instance features, as the extracted features play a an important role in accurately predicting solver performance.

Subsequent sections outline our methodology for performance prediction, present the results obtained from our prediction models, and discuss the broader implications and potential applications of our findings. Through this analysis, we hope to enhance our understanding of the CB-CTT problem and provide valuable insights for future works and practical applications.

2.4.1 Model Construction and Evaluation

Research papers have demonstrated a relationship between landscape data and the performance of resolution algorithms (Daolio et al. [2017], Liefooghe et al. [2019], Thomson et al. [2020]). That shows that machine learning models can learn from landscape data.

We employ a model-building process consisting of feature selection followed by linear regression to predict the fitness value. Instance selection is a process often used to improve the performance of predictive models. The linear regression model is used because it is a well-known and efficient type of model that deals well with quantitative data. The objective is then to assess how the sampler and its budget affect the quality of the prediction for different solvers.

Pre-processing After merging all data, there are a total of 120 features, some of which may not be useful. We first remove 21 features with a constant value: most of them are 90th percentile features, i.e., they describe the top of the landscape.

Features are then standardized, thus each feature mean is equal to 0 and the standard deviation is equal to 1. Remember that our dataset contains only 19 instances. This imbalance between the number of instances and the number of features can potentially undermine the predictive power of our models. Therefore, it is important to address this issue and aim to reduce the discrepancy for more robust predictions.

After these two steps, features have to be selected in order to improve the potential success of our models. We employ a correlation preselection approach to identify features that are highly correlated with the fitness value, which represents the outcome variable for one of the three solvers utilized. A fixed threshold of 0.9 is set to determine the features that meet this criterion.

We explored different threshold values, such as 0.8 and 0.7, in the correlation preselection process. However, we found that using a relatively high threshold of 0.9 resulted in a more consistent and stable feature selection. Lower thresholds introduced more fluctuation in the selected features, potentially leading to less robust and reliable predictions. Therefore, we opted for the higher threshold to ensure a more reliable subset of features for our prediction models.

Table 2.3: Selected features with respect to sampler and budget. A tick indicates a selected feature.

ILS_{strict}				$ILS_{neutral}$				Feature	Description
5s	10s	20s	30s	5s	10s	20s	30s		
✓	✓	✓	✓	✓	✓	✓	✓	Cu	Number of Curricula
✓	✓	✓	✓	✓	✓	✓	✓	$\overline{Fit}_{\mathrm{A}}$	Mean fitness of Group A
✓	✓	✓	✓		✓	✓	✓	$\overline{Fit}_{\mathrm{B}}$	Mean fitness of Group B
✓	✓	✓	✓	✓	✓	✓	✓	$\overline{Fit}$	Mean fitness
					✓			$\mathrm{Sink}_{\mathrm{B}}$	Number of sink nodes in Group B
					✓			CC	Number of connected components

Table 2.3 provides a summary of the selected features based on the ILS sampler version and the allocated budget. The selection process results in 3 to 6 features being chosen, with the number of curricula, the mean fitness of all sampled timeout plateaus, and the mean fitness of groups A and B consistently included. These latter two features not only capture fitness information but also indirectly convey insights about the proportion of plateaus, as they are utilized in creating the groups. Notably, the selected features exhibit a relative uniformity across different samplers and budgets, with the absence of more complex features.

A potential limitation of using this restricted feature set is the presence of multicollinearity due to the inclusion of different fitness-related features. Multicollinearity can complicate the interpretation of regression coefficients. However, in this study, our primary focus is on the predictive accuracy and precision of the models rather than the interpretability of the coefficients. As such, the issue of multicollinearity does not pose a significant concern for our analysis.

Evaluation To ensure a robust evaluation of the models, particularly considering the limited number of instances available, we employ complete 5-fold cross-validation.

Cross-validation uses complementary subsets of the data for training and testing in order to

assess the model ability to predict unseen data. With a k-fold approach, data are partitioned into k subsets, one of which is retained for testing, the remaining $k-1$ being used for training the model. The cross-validation process is repeated k times such that each subset is used once for testing. The results are then averaged to produce a robust estimation. The specificity of the complete cross-validation is to apply a k-fold on all possible cutting configurations (Kohavi et al. [1995]). Therefore $\binom{m}{m/k}$ configurations are considered instead of only k. In our case, with 19 instances and 5 folds, we have $\binom{19}{4} = 3876$ configurations. This complete 5-fold cross-validation algorithm has two main advantages. The first is to check whether the model can predict final fitness for new instance The second smooths out the impact of how the data are split between training and test sets. When two problem instances are very similar and are not in the same fold, information about the first helps in prediction. However, our objective is to obtain a robust model for all problem instances and not only very similar instances. Testing all combinations reduces this boosting effect.

The quality of the regression is assessed using the coefficient of determination, R^2, a well-known indicator for regression tasks.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

The coefficient of determination is a measure that indicates the proportion of the total variance in the dependent variable explained by the independent variables in a regression model. The number of data points in the dataset is n, representing the total observations or samples. y_i is the i-th actual value. $\bar{y}$ is the mean of the observed values y_i, representing the average value of the dependent variable. Finally, $\hat{y}_i$ corresponds to the predicted values for the i-th data point, derived from the regression model.

2.4.2 Experimental Results

Using the gathered data and the selected features, we constructed a total of 72 linear regression models. These models aim to predict the performance of three solvers (HLS, $\text{HLS}_{\text{strict}}$, and SA) using two ILS samplers ($\text{ILS}_{\text{neutral}}$ and $\text{ILS}_{\text{strict}}$) across four different time budgets.

We specifically focus on the inclusion or exclusion of the $\overline{Fit}_{\text{A}}$ feature, given its observed significant impact on the prediction performance. We can even imagine that in some cases, it is this feature that directly may give the result of the prediction model. This feature, which represents the mean fitness of Group A, exhibits the highest correlation with the final fitness values. The resulting $\mathbf{R^2}$ values for each model are displayed in Figure 2.9, where each line represents a specific (sampler, solver) pair. By employing this approach, we ensure a robust evaluation, particularly considering the limited number of instances available.

The results reveal a consistent trend of achieving relatively good to very good performance in all cases, with $\mathbf{R^2}$ values ranging from 0.62 to 0.97. This notable level of predictability can be attributed to the feature selection process employed, as previously outlined. By preselecting the most relevant features, the models are tailored to focus on the key variables that exert a significant influence on the final fitness. Consequently, the models demonstrate good accuracy and reliability in predicting the performance of the various solvers.

As anticipated, $\overline{Fit}_{\text{A}}$ proves to be an excellent standalone predictor of the final fitness achieved by the different solvers. However, the models that incorporate all the selected features outperform it, albeit marginally, suggesting the added value of utilizing the additional features. While models excluding $\overline{Fit}_{\text{A}}$ exhibit slightly lower performance, they still remain competitive. An interesting

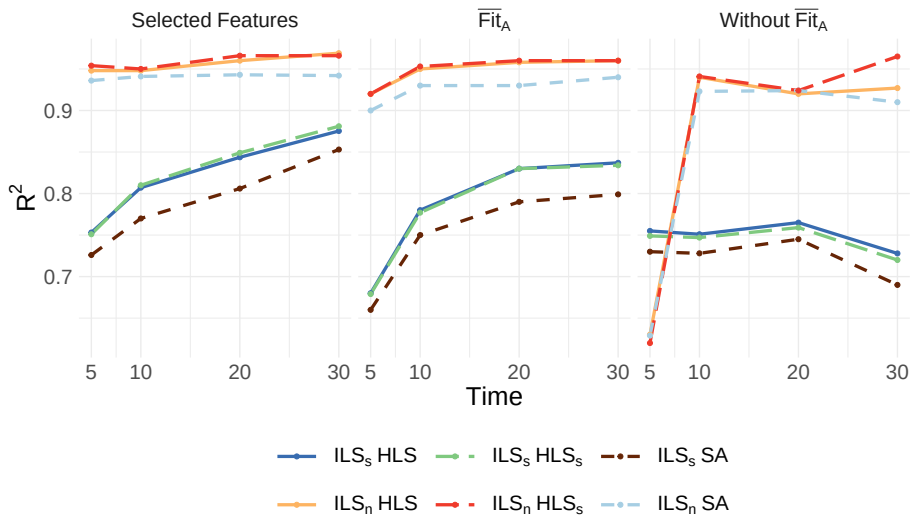


Figure 2.9: R^2 values for models, where each point represents a (sampler, solver) pair.

deviation from this overall trend is observed in the models constructed using $\text{ILS}_{\text{neutral}}$ with a 5-second sampling budget, where the exclusion of $\overline{Fit}_A$ leads to a significant decrease in R^2. This particular scenario is characterized by sampling runs that did not reach the mean fitness of the solvers, thereby rendering $\overline{Fit}_A$ a non-trivial and indispensable feature in this context. Furthermore, in this context, opting for a model based on data from 5-seconds-$\text{ILS}_{\text{neutral}}$ proves to be significantly more cost-effective than the 5-minute duration required by the solvers.

When comparing neutral versus strict acceptance sampling, $\text{ILS}_{\text{neutral}}$ consistently outperforms $\text{ILS}_{\text{strict}}$, except in the specific scenario mentioned earlier. This result aligns with expectations, as the landscape is known to exhibit neutrality. However, it is surprising that strict acceptance still produces decent models despite this characteristic. This suggests that strict acceptance is able to capture relevant information and provide valuable predictions, despite the landscape inherent neutrality.

Next, let us examine the performance of models for the different solvers. Remarkably, the linear regression models adequately predict the performance of all three solvers, despite the variations in their sampling algorithms, as observed in other studies (Thomson et al. [2020], Feutrier et al. [2022]) and exploited in Chapter 4. It is worth noting that the prediction for SA, being the most distinct from the sampling algorithm, is slightly less accurate compared to the other solvers. However, it is interesting to observe that there is no significant difference in the prediction accuracy between HLS with neutral (default version) and strict acceptance, suggesting that the two versions of HLS have statistically similar performances. However, each HLS version has statistically different final fitness based on Wilcoxon test results.

A general trend that holds for most cases, is that R^2 improves as the sampling budget increases, which seems fairly intuitive. Perhaps surprisingly, however, predictive performance remains almost flat (but very good) when all the selected features and neutral sampling are used. This robustness with respect to time makes it easy to recommend using the smallest budget in this scenario.

2.5 Conclusion

In this chapter, we proposed an approach to analyze the search landscape of the CB-CTT problem. We introduced the concept of a Fitness Network to represent the landscape. Our approach allows for the exploration of this landscape, employing two different explorers with varying time budgets. This enables us to examine how exploration strategies affect the distribution of fitness.

Through our landscape analysis, we uncovered essential patterns and behaviors within the search landscape. Additional features derived from the landscape analysis proved to be relevant in predicting performance. Our predictive models effectively captured the relationship between the selected features and the final fitness of the solvers.

Notably, our analysis revealed that the mean fitness of Group A emerged as a highly significant predictor of performance. When combined with a modest time budget of just 5 seconds, this feature led to accurate performance predictions. This finding underscores the value of an economical approach, focusing on the mean fitness of Group A within a shorter time frame, to provide reliable performance predictions.

Through our landscape analysis, exploratory strategies, and feature engineering, we gained valuable insights into the dynamics of the search landscape. However, it is essential to emphasize that, in this specific scenario, creating a predictive model avoids the need to run a solver that takes 5 minutes for each case. When compared to the 5-second runtime of the samplers, this approach proves to be highly efficient and cost-effective.

Our findings may limit the direct practical implications of the landscape analysis in this context, but they shed light on the specific feature that holds predictive power. This insight guides future problem-solving scenarios, suggesting that focusing on the mean fitness of the solutions below the 10th percentile can be a reliable indicator of solver performance. Furthermore, this knowledge allows for efficient resource allocation, as it demonstrates that good predictions can be obtained with a minimal time budget.

Nonetheless, it is important to note that the development of efficient models is not necessarily based on the inclusion of the mean fitness of sub-network A in training. By leveraging a well-chosen set of features extracted from fitness networks and instances, it is possible to train models that surpass the effectiveness achieved with just this singular feature. Such models may exhibit a higher level of performance and predictive accuracy, demonstrating the versatility and potential for optimization in the model-building process.

Chapter 3

Feasibility Prediction Models for Benchmark Selection

Publication:

- Thomas Feutrier, Nadarajen Veerapen, and Marie-Éléonore Kessaci. "Improving the Relevance of Artificial Instances for Curriculum-Based Course Timetabling through Feasibility Prediction." GECCO'23 Companion: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM 2023.

Contents

3.1 Introduction

In the previous chapter, we developed a model to predict algorithm performance on instances based on the search landscape. However, a limitation of our work was the availability of only a small set of instances at that time. To effectively perform algorithm configuration, a significantly larger sample is required to cover a wide range of possible cases.

Fortunately, the work of De Coster et al. [2022] has addressed this issue. They published a comprehensive study on the automatic selection of optimization algorithms applied on the Curriculum-Based Course Timetabling problem. Their work encompasses various aspects, including instance generation, training set creation, and data analysis, culminating in models that predict the most suitable algorithm for each instance.

One notable outcome of their study was the identification of Hybrid Local Search (HLS), introduced by Müller [2009], as the most commonly recommended method among four alternatives. These alternatives included Simulated Annealing proposed by Bellio et al. [2016], an Answer Set Programming approach employing a model developed by Banbara et al. [2019] and solved by Gebser et al. [2012], as well as a fourth method based on a MaxSAT model, coded by Asín Achá and Nieuwenhuis [2014] and resolved by Berg et al. [2019]. This finding not only affirms our interest in HLS but also validates our decision to use it as a baseline for our future algorithm configuration experiments (see Chapter 4 and 5).

Moreover, the work of De Coster et al. [2022] provides an extensive collection of instances, significantly expanding the range of available data for our work by tripling the number of real-world instances and generating thousands of artificial instances. Their contribution in terms of publicly available instances greatly enhances the scope and validity of future investigations.

Before directly using the instances generated by De Coster et al. [2022], it is crucial to recognize that these instances were generated in the first place to behave similarly to real-world instances while improving diversity. The training and testing sets are designed to encompass a broad spectrum of problems in order to test solvers on diverse problems. As a result, some instances may differ significantly from real-world instances. In the context of automatic configuration, training, and testing sets must share similar characteristics.

This chapter focuses on an analysis of the instances generated by De Coster et al. [2022]. The first part describes the process employed to generate these instances. Subsequently, we focus on an examination of the dissimilarities observed in terms of instance features. Additionally, we explore the concept of *feasibility*, which concerns all types of instances. In this thesis, an instance is deemed feasible if, across 10 seeds and within a specified time limit matching the solver constraints (for CB-CTT 5 minutes), the constructor proposed by Müller [2005] successfully generates at least one solution. This notion of *feasibility* acts as a proxy for assessing the resemblance between the training and test sets.

Then we present work that develops prediction models that form a part of a pipeline designed to efficiently predict the *feasibility* of the CB-CTT instances. Furthermore, we construct a selector, which serves as an additional filtering mechanism to refine the artificial CB-CTT instances. Through these efforts, our objective is to establish a robust framework for predicting feasibility and enhancing the quality of the generated instances.

3.2 Artificial Instance Generation

In Chapter 1, an introduction is provided to the three benchmarks employed in the study: the ITC (International Timetabling Competition), New Real-World, and Artificial benchmarks. The ITC benchmark comprises 21 instances, originally presented during the International Timetabling Competition held in 2007. These instances served to establish the CB-CTT (Curriculum-Based Course Timetabling) problem as a benchmark widely recognized within the research community. The remaining two benchmarks, New Real-World and Artificial were proposed by De Coster et al. [2022].

The combined set of real-world instances amounts to 82, encompassing a wide range of scheduling challenges faced by universities and institutions. The real-world benchmarks, including both the ITC and new real-world instances, accurately represent the scheduling problems encountered by universities. These instances effectively capture the challenges involved in transforming real-world scheduling problems into CB-CTT format, making them a reliable and practical basis for evaluating and addressing these complex issues. This set of instances serves as the standard on which the solving methods must demonstrate superior performance, and as such, it is commonly used as the testing set for machine learning in this field.

De Coster et al. [2022] work is part of a field of research that involves finding increasingly complex and efficient ways of generating complementary instances for given problems. Instance generation is based on instance space analysis, for example, i.e. the ability to represent instances in a space and thus define them more easily in order to create new ones at a later stage (Smith-Miles and Bowly [2015], Muñoz and Smith-Miles [2017]). De Coster et al. [2022] uses the results of two research papers: Lopes and Smith-Miles [2010, 2013], which created and improved the instance generator, even though De Coster et al. [2022] has tuned this generator to use the features they selected.

In De Coster work, all real-world instances help to generate artificial ones, ensuring an adequate volume of data for machine-learning methods while also validating the robustness of the results. The artificial set consists of 6942 instances generated by De Coster et al. [2022]. The rest of this section provides a brief overview of their process to facilitate understanding of the subsequent analyses. The primary objective of their study was to create a diverse set of instances for automatic algorithm selection. Their aim was twofold: to generate instances that closely resemble real-world scenarios for effective learning and to introduce instances with unique characteristics that expand the instance space.

The instance space refers to a spatial representation of the instances, and in their work, De Coster et al. [2022] utilized Principal Component Analysis (PCA) on the instance features to position the instances within this space. Now, let us examine the protocol followed by De Coster et al. [2022].

Algorithm 7 Artificial Instances Generation Procedure

1: $Features \leftarrow Extract(Instances)$
2: $Principal\_Component \leftarrow PCA(features)$
3: $Estimator \leftarrow Gaussian\ Estimation(Principal\_Component)$
4: $NewPointsPCA \leftarrow sample(Estimator)$
5: $NewPoints \leftarrow Transform\ Back(NewPointsPCA)$
6: $NewInstances \leftarrow generatorInstance(Newpoints)$

The procedure for generating artificial instances detailed in Algorithm 7, as highlighted by De Coster

et al. [2022] in their work, involves the following steps:

1. Feature Extraction: Initially, the relevant features are extracted from the 82 real-world instances. Table 3.1 details these features.

2. Principal Component Analysis (PCA): The extracted features undergo Principal Component Analysis, a technique designed to reduce the dimensionality of the feature space while capturing the most significant patterns and variations in the data. PCA offers several advantages, particularly in creating a reduced number of components that synthesize the behavior of data features and their interrelationships. By identifying a set of principal components, PCA effectively explains the majority of the variance in the feature space. These components represent new dimensions that hold greater relevance for the data because each component includes a significant portion of the behavior among all the extracted features.

3. Gaussian Estimation: An estimator, like a Gaussian distribution, is fitted to the principal components derived from the PCA. Kernel density estimation is then applied to identify the closest Gaussian distribution within the distribution of real-world instances on the new principal components. By using Gaussian kernels in the kernel density estimation, we can construct an estimator that not only generates solutions with closely aligned values within the PCA-generated space but also enhances solution diversity. The Gaussian distribution proves valuable for creating solutions, as it incorporates a density that diminishes as one moves farther away from the origin points.

4. Sampling: The estimator generates nearly 7000 points based on its computed distribution. These points have coordinates in PCA space.

5. Spatial transformation of original features: The generated or artificial points in the reduced PCA space are transformed back to their original feature space using a specific procedure. This involves repeating the matrix defined during the initial PCA step and applying its inverse to the generated data. Through this process, the artificial instances are converted to the original feature representation, resulting in a list of meaningful features for each generated instance.

6. Generation of new Instances: Finally, the newly generated points in the original feature space are passed as inputs to a generator function. This generator function leverages the information from the new points to create the desired number of artificial instances. These new instances possess characteristics similar to the original instances while also introducing new variations and filling the instance space.

By following this procedure, De Coster et al. [2022] were able to generate a large set of 6942 artificial instances. This approach allowed them to create real-world-like instances for effective learning and algorithm selection while expanding the instance space and exploring new instance characteristics.

De Coster et al. [2022] already remove obvious infeasible instances. That involves the instances where `Lectures` is greater than `Space` and where the *number of allowed Timeslots* for one course is fewer than the number of *Lectures* of this course.

Table 3.1: List of Features used by De Coster et al. [2022] to generate instances. Feature names and meanings are specified in columns 1 and 2 respectively. The third column shows whether the features are used for the prediction presented in this chapter.

Feature	Definition	Used
Lectures	The number of lectures. This value helps define the size of the problem.	✓
Courses	The number of courses that defines how groups lectures.	✓
Rooms	The number of rooms. This value helps define the size of the problem.	✓
Teachers	The number of teachers. The larger the number of teachers in relation to the number of courses, the better, as this reduces scheduling constraints.	✓
Curricula	The number of curricula. In general, the larger the number, the more complex the problem.	✓
Working Days	The number of days. This value multiplied by Periods per Day gives the total number of timeslots.	✓
Periods per Day	The Periods per day value corresponds to how a day is divided into timeslots. This value multiplied by Days gives the total number of timeslots.	✓
Unavailability	The number of unavailability constraints. Recall one corresponds to a timeslot where a teacher is unavailable.	
Lectures / Course (Min & Max)	Extrema number of lectures per course. This lets you know whether a subject with certain constraints, such as RoomStability, will be easy to meet.	
Courses / Teacher (Min & Max)	Extrema number of courses per teacher provide insights into complexity. A higher number of courses assigned to a teacher should increase complexity, because the number of lectures per teacher increases.	
Courses / Curriculum (Min & Max)	The maximum and minimum number of courses per curriculum serve as indicators of curriculum composition. When considering a fixed number of curricula, these values reveal whether one or a few curricula include a lot of courses or if the distribution is uniform.	
Room Size (Min & Max)	The size of the largest and smallest rooms allows us to assess whether all the rooms share identical dimensions, which can facilitate compliance with RoomCapacity requirements.	

3.3 Analysis of Instances

This section presents two analyses of instances from different perspectives, focusing on their instance features. The first analysis compares the characteristics of real-world instances with De Coster et al. [2022] artificial data, providing insights into their similarities and differences. The second analysis assesses the feasibility of instances based on their grouping. The objective is to examine and analyze the instances, gaining a deeper understanding of their properties and implications.

3.3.1 Real-World versus Artificial Instances

This section presents an analysis of instances, specifically focusing on the comparison between Real-World and Artificial CB-CTT instances. To conduct this analysis, we utilize the instance features discussed in Chapter 2. These features serve as descriptive attributes that can be extracted directly from the problem files without requiring any computational time, except for the time taken to read the files. By leveraging these instance features, we gain valuable insights into the characteristics and properties of the instances under investigation.

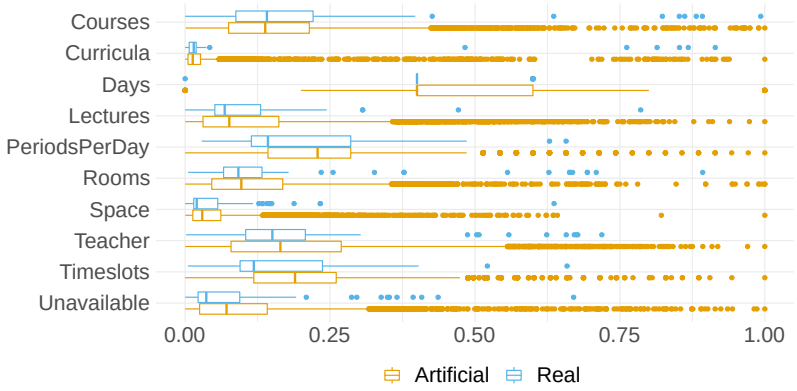


Figure 3.1: Normalized feature distributions for two sets of CB-CTT problems.

Figure 3.1 provides an overview of the distribution of instance features, as described in Section 2.2.3.1, for both real-world and artificial instances. The distributions appear to be similar between the two sets of instances, indicating that the artificial instances have been successfully generated to resemble real-world instances. However, it is worth noting that the distribution of features for the artificial instances is more spread out compared to the real-world instances. This observation can be attributed to the characteristics of the Gaussian distribution estimator used, which allows for the generation of more extreme values.

Moving on to Figure 3.2, it presents a visualization of the Principal Component Analysis (PCA) for both real-world and artificial instances. PCA is a dimensionality reduction technique. In our case, we are working with a table consisting of ten columns, where each row represents an instance. That means each instance can be depicted in a 10-dimensional space, with each dimension corresponding to a feature. To perform dimensionality reduction, PCA performs an eigenvalue decomposition of the covariance matrix. This decomposition results in a set of eigenvalues and their corresponding eigenvectors. These eigenvectors, referred to as the principal components,

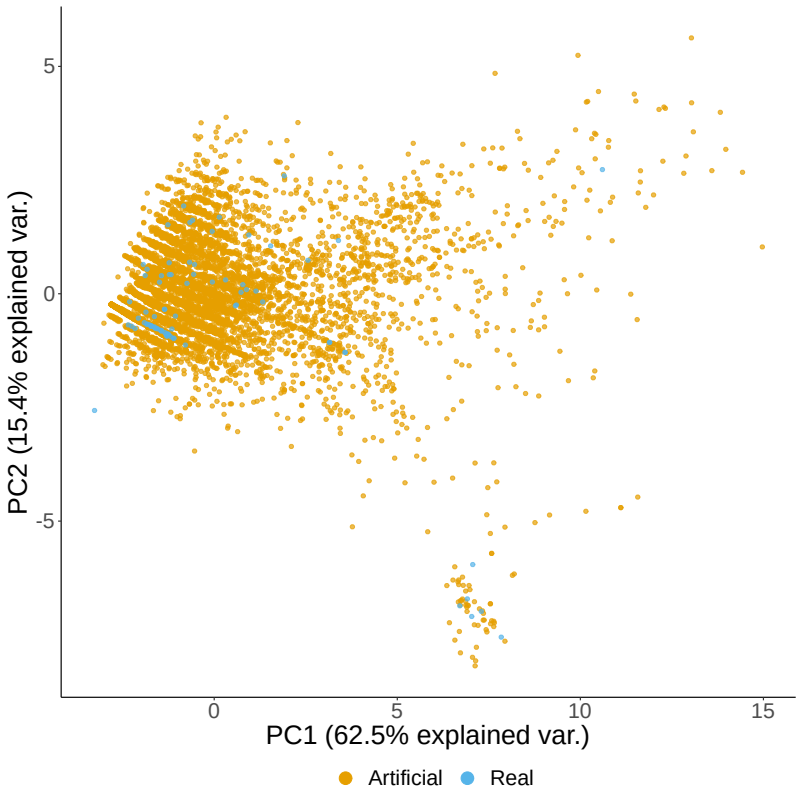


Figure 3.2: PCA Artificial versus Real.

are the key to dimensionality reduction. In our case, two dimensions that represent the most of information in the form of the principal components are kept, allowing for easy visualization while retaining the most important information.

Figure 3.2 demonstrates that the instance generator has effectively created visually similar problems, as the artificial instances fill in the gaps between and around the real-world instances. That indicates artificial instances capture the essential characteristics of real-world instances while introducing additional variations to fill the instance space. The paper of De Coster et al. [2022] specifies that their aim was also to fill the gap in instance space using their generators to diversify and create a selection of robust algorithms for highly divergent instances. Overall, these figures provide valuable insights into the comparison between real-world and artificial instances, highlighting their similarities and the effectiveness of the instance generation process in creating representative and diverse problem instances.

The key finding of the analysis is that the ranges of instance features for both artificial and real-world instances are comparable. This suggests that the generation process of artificial instances has been effective in capturing the characteristics of real-world instances. However, it is important to note that using a Gaussian kernel in the generation process has resulted in creating instances that exhibit greater divergence from the original data. While this diversification is a desired outcome,

it also raises the concern that these new instances might be significantly different from the existing data, potentially posing challenges when using them as a training set for machine learning models.

3.3.2 Feasibility Analysis

Preliminary work consisted of generating several initial solutions for each instance, whether real-world or artificial. For this, the algorithm detailed in Müller [2005] and included in the Hybrid Local Search framework has been employed. These executions revealed that the constructor, i.e. the program that builds the initial solutions, was failing on some instances. This situation presents a challenge as our focus has been primarily on optimizing solutions rather than constructing them. Additionally, it becomes problematic because the objective of this thesis is to train methods to perform well in real-world scenarios. If the generated instances are infeasible, meaning the constructor cannot build initial solutions, it can adversely affect the performance of complex pipelines due to the absence of data and the computational time invested in solving these infeasible instances becomes unproductive.

In the context of our work, a case-specific definition has to be formalized to define what means *feasibility*. An instance is deemed infeasible if the constructor (Müller [2005]) fails to generate an initial solution within 5 minutes. To ensure thorough analysis, runs of the constructor use 10 different seeds for each of the 7024 instances.

So feasibility is a characteristic that means the constructor cannot find a solution that respects the hard constraints (Section 1.4.2.1), in a comparatively long computing time. If an instance is infeasible, then the time allocated to the construction is wasted, so we want to be able to predict this feasibility using instance features.

This scenario presents a concrete application of using generated instances as training sets for machine learning models that predict feasibility. However, as mentioned in the previous section, concerns were raised regarding the similarity of instances. The visualization using PCA revealed instances that appeared highly unusual and distinct. Consequently, for this feasibility analysis, real and artificial instances are treated separately to account for their potential differences in terms of feasibility.

The first part of our analysis focuses on the global distribution of feasibility by type of instances.

Table 3.2: Frequency of infeasible and feasible instances by origin.

Artificial		Real	
Feasible	Infeasible	Feasible	Infeasible
4370	2572	77	5

Table 3.2 shows the frequency of each feasibility by type of instances: real-world and artificial, the generated instances. Thus, Table 3.2 shows that only 6% of the real-world instances are infeasible. That proves the infeasibility is not only due to the generator. Moreover, Table 3.2 shows that, contrary to the Real-World instances, 38% of the generated instances are infeasible. So the proportion is much higher. There are two possible causes. The first one is that the generator generated a high number of infeasible using infeasible real-world instances. However, Section 3.2 details the process followed, so that makes this option impossible when using a Gaussian kernel because proportions would be the same. The second is that the generator creates new instances that are

also infeasible. But in this case, infeasibility is different than for real-world instances. Because it is due to their creation rather than their difficulty to solve. In conclusion, this shows that there is a notable difference in feasibility distribution between real-world and artificial data. In this thesis work, we study whether feasibility is related to instance features, which could pave the way for prediction.

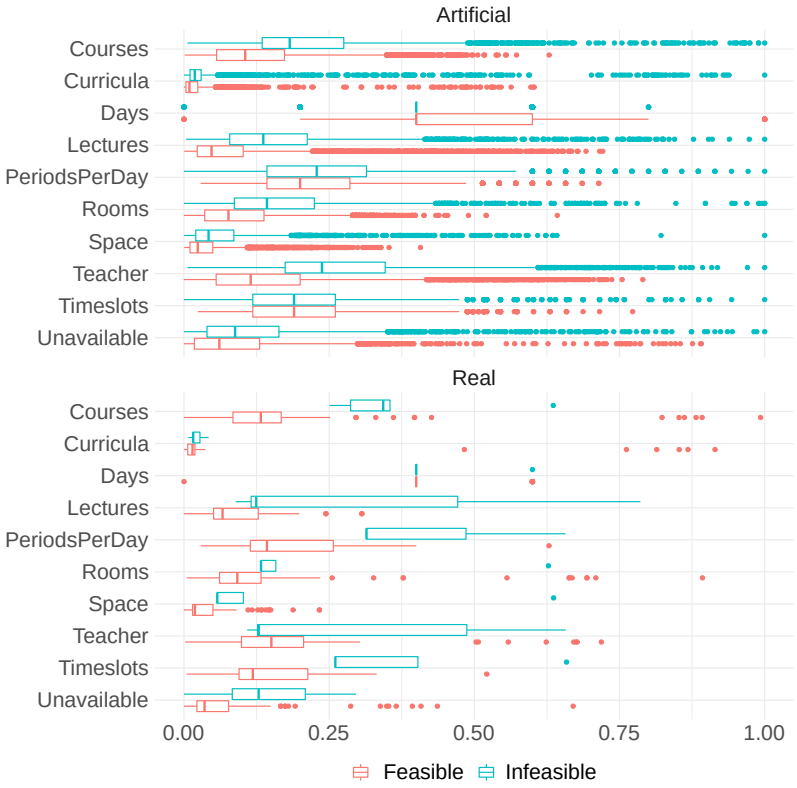


Figure 3.3: Scaled instance features distribution for the Artificial and Real-World sets and by class of feasibility

Figure 3.3 shows the distribution of the values of instances features for real and generated instances comparing also feasible and infeasible ones. Values of instance features are scaled by the MinMax standardization process. That helps to highlight the difference between feasible and infeasible. The scaling is done on all instances and values are, then, split by type of instances and feasibility.

When comparing feasible versus infeasible instances, Figure 3.3 shows different distributions. However, we see that the boxplots are visually the same for some instance features. The first boxplot of Figure 3.3 focuses on generated instances. The latter shows infeasible instances seem to have higher values for all features, except for Days and Timeslots. These observations are confirmed by several Wilcoxon signed-rank tests with a threshold of 5%. This test is performed as a two-sided test. They compare feasible and infeasible artificial instances for each feature.

Wilcoxon results show that, except for Timeslots, distributions are all significantly different. Thus, feasibility or infeasibility varies according to our selected instance features. This result implies that a feasibility prediction model may perform well on this set. Moreover, the visual difference observed for the real-world CB-CTT instances is more noticeable. That could be due to the low frequency of infeasible instances in this set as shown by Table 3.2. For example, the Courses or Rooms distributions do not overlap at all. On the contrary, the generated instances show a slight change in the distribution. Contrary to the differences noted on the second Figure 3.3 boxplot, Wilcoxon tests find Curricula, Days, Teacher, and Unavailable as significantly similar. So despite a more noticeable visual difference, Wilcoxon tests assume that the samples are significantly similar because the volume of real-world data is small, whereas Wilcoxon tests take this small sample size into account and therefore more easily assume that the data are similar.

In summary, looking at the distributions and the Wilcoxon tests, constructing a model seems to be highly possible given the divergent behavior. This model could be a random forest. Indeed, a random forest can create efficient classification rules if some features appear to be discriminant. If Wilcoxon tests denote a difference, the random forest should use these significant differences to improve prediction power. Perhaps this type of model has more difficulty with artificial data because the difference is less pronounced.

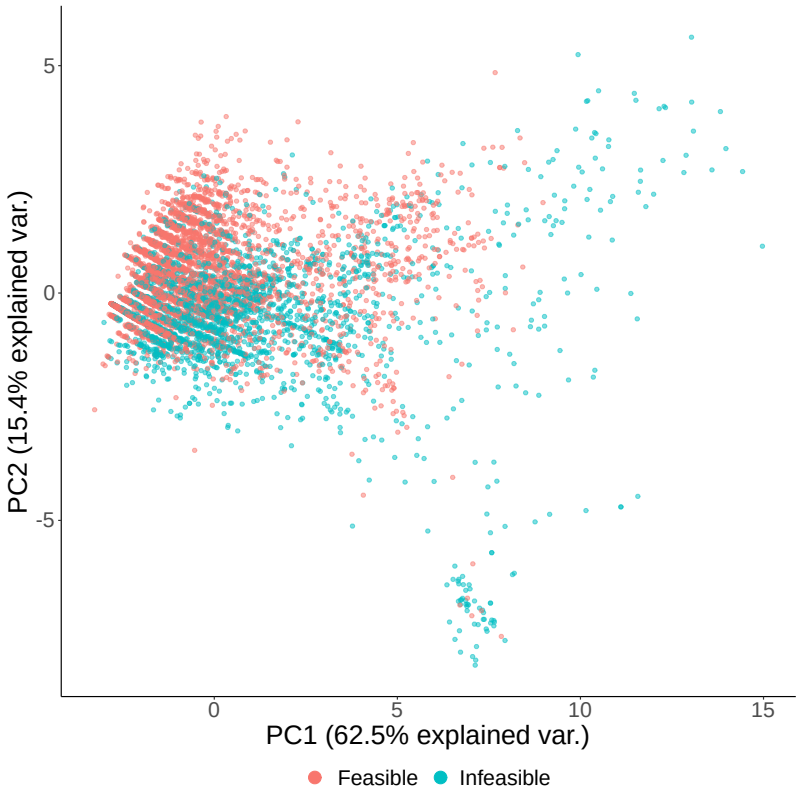


Figure 3.4: Principal Components Analysis: Feasible versus Infeasible

As explained in Section 3.2, the analysis of the distributions considering the instance features is useful to have a first comparison to see independent possible discriminating features. In addition, it is helpful to use a visualization provided by PCA to see if infeasible instances show a global pattern. PCA considers and offers a visualization using all features at the same time taking into account their interdependence relation. However, visualization using a PCA is only one way of representing the data. Ideally, the space should be displayed using all the dimensions directly, without going through a PCA, which is a tool for representing part of the information on the original dimensions.

Figure 3.4 shows the percentage of information represented by each axis or component. The representation score is the sum of both axes which amounts to 78%. Figure 3.4 shows several feasible and infeasible instances overlapping in a cluster of points more in the center. However, many infeasible instances are located in the right part of the figure, away from the cluster. That confirms the instance features surely discriminate and therefore may be used to predict the feasibility.

The analysis of feasibility has provided valuable insights into the characteristics and predictability of instance viability. By examining the distribution of feasibility among real-world and artificial instances, as well as exploring the relationships with instance features, we have gained a deeper understanding of the challenges and opportunities in constructing feasible solutions. The findings highlight the importance of distinguishing between real-world and artificial instances, as their feasibility distributions can vary significantly. Moreover, the visualizations and statistical tests have demonstrated that certain instance features exhibit discriminative behavior in determining feasibility. That opens up possibilities for developing prediction models that can effectively assess the viability of instances and guide decision-making processes.

3.4 Feasibility Prediction

In this section, we focus on the feasibility prediction in the context of our work. Building upon our previous analysis, which demonstrated the discriminative nature of instance features (Section 2.2.3.1), in determining feasibility, we now turn our attention to developing a predictive model. Our analysis revealed distinct distributions and patterns in the values of instance features for feasible and infeasible instances, indicating their potential as predictors of feasibility.

Why do we aim to predict feasibility? There is one logical reason. Firstly, it helps us save computing time. Imagine spending five minutes on ten runs trying to find a solution, only to discover that the constructor cannot create it. Predicting feasibility avoids this scenario, ensuring that computers allocate more time to resolution. However, the primary motivation for employing a machine learning process to predict feasibility is to identify artificial instances that are close to real-world instances. Feasibility is employed as a measure of instance similarity because it serves as a quantifiable behavior that appears to be influenced by the characteristics of the instance itself.

We have selected the random forest model as the classifier for predicting instance feasibility. This type of model is well-suited for binary class forecasts, making it a good choice for our task. Random forests have demonstrated strong performance in various domains and are widely used in the literature for instance characterization and even automatic algorithm selection (De Coster et al. [2022]). Leveraging the benefits of a set of decision trees, random forests offer robustness, scalability, and the ability to handle high-dimensional feature spaces.

One drawback of this method, and many other models, is its difficulty in properly handling an unbalanced dataset. That corresponds to a dataset where one class largely outnumbers the other. That is why all models, in this paper, are trained on a balanced dataset obtained via undersampling, i.e., considering a smaller number of the over-represented class. We also tested using only

stratified sampling but the performance was not good. In summary, an algorithm selects randomly an equal number of instances of the two classes we want to predict. It is this set of an even 50-50 split that is used for the training set.

In this case, we chose the under-sampling, i.e. we reduced the majority class to have the same number as the minority class. The major disadvantage of random sub-sampling is that it may discard potentially useful data that could be important to the induction process. Another possible solution would have been oversampling, i.e. generating solutions to increase the volume of data available for the minority class. A popular oversampling technique is SMOTE (Synthetic Minority Over-sampling Technique) (Chawla et al. [2002]), which creates artificial samples by randomly sampling the characteristics of the minority class occurrences. A disadvantage of these methods is the creation of a bias for the model in some cases. Moreover, the creation of multiple samples in the minority class can lead to model overfitting.

3.4.1 Feasibility Models

The primary objective of this section is to develop robust prediction models that accurately determine the feasibility of instances. By achieving this goal, we desire to effectively filter out infeasible instances, optimizing our resource allocation and computational efficiency. The models undergo testing using real-world data, ensuring their ability to accurately predict feasibility for future real-world instances. In total, three models are created, each denoted as FP^X, representing feasibility models trained on a specific subset of instances denoted as type X.

Three distinct models are developed to predict feasibility, enabling a comparative analysis of their predictive power. The first model, referred to as FP^R, is trained using a dataset comprising 82 real-world instances. Although this dataset is relatively small, it serves as our reference point for comparison with the other models and is not specifically designed as a feasibility model.

The second model, named FP^A, is trained exclusively only with artificial instances.

The third model, denoted as FP^{AC}, also utilizes data from the generated instances. However, unlike FP^A, it is trained on a carefully selected subset of artificial instances.

Table 3.3: Table of Trained Models, Training Sets, and Objectives

Model	Training Set	Goal
FP^R	Real-world instances	Reference model
FP^A	Artificial instances	Naive model
FP^{AC}	Selected subset of artificial instances	Improved model

Table 3.3 summarizes the three models used in the following sections, their training set, and their goal. Although the table makes it explicit that the models are built on instance sets, it should be noted that the sets go through the under-sampling algorithm described in the introduction to the Section 3.4. Thus, these models are slightly different at each seed tested, as the under-sampling does not select the same instances for training as indicated in Table 3.3.

In addition, all models are evaluated on real-world instances. Since FP^R uses an under-sampled percentage of real-world instances, its testing set is reduced, as testing and training do not need to have any data in common.

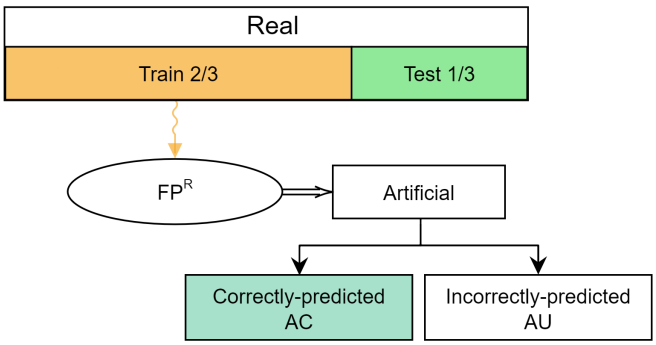


Figure 3.5: Procedure for the Selection of Generated Instances. The model FP^R is trained and then used to predict the feasibility of artificial instances. Instances whose feasibility has been well predicted are in the AC set.

Generated Instance Selection Section 3.3.2 presented an analysis of the instances that showed that the behavior of the generated instances appeared to be different, in terms of feasibility, from real-world CB-CTT. Given the generation process, detailed in Section 3.2, we consider that some generated instances are like real data, in terms of feasibility. Therefore, we aim to be able to identify those real-like generated instances.

Figure 3.5 summarizes the process of artificial instance selection. Figure 3.5 shows that the whole process begins with the training of model FP^R on 2/3 under-sampled of real-world.

The second step involves using the FP^R model to predict the feasibility of the artificial instances. Subsequently, we identify and retain the artificial instances that are accurately predicted by FP^R. Figure 3.5 highlights that the correctly-predicted artificial instance set is called AC. On the other hand, the other artificial set is called AU and is not used anyway. Once the set has been defined,

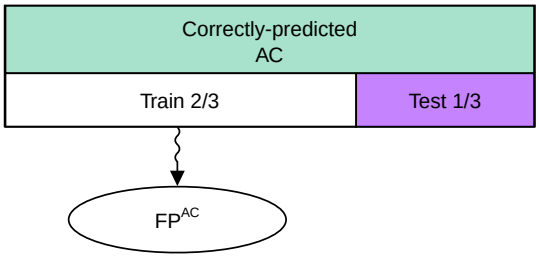


Figure 3.6: Training a Feasibility Model with Carefully Selected Generated Instances.

we can create the model FP^{AC} which, after under-sampling, is trained on the AU set. Figure 3.6 shows this process and, together with Figure 3.5, constitutes the complete process for creating the model FP^{AC}.

Our hypothesis is straightforward: if a generated instance is well-predicted by a model specifically designed for real-world instances, then its feasibility should be similar to that of real-world instances. Based on this premise, the set of artificial AC in Figure 3.5 is saved. This AC set selects

approximately 58% of the generated instances, which amounts to 4446 problems.

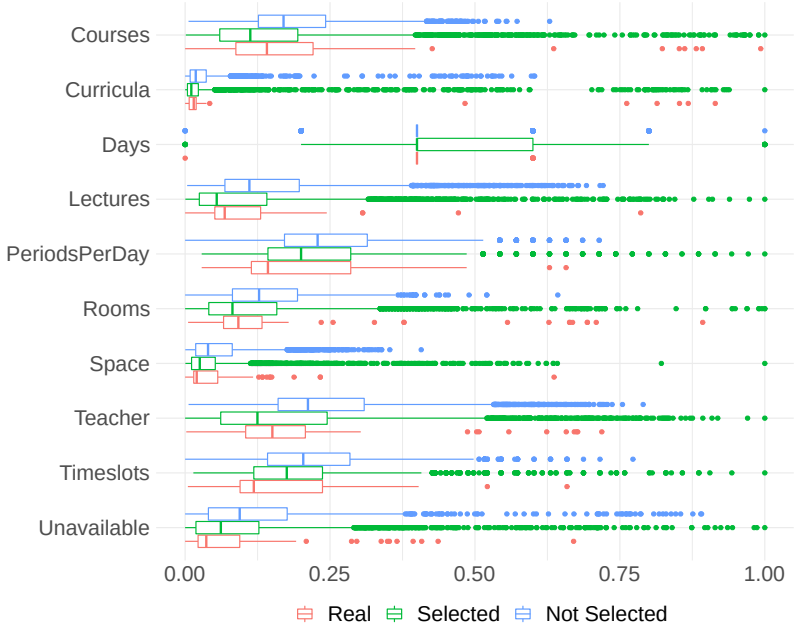


Figure 3.7: Scaled instance features distribution for the selected and non-selected Artificial and Real-World sets and by class of feasibility

Figure 3.7 provides an overview of the distribution of instance features. Upon examination, Figure 3.7 reveals no significant difference between the two sets of artificial instances: the selected and non-selected subsets, except that the distributions es not selected are less spread. In other words, there are fewer values outside the boxplots. While slight shifts in the distribution between the unselected artificial and real-world instances are observed, the selected seems to align more closely with the value space. However, this figure alone does not enable us to draw any conclusions regarding the performance of the prediction models.

The analysis carried out so far does not indicate any particular behavior of AC, the selected artificial instances, compared to the unselected set or AU within the set of selected instances. Despite this, we proceeded to train a prediction model using these selected instances. The resulting model of the set AC, FP^{AC} shown in Figure 3.6, is now ready for testing.

3.4.2 Models Evaluation

Finally, we trained three models. This part contains details about their evaluation of their ability to predict the feasibility of real-world instances. FP^{A} and FP^{AC} are evaluated 50 times with different subsets of their training sets. Recall the under-sampling process; these subsets are balanced to have the same number of infeasible and feasible instances. To obtain different training sets, different seeds are tested and the under-sampling is rerun. Using different training subsets helps to test the robustness. Then, the accuracy corresponds to the proportion of real-world instances well-predicted.

The evaluation of the FP^R model involves two distinct cross-validation techniques. First, a 3-fold cross-validation process is employed, where 2/3 of the available real data are utilized for training, while the remaining 1/3 is set aside for testing. However, due to the limited number of real instances, the model is also subjected to leave-one-out cross-validation. In this approach, the model is trained on all real-world instances except one, which is subsequently used for testing. These two evaluation methods serve specific purposes. Leave-one-out cross-validation is particularly advantageous when working with small training and test sets, as it involves testing with just a single instance. On the other hand, 3-fold cross-validation provides more realistic results by testing the model across multiple groups. For FP^R, both evaluation techniques are employed due to the scarcity of real-world data. While leave-one-out is practical for small datasets, 3-fold cross-validation allows for comparisons with other methods and offers a comprehensive assessment of the model performance.

Table 3.4: Model Performance Metrics on real-world problems.

Model	Accuracy		Sensitivity	Specitificity
	Train	Test		
FP^R 3-Fold	0.81	0.78	0.67	0.9
FP^R LOOCV	0.75	1	1	1
FP^A	0.82	0.36	0.33	0.6
FP^{AC}	0.77	0.77	0.76	0.93

Table 3.4 contains the average details of each model. Moreover, it also includes the average accuracy of the learning. Accuracy represents the proportion of instances whose feasibility has been correctly predicted out of the total number of instances to be predicted.

Sensitivity, also known as True Positive Rate or Recall, is calculated as follows:

$$Sensitivity = \frac{True\ Positives}{Actual\ Positives}$$

This metric quantifies the number of positive instances correctly predicted as positive, divided by the total number of actual positive instances. In our context, an instance is classified as positive if it is considered feasible.

Specificity, often referred to as True Negative Rate, is computed as follows:

$$Specificity = \frac{True\ Negatives}{Actual\ Negatives}$$

It measures the number of negative instances correctly predicted as negative, divided by the total number of actual negative instances. In our scenario, an instance is categorized as negative when it is identified as infeasible.

Firstly, Table 3.4 shows that FP^R has good accuracy. Notably, 3-fold cross-validation reduces the sensitivity of FP^R. That is probably due to the low number of instances in the real-world test. We hypothesize that cross-validation is definitely not suitable for evaluating this model. Up to this point, only the Leave-One-Out cross-validation method has consistently shown the effectiveness of FP^R. It is worth noting that this model serves as our reference for predicting feasibility in real-world scenarios. The confirmation of our hypothesis and the analyses of the discriminating power

of instance characteristics by at least one cross-validation method led us to further evaluate the other models.

Table 3.4 shows that FP^A presents a high accuracy for the training test. That means FP^A predicts efficiently the feasibility of a problem if it is artificial. The accuracy of the training is good, above 0.8. Unfortunately, FP^A fails to predict real-world feasibility. That confirms a difference in the behavior or profile of these two sets of benchmarks, real-world versus artificial, despite the process used by De Coster et al. [2022]. Indeed, the accuracy of this scenario is 0.36, which means that a random classification would be more efficient. This behavior is logical if we take into account the difference between the behaviors noted during the analyses.

Now let us look at model FP^{AC} using generated instances. Table 3.4 clearly shows that the model obtains a good accuracy, compared to the other model. Indeed, it has a score of 0.77. The one trained on the real instances is much worse than the real one, when evaluation is done with LOOCV, but proves that we can use some of the generated data to predict the behavior of real-world instances. The performance of FP^{AC} is better than FP^R if we compare with the same cross-validation method, specifically in terms of sensitivity.

The conclusion of this part is the success of our model using selected generated instances. One of the next steps would be to create a finer selection to reach a model as efficient as FP^R. Even if we have the accuracy of the models and we have verified that their confusion matrix is balanced, we look more deeply into the three models to understand the success or not of these models.

3.4.3 Prediction of Selected Artificial Instances

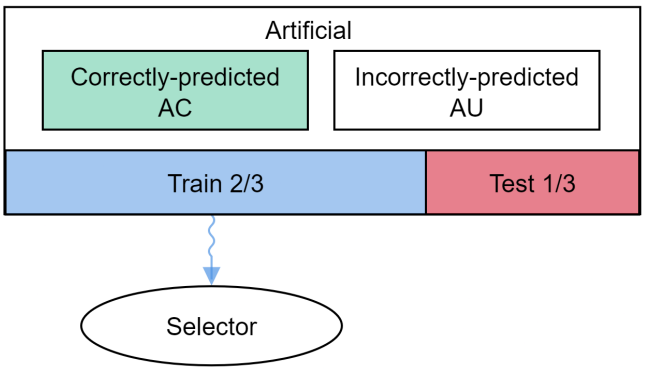


Figure 3.8: Process for constructing the selector of generated instances.

Section 3.4.1 has demonstrated the strong predictive performance of FP^{AC}. Recall that the instances used for selection were initially identified using FP^R.

In the subsequent part of this section, we propose a model designed to determine whether an artificial instance should be selected or not. This model exclusively utilizes the information encapsulated within the instance features.

The motivation behind such a model is to eliminate the need for computing the feasibility of all generated instances and relying on another model. Instead, having an initial artificial data sample

categorized into two classes, namely *selected* and *not selected*, would be sufficient. This approach streamlines the selection process and reduces dependence on additional models for determining the instance state of selection.

As a reminder, the previous section presented a comparison between the selected and not selected instances, revealing discernible differences. This divergence in behavior was already highlighted in Figure 3.7. Consequently, we can leverage the instance features to develop a selector that chooses if a new instance is selected to be part of the train test.

To build the dataset we use FP^R that tells which artificial instances to choose according to the prediction (Section 3.4.1). If an instance is well-predicted then it is selected. Following this process, we create a dataset of artificial instances. Then we create a balanced training set of selected or not selected that represents almost 66% of data. The *Selector* is then created and is a random forest model that is specifically trained to perform the instance selection task.

The process for training the selector construction pipeline is illustrated in Figure 3.8, providing a visual representation of the steps involved. This pipeline involves utilizing the predictions from FP^R, generating a balanced training set, and training the *Selector* to effectively discern between selected and non-selected instances based on their distinctive features.

The *Selector* performance is evaluated using a 3-fold cross-validation. The cross-validation procedure returns a sensibility of 0.73, a Specificity of 0.86, and an Accuracy of 0.82. Thus, we can validate this model for use in future contexts. The fact that the *Selector* performance is good confirms our analysis in Section 3.4.1 where we noticed a pattern difference between distributions of the instance features of selected or not selected generated instances.

Hence, *Selector* demonstrates its effectiveness in efficiently identifying instances that exhibit similar characteristics. In the following section, we implement the comprehensive process that encompasses our two models: *Selector* and FP^{AC}. The former is responsible for selecting instances based on its predictions, which are then used as input for the latter model. The objective is to test the scenario where *Selector*, identifies and selects instances, followed by training a feasibility prediction model, FP^{AC}, solely on the selected instances. This approach aims to verify that any errors made by each individual model do not accumulate or compound during the selection and feasibility prediction process. By ensuring that the errors of the two models do not amplify each other, we can enhance the overall robustness and reliability of our predictions.

3.5 Pipeline Validation

To summarize the work already presented in this chapter, the objective of this chapter was initially to assess the similarity between artificial and real-world instances. Preliminary findings indicated a degree of similarity, despite the influence of using a Gaussian kernel for instance generation. However, initial executions revealed that certain new instances (De Coster et al. [2022]) appeared to be infeasible, making it exceedingly challenging to generate an initial solution that satisfied the four hard constraints.

Recognizing the significance of identifying this *feasibility* aspect, we attempted to predict it by leveraging the instance features. By using instance features, we aimed to develop a predictive model that could discern the feasibility of instances more effectively. The most efficient model uses a selected subgroup of artificial instances. Thus we have also proposed a model that selects the generated instances for this model. This means that by creating a pipeline to predict the feasibility of real-world instances, we have created a way to automatically select artificial instances in the

future that are considered more similar to the real-world data. Both models use the same instance features as explicable features and perform well.

This section presents a pipeline that includes two models in sequence. Then it evaluates the feasibility prediction of the whole pipeline. The objective of this pipeline is to show that we can use this pipeline as is in future work to create a filter that remains efficient despite two models in sequence.

The created pipeline is described by Figure 3.9. In order to obtain statistically robust results, the pipeline is executed 30 times.

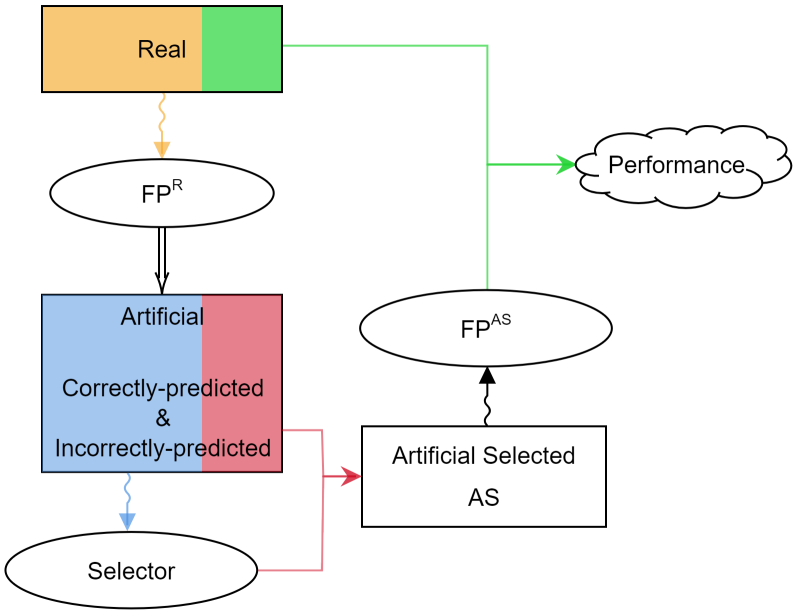


Figure 3.9: Feasibility Pipeline. FP^R is first trained on Real. Then, FP^R is employed to generate two types of artificial instances Correctly-predicted and not. After that, *Selector* is trained using a part of Artificial to select remaining artificial instances. Finally, artificial instances selected by *Selector* serve as the training dataset for FP^{AS}, which is in end evaluated using the remaining real-world data.

Firstly, the feasibility model FP^R is trained on a balanced training set derived from real-world instances. Subsequently, this model is employed to select the generated instances, following the procedure outlined in Section 3.4.3. By running FP^R on all the artificial data, instances whose feasibility is accurately predicted are designated as *selected*, while others are deemed *not selected*.

For each generated instance, we obtain its corresponding category: selected or not. *Selector* is then trained using a balanced training set constructed from this selection data frame. Then, it is tested on the remaining instances.

The generated instances predicted as selected are set aside to form the training set for the

final feasibility prediction model: FP^{AS}. Finally, this model is tested on the real-world instances not used at the beginning by FP^{R}. This entire process is repeated 30 times, with each iteration involving two balanced training sets: one for FP^{R} and another for *Selector*. By repeating the iterations, we assess the robustness of the pipeline.

Ultimately, the performance of the feasibility prediction using this pipeline is measured in terms of mean sensitivity (0.74), specificity (0.80), and accuracy (0.74). These metrics provide an average assessment of the pipeline effectiveness.

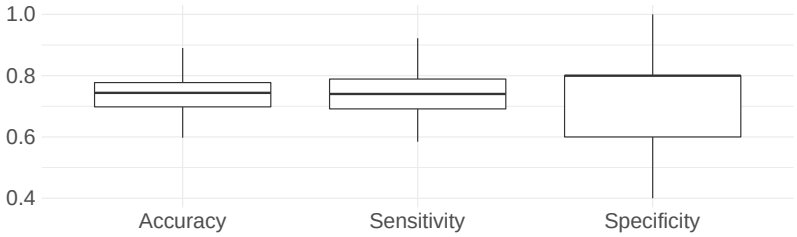


Figure 3.10: Performance of feasibility prediction at pipeline output.

Figure 3.10 shows the distribution of the three performance prediction metrics among the 30 repeats. Specificity concerns the class of infeasible instances. The pipeline, which connects and chains the models together, delivers good results, reaching at least 0.74 in all three metrics. This indicates its ability to predict feasibility.

A way to enhance the pipeline performance could consist of the incorporation of additional instance features, drawing inspiration from the work of De Coster et al. [2022]. This approach aims to leverage the benefits of augmenting the feature set in order to improve the predictive performance of the pipeline.

3.6 Conclusion

In this chapter, the primary objective was to investigate whether the instances generated by De Coster et al. [2022] could serve as viable training sets for configuration. We initiated our analysis by contrasting and examining these two sets. The evaluation of the instance features revealed that the artificial instances exhibited significant variability and occupied distinct regions within the instance space. This characteristic could pose challenges when aiming to achieve real-world performance.

Additionally, our initial experiment highlighted the prevalence of infeasibility as a concrete obstacle. Consequently, it became imperative to swiftly identify and exclude infeasible instances, both artificial and real.

After conducting a comprehensive analysis to identify patterns among the infeasible instances, we opted to predict their feasibility. The underlying idea behind this decision was to develop a model that could act as a filter for future instances while also assessing the similarity between the training data (artificial instances) and real-world instances. By leveraging feasibility as a proxy, we aimed to establish a connection and measure the degree of similarity.

Regrettably, the FP^{A} proved ineffective in its current state. As a result, we embarked on training a model directly on real-world instances to identify which artificial instances demonstrated better feasibility prediction performance. These instances were deemed to bear closer resemblance to real-world scenarios compared to others.

Upon training the model using these selected artificial instances, we observed significant improvements in the results. This underscores the importance of carefully selecting artificial instances before utilizing them directly.

As part of our future work, we are developing a model called *Selector* that employs instance features to identify artificial instances exhibiting similar feasibility patterns, then it says if an instance is selected or not.

To validate our approach, we performed a sequential execution of all these tasks to mitigate any cumulative errors arising from *Selector* and feasibility prediction models, FP^{AS}. The pipeline results yielded positive outcomes, with all metrics reaching at least a mean value of 0.74.

Furthermore, we envision the possibility of incorporating additional instance features to augment the model and further enhance its performance.

Chapter 4

Iterated Sequential Local Search Framework

Publication:

- Thomas Feutrier, Marie-Éléonore Kessaci, and Nadarajen Veerapen. "Investigating the landscape of a Hybrid Local Search approach for a timetabling problem." GECCO'21 Companion: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM 2021.

Contents

4.1 Introduction

The analysis of methods used for combinatorial problem-solving can shed light on how solvers work. That means investigating how the different components of this method interact, how they collectively provide high-quality results, how they perform individually, and whether combining them results in the overall method achieving the minimum quality provided by the best component, or whether synergy is created and thus adds performance.

This chapter focuses on a state-of-the-art method for solving the problem addressed in this thesis, namely Curriculum-Based Course Timetabling (CB-CTT). This method is Hybrid Local Search (HLS) (Müller [2009]) and was introduced in Chapter 1. This metaheuristic is an iterative method that executes a sequence of three local search algorithms: Hill Climbing, Great Deluge, and Simulated Annealing. Each of these algorithms possesses mechanisms that differ from their basic definitions in the literature. Therefore, these methods are considered part of Hybrid Local Search.

Hybrid Local Search is a method designed and optimized to win the International Timetabling Competition (ITC) in 2007. While it is continually being developed to adapt to more recent editions of the ITC, the method is tuned for the ITC benchmarks provided at that event. This resolution method for CB-CTT can be considered complex, as it is a hybrid approach involving three different local search methods. Therefore, when emphasizing the understanding of the components of this algorithm, questions about the need for such complexity need to be raised. Does Hybrid Local Search need to have these three components in this order? Can the parameters be further tuned? Can all the strategies used in HLS be employed to create a more efficient method? All these questions about the various modifications to be made to HLS lead the work in this thesis to analyze HLS in terms of performance first, but above all, to analyze the structure of the solving method itself. Several papers in the literature, such as Marmion et al. [2013], have explored the generalization of local search methods into a single algorithm that can be instantiated for specific local search purposes. Thus, HLS could be considered as a simple algorithm that iterates over a sequence of local search algorithms. The conceptual abstractions and questions regarding the structure of HLS underscore the need for a framework that enables a straightforward and practical implementation of this generalization. To facilitate this process, a tool that simplifies the instantiation of the generalization and allows for the examination of multiple scenarios is necessary.

This chapter provides an initial analysis of the components of Hybrid Local Search, representing the first step in questioning the complexity of Hybrid Local Search. Additionally, this chapter details how the Iterated Sequential Local Search framework was derived from the Hybrid Local Search method. The latter can instantiate HLS but offers much greater flexibility to create new algorithms using all the original HLS components. Finally, the chapter describes how, concerning the literature, the generalization of local search algorithms has been possible and under which conditions it applied to Hill Climbing, Great Deluge, and Simulated Annealing.

The first section of this chapter presents the preliminary experiments that analyze the individual performances of the local search algorithms contained in Hybrid Local Search. The second section focuses on explaining how it is possible to define and create a framework, Iterated Sequential Local Search, that can instantiate HLS while offering additional flexibility to re-tune parameters and test new structures. Finally, the last section explains the framework code, specific optimization features that have been implemented, and the basic framework that has been used to save time. This is followed by an explanation of how and what parameters are required to run the framework from the command line.

4.2 Preliminary Experiments

In Chapter 1, we provide a detailed exposition of the HLS algorithm. This discussion encompasses its defining characteristics, the local search strategies it employs, and the neighborhood operators it integrates. Chapter 1 provides an understanding of how HLS works factually.

HLS is an approximate solving method that won the International Timetabling Competition in 2007. It remains a major approach in recent works related to the Curriculum-Based Course Timetabling problem. For example, its performance is compared with exact methods and other heuristic approaches in De Coster et al. [2022]. Additionally, HLS serves as a performance baseline when new solvers are proposed (Coşar et al. [2023]). Recent work has even investigated how to improve the performance of Hybrid Local Search by manipulating the selection of neighborhood operators (Song et al. [2021]).

Hybrid Local Search is therefore still a timely method for solving the CB-CTT problem. This section attempts to understand how this method works so well. To this end, the following work aims to study the performance of HLS and compare it to its components, i.e. a version of HLS in which some of its local search algorithms are disabled. This study provides additional information on the success of HLS and how, in terms of performance, the three local search algorithms included in HLS work together and separately. It is worth noting that no parameters are altered in this study, maintaining the same values presented in Chapter 1. Furthermore, this study exclusively employs the ITC benchmark, aligning with HLS design parameters. The objective is to gain insights into HLS performance when applied to datasets for which it was originally tailored, thus providing a comprehensive understanding of its operation in optimal conditions.

4.2.1 Experimental Protocol

This section presents the instances used and the methods evaluated. As introduced previously, the main idea is to analyze the performance of the local search algorithms contained in Hybrid Local Search to question their impact and understand which method contributes most and whether it is necessary to use all three methods.

Chapter 1 details the Hybrid Local Search. The latter is an approximate method that executes a sequence of three local search algorithms during a fixed amount of time, except if it finds a perfect solution, when the algorithm stops and the objective function returns zero. In our protocol, we are keeping the same code provided by Müller [2009]. Thus, Hill Climbing, Great Deluge, and Simulated Annealing remain the same as the original HLS detailed in Chapter 1.

Hill Climbing is a local search method used for pure intensification. In other words, it does not accept non-improving solutions. In the case of Hybrid Local Search, HC accepts improving or equivalent solutions because of neutral acceptance criterion. Consequently, it generally returns the first local optimum found, which has generally a poor fitness value. That is why studying Hill Climbing alone is not meaningful, given its poor results. So, commonly, Hill Climbing is almost always associated with another mechanism. This study combines the Hill Climbing and the Great Deluge (GD) of HLS. Furthermore, since Great Deluge seems to accept more perturbations than Simulated Annealing, then combining it with a pure intensification method could improve its performance ensuring that Great Deluge always begins with an almost local optimum. Thus, this work studies the performance of the algorithm iterating over the sequence HC then GD: HCGD.

The second algorithm investigated is the Simulated Annealing (SA) included in the Hybrid Local Search. Simulated Annealing is commonly known in the field of combinatorial problems

for being a local search method that mixes diversification and intensification and provides good results in general (Hasançebi et al. [2010], Henderson et al. [2003], Rutenbar [1989]). Indeed, this neighborhood-based method can escape from a poor local optimum with its metropolis acceptance. That is why this study examines a Simulated Annealing that iterates on itself according to the HLS stopping criteria. HLS Simulated Annealing has a mechanism that changes its parameters for the next call which should help with the diversification when SA is called iteratively.

Figure 4.1 presents a representation of Hybrid Local Search to show how HLS has been sliced up.

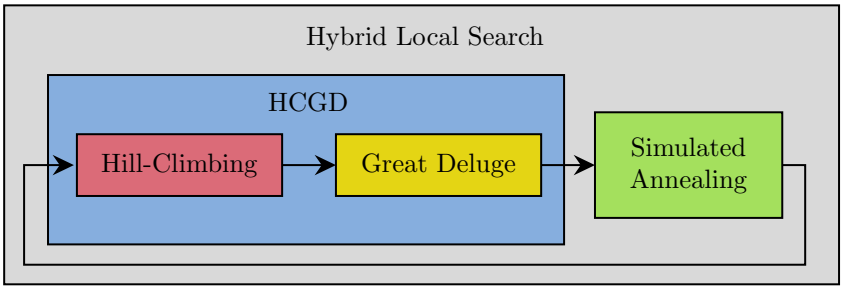


Figure 4.1: Hybrid Local Search and its sub-algorithms: HCGD and SA.

Figure 4.1 shows that HLS can be considered as an iterated method that executes HCGD and then Simulated Annealing (SA). The idea is to deactivate either the HCGD box for studying SA alone or the SA box for studying HCGD alone.

The experiments consist in running the HLS algorithm, the iterated HCGD algorithm, and the iterated SA algorithm on each of the 21 instances of the ITC 2007. These are the instances on which HLS has been optimized and for which it was initially coded. We considered as preferable to test HLS under the conditions for which it was designed, and not to use the other benchmarks. The particularities of these instances are presented in Chapter 1 and in Chapter 3.

Subsequently, 100 runs are performed on the HLS, HCGD, and SA algorithms for each instance of the benchmark instances. Each of the 100 runs employs a distinct random seed, allowing us to calculate average performance metrics. Each seed corresponds to a different initial solution. However, the 100 runs of the three algorithms use the same seeds. These runs were executed on an Intel Xeon Silver 4114 CPU operating at 2.20GHz. The outcome considered for evaluation is the fitness of the best solution reached within a time limit of 300 seconds or 5 minutes.

4.2.2 HLS, HCGD, and SA Performance Results

All the runs performed represent 2100 executions of 300 seconds, i.e. 2100 fitness values returned by the HLS, HCGD, or SA algorithms.

Table 4.1 provides summary statistics of the fitness values for the solutions returned by each of the three methods. Table 4.1 contains, for each instance, the average fitness value obtained over the 100 runs for each algorithm. In addition, Table 4.1 shows the standard deviation of this fitness to give a better idea of convergence. For each row, Table 4.1 puts in bold algorithms that correspond to the best algorithms according to the Friedman statistical significance test (Hollander et al. [2013]). Table 4.1 also has three columns that show the ranking of the three methods to summarize the order for each instance.

Table 4.1: Fitness values of the best solution found by HCGD, SA, and HLS. Bold values mean that the algorithm is statistically better than the others. The rank of each method is provided in the three last columns. The four groups organize together instances with similar these rankings.

Group	Inst	HCGD	SA	HLS	Rank		
		$\text{Mean}_{(\text{sd})}$	$\text{Mean}_{(\text{sd})}$	$\text{Mean}_{(\text{sd})}$	HCGD	SA	HLS
A	01	$\mathbf{5}_{(0)}$	$5.1_{(0.2)}$	$\mathbf{5}_{(0)}$	1	3	1
	11	$\mathbf{0}_{(0)}$	$0_{(0)}$	$\mathbf{0}_{(0)}$	1	3	1
B	02	$72.9_{(14.4)}$	$61.6_{(8.3)}$	$\mathbf{58.2}_{(7.5)}$	3	2	1
	03	$93_{(9.9)}$	$91_{(8.2)}$	$\mathbf{84.9}_{(5.9)}$	3	2	1
	06	$67_{(9.7)}$	$\mathbf{51.7}_{(5.4)}$	$55.5_{(5.9)}$	3	1	2
	07	$32.2_{(9.2)}$	$\mathbf{15.4}_{(3.4)}$	$20.1_{(4.4)}$	3	1	2
	08	$45.7_{(3.2)}$	$\mathbf{42.8}_{(2.5)}$	$44.5_{(2.8)}$	3	1	2
	10	$40.8_{(9.1)}$	$\mathbf{18.9}_{(4.8)}$	$21.5_{(5.1)}$	3	1	2
	14	$61.2_{(4)}$	$\mathbf{59.9}_{(3.6)}$	$60.3_{(3.2)}$	3	1	2
	15	$92.2_{(9.9)}$	$90.3_{(8.2)}$	$\mathbf{85.1}_{(5.8)}$	3	2	1
	16	$57.1_{(8)}$	$\mathbf{38.8}_{(4.9)}$	$41.3_{(5.5)}$	3	1	2
	17	$87.3_{(7.5)}$	$\mathbf{82.1}_{(5.2)}$	$84_{(5.4)}$	3	1	2
	19	$77.7_{(8.8)}$	$\mathbf{70}_{(4.9)}$	$\mathbf{69.7}_{(4.4)}$	3	1	1
	20	$52.1_{(13.3)}$	$\mathbf{31.5}_{(6)}$	$36.5_{(6.8)}$	3	1	2
	21	$118.6_{(10.2)}$	$\mathbf{106.6}_{(6.5)}$	$107.8_{(7.2)}$	3	1	2
C	05	$\mathbf{331.9}_{(13.6)}$	$362.5_{(29.8)}$	$335.9_{(14.1)}$	1	3	2
	09	$\mathbf{109.2}_{(4)}$	$110.3_{(4.7)}$	$\mathbf{109.3}_{(3.7)}$	1	3	1
	12	$\mathbf{345.4}_{(9)}$	$373.1_{(15.4)}$	$350.7_{(8.8)}$	1	3	2
	18	$\mathbf{85}_{(3.8)}$	$92.6_{(5.3)}$	$87.2_{(3.4)}$	1	3	2
D	04	$40.9_{(3.2)}$	$\mathbf{39.8}_{(2.8)}$	$40.9_{(2.6)}$	2	1	2
	13	$73.8_{(3.9)}$	$\mathbf{73}_{(4.7)}$	$74_{(3.8)}$	2	1	2

Set of the 21 instances from the ITC 2007 dataset has been classified into four distinct groups (A, B, C, and D). This classification is based on the performance ranks of the three optimization algorithms: HCGD, SA, and HLS. The aim is to make it easier to read and see the main performance behaviors of the three algorithms.

Group A consists of the two instances that can be considered very easy. In this group, the ranks may not provide significant differentiation due to the similarity in fitness values among the algorithms. Group B includes instances where HCGD ranks third among the three algorithms. These instances are moderately challenging. Group C comprises instances where SA ranks last among the three algorithms. This group represents cases where SA performs the least effectively. Group D covers the remaining two instances, which exhibit their unique characteristics. This classification of instances is not used in the next chapters, but it does highlight the overall behavior, in terms of performance, of the three algorithms.

The first notable result is that HCGD performs badly when no longer associated with SA. Indeed, Group B includes the majority of ITC 2007 instances, which means that it is often worse than the other two algorithms.

SA consistently demonstrates superior performance compared to HLS and HCGD across most of instances. Iterated Simulated Annealing of HLS emerges as the top-performing algorithm in 12

instances, followed by HLS in 7 and HCGD in 4. This notable trend prompts us to focus on the comparative analysis by assigning rankings to the algorithms for each instance.

Furthermore, Table 4.1 provides a detailed analysis of the rankings for each method across instances. Another insightful perspective is to examine the average rank for each algorithm, moving beyond the observation that SA frequently achieves the top rank.

Surprisingly, when summing the ranks, HLS edges ahead of SA by a single point, in contrast to the initial expectation. A closer look at Table 4.1 reveals that HLS consistently secures either the first or second rank, never falling to third place, while SA occasionally underperforms with a third-place ranking in certain instances. The results presented in Table 4.1 illustrate that SA performs well in the majority of instances, with a few instances presenting notable challenges. These challenging instances are grouped in A and C, where HCGD emerges as the top-performing algorithm. Additionally, HLS exhibits consistent performance across instances, shedding light on Müller strategy of harnessing both HCGD and SA sequentially to secure victory in ITC 2007 (Müller [2009]).

4.2.3 Conclusion and Motivation

In summary, these experiments have demonstrated that while Hybrid Local Search, as originally proposed by Müller [2009], is a robust algorithm, it can be outperformed by one or two of its own components executed solely. Notably, the results reveal that the Iterated Simulated Annealing, a version of Hybrid Local Search without the use of HC or GD components, outperforms the original method on more than half of the ITC 2007 instances. Surprisingly, in certain cases, instances fare even better with HCGD.

Although Hybrid Local Search is a well-performing fixed algorithm, it can still be outperformed by alternative algorithms. What is particularly intriguing is that lighter variants of Hybrid Local Search, achieved by deactivating specific local search components while keeping their parameters intact, yield superior solutions compared to the original method.

This realization has sparked the idea of creating an adaptable Hybrid Local Search framework, where variants like Iterated Simulated Annealing or iterated HCGD can be easily generated. In this endeavor, we do not limit ourselves to simply deactivating fixed algorithms without altering their parameters. Instead, we would like to provide the flexibility to manipulate all the parameters of each local search, redefine the established order of the local search sequence within HLS, and even directly customize the local search algorithms themselves. The idea would be to extract the most valuable elements from each local search, lighten the code, and enhance overall performance while creating the best new solver.

4.3 The Iterated Sequential Local Search Framework

This section outlines work toward developing an adaptable framework capable of generating a wide array of algorithms that iterate through a sequence of modular local search algorithms.

Section 4.3 starts by discussing the initial concepts that led us to transform a fixed-structure Hybrid Local Search algorithm into a modular and highly configurable framework.

Subsequently, Section 4.3 focuses on the various parameters and considerations necessary to adapt this generalization to our specific problem. Our leading goal is to demonstrate how this framework, developed on the basis of the original Hybrid Local Search, can finally be used to instantiate Hybrid Local Search (HLS).

Literature Context The development of the Iterated Sequential Local Search Framework is rooted in the context of generalizing Hybrid Local Search (HLS), a method that combines various local search techniques. This transformation aims to create adaptable software able to generate efficient iterated methods tailored to specific problem domains. Addressing the need for such adaptability and generality has been a central theme in the study of neighborhood-based algorithms, leading to the emergence of various approaches.

One notable contribution in this field is the concept of Generalized Local Search Machines (GLSM), as introduced by Hoos and Stützle [2004]. GLSM effectively manipulates Stochastic Local Search (SLS) algorithms and hybridizes them, exemplifying the power of generalization.

Marmion et al. [2013] extended the concept of generalization to Iterated Local Search (ILS) in their work on Generalized Local Search. Building upon the principles of ILS (Lourenço et al. [2019]), their approach demonstrates the adaptability of generalization techniques across different local search paradigms.

In the field of combinatorial optimization, Schaerf and Meisels [2000] also contributed to Generalized Local Search, developing an abstract code that combines the strengths of strategies proposed by Schaerf [1997] and Glover et al. [1993]. This abstract approach fosters flexibility and adaptability by integrating diverse strategies.

These works on generalization highlight the importance of creating frameworks that transcend specific algorithms. They provide a solid foundation for the development of novel and customized local search methods to tackle complex optimization problems. In the following sections, we focus on our contribution to this endeavor, where we outline how we have generalized the HLS method within the Iterated Sequential Local Search Framework.

4.3.1 Hybrid Local Search Abstraction

In this part, the idea is to present the process that led to the Iterated Sequential Local Search framework.

Thus, the first possible step towards abstraction is to consider that the three heuristics are each only variants of some Local Search algorithms with a different set of parameters θ_x. For example, θ_i corresponds to Hill Climbing from HLS, θ_j to Great Deluge, and θ_k to Simulated from HLS.

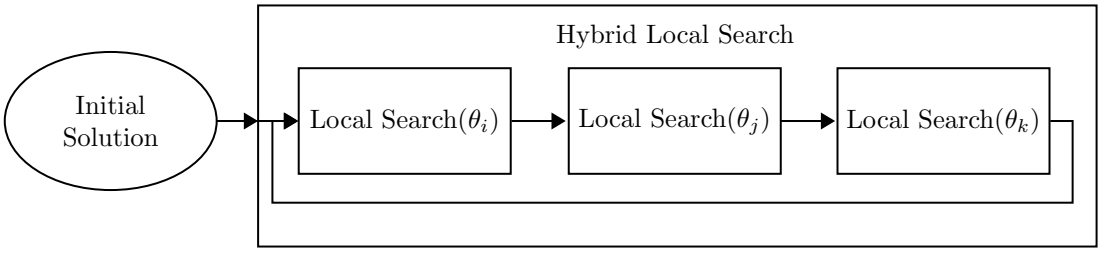


Figure 4.2: Hybrid Local Search First Abstraction: Iterated Sequence of three Local Search algorithms

Figure 4.2 shows a flowchart when we consider that each local search corresponds to a generic local search algorithm with a particular set of parameters. Figure 4.2 highlights that Hybrid Local Search is an algorithm that iterates on a sequence of local search algorithms. Thanks to this

first abstraction, the framework enables the modification to change the order of the different local search algorithms included. To change this order, the set of parameters given to each local search must be changed. The order of the sequence can change the results because the synergies between components might be not the same.

A second abstraction would concern the number of local search methods. Remember that Section 4.2 has shown that deactivating one local search in HLS can improve its performance on some instances. So, the next abstraction needs to be able to deactivate local search algorithms as required. Deactivating one or more of the three local search algorithms could be done via a θ_0 set that goes directly to the next one. However, the aim is to develop a framework that is as configurable as possible. The idea is therefore to set a number n of local search algorithms per iteration. Recall that there is one sequence of local search algorithms that repeats at each iteration. Depending on the n chosen, the framework requires n number of θ_x.

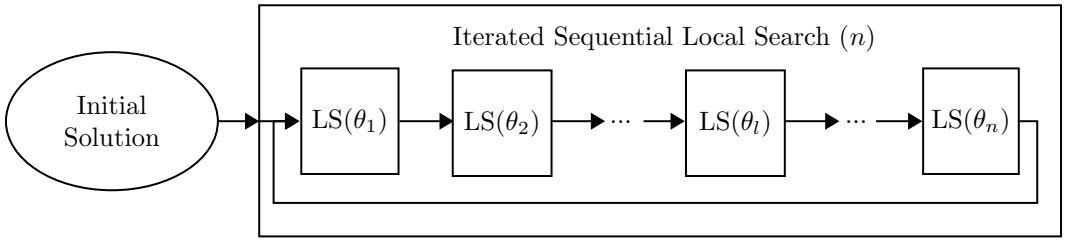


Figure 4.3: Iterated Sequential Local Search. n represents the length of the sequence.

Figure 4.3 illustrates the principle behind our framework. It showcases an algorithm that iteratively applies a sequence of local search algorithms in a loop. This sequence comprises n neighborhood-based algorithms, which may be the same algorithm repeated several times or different ones, it does not matter. Within this sequence, each local search algorithm starts from an input solution and moves to a current solution before forwarding it as input to the next local search. Our framework can emulate the behavior of Hybrid Local Search (HLS). For instance, when n equals 3, and the three local search algorithms are, in this order, Hill Climbing (HC), Great Deluge (GD), and Simulated Annealing (SA), it exactly operates as a Hybrid Local Search.

One potential concept for enhancing the modularity of the framework involves the creation of sequences that can evolve with each iteration. However, at present, we have set aside this idea due to the significant complexity it would introduce to the framework design. One drawback of creating multiple sequences that follow one another would be that it could be considered as having one single lengthy sequence. Nevertheless, when employing machine learning methods, the sequence evolution could be guided by the behavior of the solutions during the run.

At this stage, each Local Search is defined by a set of parameters that determine whether it behaves as Hill Climbing, Great Deluge, or Simulated Annealing. The details of how we achieve this versatility in the Local Search algorithm are explained below in the next section.

4.3.2 Pattern Designation of Local Search

The previous section showed how starting from Hybrid Local Search, we obtained the idea and overall structure of the Iterated Sequential Local Search framework presented in Figure 4.3. The core abstraction consists of considering the main similarity between Hill Climbing, Great Deluge,

and Simulated Annealing of the Hybrid Local Search. That is, to consider each as a local search and thus, to consider each as a variant of a generic local search algorithm.

As mentioned earlier, prior work has focused on generalizing algorithms such as Iterated Local Search, essentially an iterative process combining local search and perturbation phases. One approach might have been to create an algorithm that takes only one parameter. Then, depending on the value of this parameter, the local search would launch Hill Climbing (HC), Great Deluge (GD), or Simulated Annealing (SA) as described in Chapter 1. In this scenario, the generic local search would have a fixed number of versions, three algorithms already implemented. Thus, no new local search could be created by a new set of parameters. Only the behavior of HLS local search methods would change, such as having a higher temperature at the beginning of Simulated Annealing. However, our goal is to achieve maximum abstraction to minimize redundancy in the code and enhance modularity. As a result, we have chosen to develop an adaptable local search algorithm capable of simulating the behaviors of HC, GD, or SA. This approach not only streamlines the coding process but also improves the overall modularity of the framework.

Algorithm 8 Generic Local Search (GLS)

```
1: GLS_INIT(parameters)
2: while not GLS_TERMINATION(parameters) do
3:     Neighbor ← Neighbors(Current)
4:     if GLS_ACCEPTANCE(Neighbor, Current, parameters) then
5:         Current ← Neighbor
6:     end if
7:     GLS_UPDATE(parameters)
8: end while
9: Return Current
```

Algorithm 8 represents the Generic Local Search (GLS) that serves as an abstract representation of every Local Search component within our Hybrid Local Search (HLS) framework. Our primary objective in designing this algorithm was to create a significantly more flexible framework than the original Hybrid Local Search code. This generalization of Local Search in Generic Local Search (GLS) takes inspiration from the three approximate methods encapsulated within HLS.

GLS commences with the GLS_INIT procedure, which initializes values and computes all the parameters required for the subsequent works of the GLS. Additionally, this procedure may encompass operations that are executed exclusively during the first invocation.

Algorithm 8 follows an iterative pattern. This pattern continues until GLS_TERMINATION returns false. This procedure follows specific rules using parameter values. The core of the pattern involves the manipulation of a single solution and the generation of a neighbor using a function that employs one or more operators.

The neighbor generated in this process is then subjected to the GLS_ACCEPTANCE procedure, which returns whether the neighbor becomes the new current solution. For this, the procedure requires the values of the parameters, the current solution, and the neighbor solution. If accepted, the new solution undergoes fitness evaluation to assess if it represents a global improvement. For example, in the context of Simulated Annealing, this step allows the best solution found to be memorized throughout the execution.

The last procedure executed at each iteration in the algorithm is known as GLS_UPDATE, in charge of adjusting parameter values as each new neighbor is explored. These operations may

be at specific intervals, typically every x iteration, and GLS_UPDATE autonomously manages this process based on its internal parameters. Furthermore, GLS_UPDATE may include a set of parameter modifications exclusively applied during the final exploration iteration, triggered when the termination criterion is met. These modifications are often designed to prepare the parameters for the subsequent call of the GLS.

Additionally, the parameters controlling the acceptance and termination functions are updated as the algorithm progresses. Importantly, if the termination criterion is satisfied, the algorithm returns the current solution, not necessarily the best solution saved. In contrast, Iterated Sequential Local Search returns the best solution identified during its execution.

In the following paragraphs, we show how Algorithm 8 can instantiate any of the three local search algorithms contained in Hybrid Local Search. We are going to use the pseudo-codes presented in Chapter 1 but modified so that the similarity with Algorithm 8 is clear.

Hill Climbing As mentioned in Chapter 1, Hill Climbing of Hybrid Local Search is uncommon due to the acceptance procedure that accepts equivalent neighbors. This type of HC is known in the literature as neutral HC. This type of acceptance function is useful when facing very neutral problems, where numerous solutions share the same fitness and where looking for a strictly improving solution first means traversing some of the plateau. Indeed, equivalent neighbors form plateaus in the landscape. Therefore, this neutral acceptance implies that a bound must be set, in this case, $maxIdle$, for the algorithm to stop, otherwise, it may loop over the same plateau defined in Chapter 2.

Algorithm 9 presents the modified Hill Climbing pseudo-code of Hybrid Local Search presented in Chapter 1. This pseudo-code has been modified in structure but works in the same way. Algorithm 9 represents the pseudo-code modified to externalize the repeating operations specific to Hill Climbing in order to display the Generic Local Search presented in Algorithm 8.

Algorithm 9 clearly shows the same pattern as Generic Local Search for the main procedure. Concerning the auxiliary procedures, GLS_INIT prepares the two parameters used, i.e. the $MaxIdle$ at 50,000 and the $IterWithoutImprovment$ at 0, these two parameters are only used to determine whether the HC should end or not.

GLS_UPDATE resets the value of $IterWithoutImprovment$ to 0 if the new $Current$ is better than the best solution found overall, then increases the value by 1. The best solution fitness is saved in the memory and is always readable by all procedures.

GLS_ACCEPTANCE features the neutral Hill Climbing comparison of HLS. Algorithm 9 and Algorithm 8 do not specify this, but on each acceptance, the new $Current$ is compared with the best solution found during all sequences handled by HLS or by the ISLS framework.

Finally, GLS_TERMINATION checks whether or not the maximum number of iterations without improvement has been reached.

Algorithm 9 clearly shows that it is possible to modify the HLS Hill Climbing pseudo-code while preserving its specifics to adapt it to the formulation proposed by Algorithm 8.

Great Deluge The Great Deluge of HLS behaves like the method described by Dueck [1993]. However, it introduces a new parameter, at, which extends both the duration and the influence of the $Bound$ parameter. This extension occurs when several sequences of the global algorithm fail to discover a new global optimum. Great Deluge accepts solutions below the $Bound$ value. The threshold progressively decreases with each iteration.

Algorithm 9 Generic Local Search - Hill-Climbing

```
1: procedure GLS_INIT(parameters)
2:     MaxIdle ← 50,000
3:     IterWithoutImprovement ← 0
4: end procedure
1: procedure GLS_UPDATE(parameters)
2:     if Current < BestFitnessFound then
3:         IterWithoutImprovement ← 0
4:     end if
5:     IterWithoutImprovement + 1
6: end procedure
1: function GLS_ACCEPTANCE(Current, Neighbor, parameters)
2:     Return Neighbor ≤ Current
3: end function
1: function GLS_TERMINATION(parameters)
2:     Return MaxIdle < IterWithoutImprovement
3: end function
1: function HILL-CLIMBING(Current)
2:     GLS_INIT(parameters)
3:     while not GLS_TERMINATION(parameters) do
4:         Neighbor ← Neighbors(Current)
5:         if GLS_ACCEPTANCE(Current,Neighbor,parameters) then
6:             Current ← Neighbor
7:         end if
8:         GLS_UPDATE(parameters)
9:     end while
10:    Return Current
11: end function
```

Algorithm 10 is a modified version of Great Deluge pseudo-code. It also exhibits the same pattern as the algorithm described in Algorithm 8.

Great Deluge (GD) needs more input values, so GLS_INIT sets the values of three parameters: *UpperBoundRate*, *LowerBoundRate*, and *CoolingRate* which respectively set the value of the Bound at the beginning, the value of the Bound where GD stops, and the coefficient that decreases the Bound. The fourth parameter *at* is given by HLS or in our case by ISLS and corresponds to the number of algorithm sequences executed without finding a new best optimum. Finally, if this is the first time the GD has been executed, the *Bound* is calculated; otherwise, the value of the parameter left over from the last execution is used.

GLS_TERMINATION examines whether the *Bound* value reaches a specific threshold. The latter uses the best fitness found throughout the algorithm execution. If a better solution is discovered during the algorithm run, this threshold decreases. This approach effectively prolongs the algorithm execution when it has enhanced the overall score of ISLS. Furthermore, the threshold is adjusted based on the variable *at* and is, therefore, lower if the global algorithm, the ISLS framework, has encountered challenges in enhancing its score over a sequence. This stopping criterion takes into

Algorithm 10 Generic Local Search - Great Deluge

```
1: procedure GLS_INIT(parameters)
2:     UpperBoundRate ← 1.15
3:     LowerBoundRate ← 0.9
4:     CoolingRate ← 0.9999999999874
5:     At ← NumberSequenceWithoutGlobalImprovment
6:     if First call then
7:         Bound ← BestFitnessFound ∗ UpperBoundRate
8:     end if
9: end procedure
1: function GLS_TERMINATION(parameters)
2:     Return Bound > BestFitnessFound ∗ LowerBoundRate^at
3: end function
1: function GLS_ACCEPTANCE(Current, Neighbor, parameters)
2:     Return Neighbor ≤ Bound
3: end function
1: procedure GLS_UPDATE(parameters)
2:     Bound ∗ CoolingRate
3:     if GLS_TERMINATION(parameters) then
4:         Bound ← BestFitnessFound ∗ UpperBoundRate^at
5:     end if
6: end procedure
1: function Great Deluge(Current)
2:     GLS_INIT(parameters)
3:     while not GLS_TERMINATION(parameters) do
4:         Neighbor ← Neighbors(Current)
5:         if GLS_ACCEPTANCE(Current, Neighbor, parameters) then
6:             Current ← Neighbor
7:         end if
8:         GLS_UPDATE(parameters)
9:     end while
10:    Return Current
11: end function
```

consideration the recent overall progress of the entire framework.

At each iteration, GLS_UPDATE updates the *Bound* value and also performs a final update. In fact, GLS_UPDATE prepares *Bound* for its next call if the method is about to stop.

The GLS_ACCEPTANCE procedure simply verifies whether the neighbor falls below a specified threshold, the *Bound* value. This acceptance criterion provides a wider margin for perturbation and maintains determinism, as the *Bound* consistently decreases predictably.

Simulated Annealing The Simulated Annealing of HLS does not behave exactly like the traditional Simulated Annealing. It is a local search that accepts better solutions based on the Metropolis criterion but usually stops when the temperature gets too low. However, the HLS Simulated Anneal-

ing stops after a set number of iterations without improvement, similar to the HLS Hill Climbing. The cooling schedule and this number of iterations without improvement depend on the problem size as explained in Chapter 1.

Simulated Annealing (Algorithm 11) has been adapted from its initial version described in Müller [2009] to fit GLS (Algorithm 8). Algorithm 11 represents an algorithm that acts in the same way as the HLS SA. Algorithm 11 shows the main function is again the same as for Generic Local Search.

GLS_UPDATE configures the *CoolingRate* that is applied to decrease the temperature. Then stores the values for the number of iterations between two cooling and the maximum number of iterations without improvement. The procedure also initializes the counters for the cooling and termination processes. GLS_INIT configures the temperature value if this is the first call to SA, so it requires a final update in GLS_UPDATE.

GLS_TERMINATION checks that the number of iterations without overall improvement has not been reached. This limit varies according to the instance processed but is fixed per instance and during the run.

GLS_ACCEPTANCE tests with the metropolis metric whether the $Neighbor$ is accepted or not.

For the update function, the change is more complex, as the updates are more numerous and conditional. Thus, GLS_UPDATE incorporates the necessary modifications to the parameters of the termination criterion, here the procedure executes the count of the number of SA iterations without global improvement. In addition, GLS_UPDATE manages the counting of the number of iterations since the last cooling, and performs it if the number is reached, then resets the counter to 0. Finally, like GD, GLS_UPDATE has a final update to prepare the value of $Temperature$ for the next SA call in ISLS. To do this, the value is reheated according to a process described in Chapter 1.

4.4 Implementation of Iterated Sequential Local Search

This section presents some important points regarding the concrete implementation of the Iterated Sequential Local Search. The algorithm was developed in C++ using the MH-builder platform developed by the research team, ORKAD. The framework thus developed is the application of the idea set out in Figure 4.3 and Algorithm 8 but applied to the CB-CTT. It is designed to offer great flexibility, allowing a wide range of configurations. Configurators can customize the program and create their local search algorithms. Also, this section includes a detailed description of the parameters and how to use them as a user guide. The aim is to help the understandign of how it works.

4.4.1 Framework: MH-builder

This thesis needs a framework as a base for implementing the Iterated Sequential Local Search described above. The literature encompasses a plethora of frameworks designed to facilitate the development of approximate methods for solving combinatorial problems more efficiently (Parejo et al. [2012]).

Metaheuristic Framework Examples from Literature The EasyLocal++ framework proposed by Di Gaspero et al. [2001], developed in C++, offers a Local Search generalization similar to our Generic Local Search. It specializes in local search, providing flexibility for tailored algorithms.

Algorithm 11 Generic Local Search - Simulated Annealing

```
1: procedure GLS_INIT(parameters)
2:     CoolingRate ← 0.82
3:     MaxIterCooling ← InstanceDependentValue1
4:     MaxIterWithoutImprovment ← InstanceDependentValue2
5:     IterWithoutImprovment ← 0
6:     IterWithoutCooling ← 0
7:     if First Call then
8:         Temperature ← 2.5
9:     end if
10: end procedure
1: function GLS_TERMINATION(parameters)
2:     Return MaxIterWithoutImprovment < IterWithoutImprovment
3: end function
1: function GLS_ACCEPTANCE(Current,Neighbor,parameters)
2:     Return Uniform(0,1) < exp^(Current−Neighbor / Temperature)
3: end function
1: procedure GLS_UPDATE(parameters)
2:     IterWithoutCooling + 1
3:     if IterWithoutCooling > MaxIterCooling then
4:         IterWithoutCooling ← 0
5:         Temperature ← Temperature * CoolingRate
6:     end if
7:     if BestFitnessFound < Current then
8:         IterWithoutImprovment + 1
9:     end if
10:    if GLS_TERMINATION(parameters) then
11:        Temperature ← Reheat(Temperature)
12:    end if
13: end procedure
1: function Simulated Annealing(Current)
2:     GLS_INIT(parameters)
3:     while not GLS_TERMINATION(parameters) do
4:         Neighbor ← Neighbors(Current)
5:         if GLS_ACCEPTANCE(Current,Neighbor,parameters) then
6:             Current ← Neighbor
7:         end if
8:         GLS_UPDATE(parameters)
9:     end while
10:    Return Current
11: end function
```

Paradiseo (Cahon et al. [2004]), commonly used in C++, is now available in Python, enhancing its accessibility. This versatile framework focuses on metaheuristics and excels in multi-objective

optimization, making it suitable for complex problems. It benefits from an active community.

jMetalPy (Benítez-Hidalgo et al. [2019]), a Python framework, is designed for multi-objective optimization. Like Paradiseo, it offers diverse metaheuristics and includes benchmark problems for algorithm evaluation. It is open-source, encourages contributions, and receives active development.

MH-Builder This thesis uses the MH-builder platform developed by the ORKAD research team. MH-builder is a useful software written in C++ that facilitates the integration of various problem domains and multiple algorithms for solving them. One notable advantage of MH-builder is its ability to connect problems, solvers, and operators. In addition, parameters and algorithmic components can be changed on the fly during execution. All that is required is to implement the necessary code for loading the problem and defining the neighborhood operators. The platform tackles the complex task of linking these components together, simplifying the development process, and reducing the complexity of implementation.

A strength of the C++ language is its exceptional execution speed, particularly when the code is optimized. This attribute makes C++ a popular choice for solving combinatorial problems where performance is critical. Leveraging the efficiency of C++, MH-builder provides researchers and practitioners with a robust and efficient framework to tackle a wide range of optimization challenges.

By harnessing the power of MH-builder and the advantages offered by the C++ language, we could investigate various problem domains, experiment with different algorithms, and optimize their performance. The seamless integration and high-speed execution capabilities of MH-builder greatly enhanced the work efforts, enabling me to efficiently address complex combinatorial problems and achieve significant advancements in this thesis work.

4.4.2 Coding Specificity

Solution Representation MH-builder provides you with the flexibility to choose your preferred solution representation during implementation. You simply need to create a function that calculates the score of the solution and ensure that the operators utilize this representation optimally. The most efficient and straightforward approach discovered for representing the problem is by using a matrix. As explained in the first chapter, a matrix can efficiently represent a schedule, where each row corresponds to a room and each column corresponds to a timeslot. With this structure, it becomes easy to identify the day associated with a specific timeslot, especially when the number of timeslots per day is known. Regarding the rows, rooms are often identified by names such as A203. In the matrix, these strings are automatically converted into indices, allowing each cell to store the corresponding lecture.

One drawback of this encoding is the potential memory usage for unused cells. However, in the case of CB-CTT problems, which span over a week, the number of empty cells is not significant enough to cause memory issues. By using the matrix representation in MH-builder, you can effectively store and manipulate the necessary data for solving CB-CTT problems, achieving efficient memory utilization, and facilitating the optimization process.

Neighborhood Operators And Evaluation For the neighborhood operators, we have implemented them based on the descriptions provided in the first chapter of this thesis. These operators perform the specific tasks outlined by Müller for the problem domain. To optimize the solver execution, we have directly incorporated the evaluation of a solution within the neighborhood operators.

In essence, we have ensured that each new neighbor receives a score immediately without any additional computational overhead. To achieve this, we examined the soft constraints affected by each operator and developed data structures that track the real-time number of violations without the need to analyze the entire schedule.

For instance, when a lecture, such as lecture 1 of course 1 belonging to curriculum 1, is moved to a different room, we only need to update the violations related to Room Capacity and Room Stability. Moreover, we update only the structure that stores Room Stability violations for course 1. By deducting the previous value and adding the new value, we can obtain the final score.

To optimize this process further, we have implemented a local count of violations for each structure, such as curricula and courses. This approach eliminates the need for a costly calculation of the objective function, resulting in improved performance and efficiency. For instance, concerning RoomCapacity, each lecture scheduled has a local RoomStability violation counter.

By incorporating these optimizations, we have significantly reduced the computational time cost associated with evaluating the solution fitness, enabling the solver to operate more efficiently.

In this section, not every mechanism is detailed and explained, as the emphasis is on coding optimizations that help the framework achieve better performance. It is important to emphasize that significant thought and consideration were given to the coding process to prioritize efficiency. By adopting this approach, we aimed to maximize the number of explorations conducted per minute, ensuring that our solver operates at its highest computational efficiency.

Heuristic In terms of coding heuristics, we employ a versatile modular heuristic approach. This heuristic, depending on the provided parameters, can activate different functions to implement acceptance and ending criteria, such as Hill Climbing, Great Deluge, or Simulated Annealing.

As highlighted earlier, the modular heuristic allows for the hybridization of methods. It goes beyond being only Simulated Annealing or Hill Climbing. It can incorporate probabilistic acceptance using temperature while utilizing the bound from the Great Deluge as its end criterion, which dynamically evolves based on the best fitness value. This flexibility allows us to leverage the strengths of various techniques and adapt the heuristic to different problem scenarios.

In our work, we have ensured that our approach can hybridize strategies from HLS (Hill Climbing, Great Deluge, and Simulated Annealing). As part of this effort, we introduced the option of strict acceptance for HLS, which has been observed to enhance performance in certain cases.

Additionally, we have incorporated the capability to modify the stopping criterion specifically for Hill Climbing. In this context, HLS considers that if no local improvements are achieved within a specified number of explorations, such as 50,000, the search process should end. This adjustment ensures that the algorithm prioritizes finding significant improvements in the global context.

4.4.3 Framework Execution Parameters Overview

The following section focuses on the more practical part of ISLS framework. That means explaining how to manipulate the input parameters to use this program efficiently. The aim is to produce a concise manual. Admittedly, we have not used a parser that assigns automatically, consequently the order is very important. Moreover it can be a bit daunting to understand a command line without this part. An explanation of parameters may offer an overview of all current possibilities and even the future, which has the framework developed using MH-builder, another framework developed by the research team ORKAD. The parameters are given in order to make the explanations as clear as possible.

```
./exe ins01.txt 300 1 inst01_init_sol1.txt output1.txt
1 1 1 0 0 2 _ 1 1 50,000 _  3 2 5 0.99 0.9 20 0.8 13 19 _
```

Figure 4.4: Example of a real call of Iterated Sequential Local Search

Figure 4.4 shows an example of a command line used in this thesis to run the Iterated Sequential Local Search. Figure 4.5 shows a theoretical command line.

```
[EXE] [Instance] [Time Budget] [Seed] [Init_Sol] [Output]
[Neighborhood Operators] [NB_Heuristics] _ [PARAMETERS SETS] * [NB_Heuristics] _
```

Figure 4.5: Command-line arguments of Iterated Sequential Local Search

Instance The two figures show that after calling the compiled file, the first input is the access path to the file of a Curriculum Based-Course Timetabling instance.

Time Budget The second parameter is the allocated time in seconds. As seen in the previous sections, the program checks after each evaluation of a neighbor that it has not exceeded the time limit, which is in milliseconds. If it does, the program stops the exploration and only displays the value in the terminal in a fitness manner. If the value is zero, the algorithm stops immediately.

Seed To ensure the replicability of the results of the Iterated Sequential Local Search, we have implemented the option of setting the seed. That also makes it possible to test average performance on the same problem. The random mechanism uses every time the same object for each random call, which validates the randomness control of our heuristic.

Initial Solution The fourth input parameter is the file path of the initial solution loaded. As a reminder, we have not incorporated Muller constructor directly into the code. Figure 4.4 shows the case where we generated solutions in a folder for each instance with different seeds.

We do not use more than 30 different seeds for each instance. So we made sure we used the same seed number for the initial solution and for the Iterated Sequential Local Search. Thus, the diversity was the greatest, because each run used a different initial solution and a different exploration seed.

Output Solution As before, this parameter is a path, ideally absolute. However, this parameter corresponds to the output file created by the program to store the best solution found during exploration. As a reminder, if our solver finds several optimal solutions, the first one found is returned. The coding way accepts a new best solution only if the latter outperforms the current best solution, found so far during the current run.

Neighborhood operator The next five parameters are boolean, accepting only 1 or 0. These activation parameters are used to control the neighborhood operators. Their order is as follows: Time Move, Room Move, RoomStability Move, MinWorkingDays Move, and Lecture Move. The

first chapter of this thesis details how each works. Figure 4.4 shows an example where we have disabled operators 4 and 5, i.e. MinWorkingDays Move and Lecture Move. Currently, the Iterated Sequential Local Search can only deactivate or activate operators, which offers the opportunity to select those that are most effective, for example. An improvement objective would be to offer more neighborhood operators and to replace the boolean parameters with float parameters between 0 and 1. In this configuration, the values would correspond to the selection probability of each operator. As a reminder, it is currently a uniform distribution that handles selection.

Number of Heuristics This parameter corresponds to the number of heuristics per loop in our method. So recall that our solver creates an iterative solution optimization method. It is based on the Hybrid Local Search algorithm, which executes three heuristics in succession at each iteration of a loop: Hill Climbing, Great Deluge, and Simulated Annealing. Our algorithm accepts an integer input value strictly greater than 1. This value informs the program of the number of parameters it needs to read next. In this work, we have limited the number of algorithms per loop to three, as we consider that a high value would prevent the repetition effect. Figure 4.4 shows an example in which the created algorithm has two heuristics per sequence.

Depending on the number given to the previous parameter, the algorithm requests a group of parameters delimited by an underscore each time.

Heuristics strategies The first two parameters in each of these groups are, in order: the code for the acceptance criterion, then the code for the ending criterion.

Acceptance Condition The possible values of the Acceptance conditions are 0, 1, 2, and 3, respectively for Strict acceptance, Neutral Acceptance, acceptance with the Bound, and probabilistic acceptance with a temperature.

Ending Condition The possible values of ending conditions are also 0, 1, 2, and 3, respectively for the Maximum Number of evaluations without local improvement, the Maximum Number of evaluations without global improvement, the Bound reaches the lower bound, and the maximum number of evaluation without global improvement using an instance depending coefficient. Please note that to easily find the starting heuristics, the combination 1 1, 2 2, and 3 3 correspond to Hill Climbing, Great Deluge, and Simulated annealing from HLS.

Parameter Heuristic The other parameters are conditional. In other words, write only those that are necessary. We made this choice to avoid having a succession of zeros for each set.

The order in which to enter the parameters is given by Figure 4.6.

```
[MaxIdle] [UpperBoundRate] [CoolingRateBound] [LowerBoundRate]
[Temperature] [CoolingRateTemp] [CoolingScheduleTemp] [ReheatRate]
```

Figure 4.6: Order of parameters per heuristic for the Iterated Sequential Local Search

Table 4.2 summarizes which parameters are to be given according to each choice of heuristics strategies. As a reminder, each choice of acceptance or ending strategy requires certain parameters, as described in Section 4.3.2.

Table 4.2: Parameters to be set according to the heuristics required

Criterion	Value	Parameter
Acceptation	0	No parameter
	1	No parameter
	2	UpperBoundRate, CoolingRateBound, LowerBoundRate
	3	Temperature, CoolingRateTemp, CoolingScheduleTemp, ReheatRate
Ending	0	MaxIdle
	1	MaxIdle
	2	UpperBoundRate, CoolingRateBound, LowerBoundRate
	3	InstanceRate

The different names of the parameters are quite transparent, but for those of the probabilistic acceptance using the temperature inherited from Simulated Annealing, certain precisions seem to be necessary. CoolingScheduleTemp corresponds to a coefficient that sets the number of evaluations between two cooling phases. The ReheatRate is filled in at the end of the heuristic to raise the temperature. Both are instance dependent coefficients.

Figure 4.4 shows that the first heuristic only requires the MaxIdle parameter as the ending criterion was 1. The second heuristic required many more input parameters as it had the 3 2 combination which needs 7 parameters to work. As a reminder, these are always given in the order provided by Figure 4.6.

4.5 Conclusion

This chapter focuses on an efficient method for tackling the Curriculum-Based Course Timetabling problem. At its core lies the Hybrid Local Search algorithm; a metaheuristic approach that employs a sequential loop of three distinct local search algorithms.

This chapter focuses on the performance analysis of Hybrid Local Search and its structure. More specifically, the preliminary results questioned the use of three methods in Hybrid Local Search. They showed that for the ITC benchmark, for which Hybrid Local Search was tuned, Hybrid Local Search could be beaten by a simpler version of itself, i.e. iterated Simulated Annealing, on a majority of instances. The results show that it is interesting to question the rigidity of the Hybrid Local Search structure and to study variants using the components and strategies contained in Hybrid Local Search but changing the structure, the number of algorithms per sequence, the parameters, and the local search itself.

This section explores the generalization of the Hybrid Local Search (HLS) algorithm, giving rise to the Iterated Sequential Local Search (ISLS) framework. ISLS offers a high degree of flexibility, allowing the creation of new resolution algorithms while retaining the ability to instantiate the original HLS. Additionally, we focus on the concept of generalizing local search algorithms into a unified, adaptable algorithm, highlighting the possibility of employing a generic local search to instantiate the three algorithms present in HLS.

In this chapter, we detail the implementation of the Iterated Sequential Local Search (ISLS) framework on the MH-builder platform, developed by the ORKAD research team. The chapter provides insights into coding considerations for optimizing performance and offers instructions for running the framework from the command line. Additionally, parameter naming conventions are

explained.

With the introduction of the Iterated Sequential Local Search (ISLS) framework, new possibilities emerge. ISLS allows for the creation of new algorithms, featuring novel sequences of local search methods, themselves incorporating acceptance and termination strategies from diverse Hybrid Local Search algorithms. This framework paves the way for automating the configuration of solution algorithms for addressing the Curriculum-Based Course Timetabling problem.

Chapter 5

Automatic Configuration of ISLS

Publication:

- Thomas Feutrier, Marie-Éléonore Kessaci, and Nadarajen Veerapen. "When Simpler is Better: Automated Configuration of a University Timetabling Solver." CEC 2023: IEEE 2023 Congress on Evolutionary Computation. IEEE 2023.

Contents

5.1 Introduction

Over the years, the field of automatic algorithm configuration has seen notable development, giving rise to numerous methods with diverse strategies. These algorithms automatically design configurations to improve the performance of the solving algorithms using these parameters. This thesis focuses on the application of metaheuristics for solving combinatorial optimization problems. Such methods involve a multitude of parameters that govern the behavior of the solving algorithm during execution.

A configuration represents a set of parameters with fixed values, and determining the ideal configuration for optimizing the performance of the resolution method is a task that can be tackled through parameter tuning and parameter control (Eiben et al. [1999]). Parameter control involves an online process where parameters are adjusted during runtime (Karafotias et al. [2014]). Conversely, parameter tuning, also known as algorithm configuration, is an offline process that seeks to optimize and explore different configurations before concrete execution, ultimately selecting one or more configurations (López-Ibáñez et al. [2016]).

Chapter 4 detailed our thesis work on metaheuristic abstraction, and more specifically the abstraction of the Hybrid Local Search. It presented the Iterated Sequential Local Search (ISLS), an algorithm that executes in a loop for a given time a sequence of local search algorithms where their number and nature have to be specified. These local search algorithms follow the scheme defined in the thesis: Generic Local Search. Depending on the given parameters, this algorithm takes on the desired nature by hybridizing the acceptance and termination criteria of two different local search algorithms. For example, the local search can implement the simple acceptance criterion of Hill Climbing and the termination criterion of Simulated Annealing, while these two methods are generally each on their own.

Hence, Chapter 4 illustrates ISLS as a framework capable of instantiating the state-of-the-art method for addressing the Combinatorial Benchmark Curriculum-Based Timetabling (CB-CTT) problem. ISLS is a highly parameterized framework that allows for the customization of each local search behavior based on the specified values. Leveraging this flexibility, an automatic algorithm configurator can harness the full potential of a framework like ISLS, thereby offering a high-performance algorithm for CB-CTT. It is important to note that ISLS incorporates all components of HLS, which represents the current state-of-the-art local search method. This chapter employs an automatic algorithm configuration algorithm, irace, within the ISLS framework, using a selected group of instances from Chapter 3 as training data. The goal is to craft a tailored resolution algorithm optimized for CB-CTT, demonstrating superior performance compared to the state-of-the-art.

This section begins by providing an overview of the background of our work in automatic algorithm configuration, with a specific focus on configurators employing diverse methodologies to identify optimal configurations. The subsequent portion of the chapter showcases the practical application of ISLS and the irace configurator. This marks the first instance of creating an algorithm based on ISLS, retaining the original sequence while exploring a subset of its parameters, and achieving superior performance compared to the original HLS approach. To conclude, the chapter ends with an ablation analysis, guided by literature (Fawcett and Hoos [2016]), aimed at pointing out the parameters that bear the most significant impact in this specific context.

5.2 Automatic Algorithm Configuration

This section focuses on the complexities of automatic algorithm configuration, beginning with a precise formal definition. Subsequently, we explore the specific variant of Algorithm Configuration employed in our work and dissect the various constituent sub-problems. To round off the discussion, we explore existing algorithmic solutions proposed in the literature, offering insights into a select few of these notable approaches.

5.2.1 Definition

In the field of operational research and discrete problem solving, the task of tuning algorithms is a common challenge. Metaheuristics, in particular, offer a high degree of adaptability, enabling algorithms to utilize diverse neighborhood operators, various stopping criteria, and adjustable execution lengths. Moreover, heuristics, in their abstract nature, can modify entire operational components when applied to specific real-world problems, with these modifications also regarded as parameters to consider. This flexibility has paved the way for the fine-tuning of algorithm parameters, striving to achieve optimal performance within constrained time limits for each distinct problem.

Combinatorial optimization is a domain where the use of metaheuristics introduces unique challenges to be addressed. As a reminder, metaheuristics are generic algorithms applied to solutions to systematically explore the search space and try to find the best solution within a limited time budget. However, there is a plethora of diverse solution methods, each behaving differently based on the values of specific parameters. Automatic Algorithm Configuration (AAC) is an approach to determine the most suitable version of a heuristic for a given problem. An algorithm configuration problem can be formally stated as follows (Hoos [2012]):

1. An algorithm A with a set of parameters that significantly affect its behavior.
2. A configuration space Θ consisting of different configurations. Each configuration θ in Θ represents a unique set of values, with one value for each parameter p_i.
3. A fitness function or objective function f is provided by the problem to evaluate the final solution returned by a configuration.
4. A set of problem instances I_{train}.
5. $\arg\min_{\theta \in \Theta}$(Cost Function) that represents the objective function of the AAC algorithm.

The parallel pursuit of cost minimization by Automatic Algorithm Configuration (AAC) and the Combinatorial Curriculum Timetabling Problem (CB-CTT) aligns the algorithm configuration problem with optimization objectives. In this context, AAC navigates through various configurations, much like a local search exploring different timetables in the case of CB-CTT, both aiming to optimize their respective objectives.

In automatic configuration, a configuration θ is defined as a list of values for each parameter. The parameters can take different types, including cardinal, boolean, and numeric (integer or real) with predefined ranges. Furthermore, it is common to specify forbidden configurations. For example, when a local search is tuned, it must be forbidden to deactivate all neighborhood operators, otherwise the algorithm does not work. In general, the main objective of automatic configuration is to discover configurations that yield better performance compared to the initial configurations provided for a given set of problem instances. In the context of this thesis, the primary goal

of Automatic Algorithm Configuration is to obtain configurations that outperform Hybrid Local Search (HLS), serving as the performance baseline. Consequently, the θ configuration of ISLS that instantiates HLS, is consistently included among the initial configurations provided to the configurator.

Automatic algorithm configuration operates on the basis of providing a configurator with a predefined set of configurations for evaluation. To effectively explore the configuration space, the configurator necessitates knowledge of the parameters, their allowable ranges, and the available options. Additionally, an evaluation function is required to assess the performance of the solutions produced by these configurations. Typically, the optimization focus lies on enhancing performance, although in certain scenarios, such as with exact methods, the primary objective may be to minimize execution time when performance remains consistent.

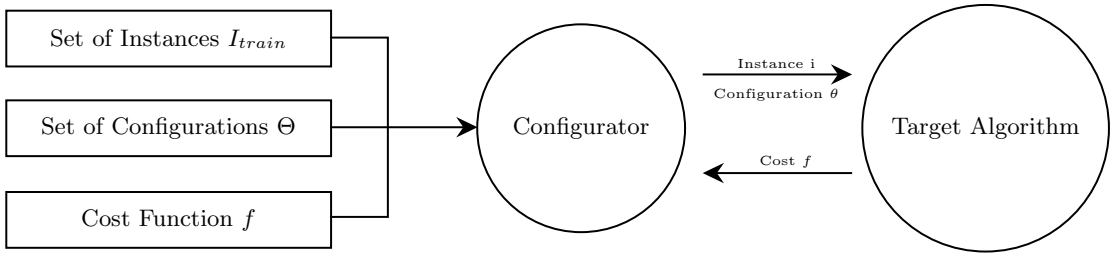


Figure 5.1: Generic Process of Automatic Algorithm Configuration

The principle of automatic algorithm configuration can be summarized by Figure 5.1, which clearly shows that parameters, a target function, and instances are given to the chosen configurator, which tests different configurations with the solver, often referred to as the target algorithm in automatic configuration. There are many sub-problems of AAC whose objectives are to return the best algorithm to use, and they generally follow the same schema detailed previously.

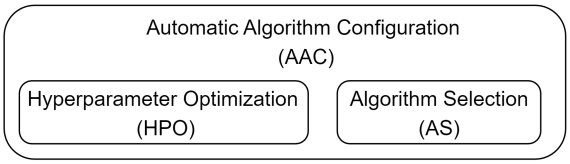


Figure 5.2: Automatic Algorithm Configuration sub-problems

Figure 5.2 illustrates the classification of major AAC types. AAC encompasses Algorithm Configuration (AC), as well as Hyperparameter Optimization (HPO) and Algorithm Selection (AS).

Hyperparameter Optimization (HPO) HPO is also a very widely studied area of research, with several reviews (Luo [2016], Yang and Shami [2020], Bischl et al. [2023]). Hyperparameter Optimization (HPO) is a machine learning technique aimed at finding the best hyperparameter settings for a model. Hyperparameters are pre-defined configuration settings, such as learning rates and layer sizes, crucial for model performance. HPO methods systematically search for optimal hyperparameters, enhancing model convergence and generalization. Algorithm Configuration focuses

on optimizing algorithm parameters for a specific problem, while Hyperparameter Optimization fine-tunes hyperparameters of machine learning models for improved performance.

Algorithm Selection (AS) is indeed a subproblem of AC (Algorithm Configuration). However, it is important to note that in our subsequent work, we do not focus on this specific type of AAC. Algorithm selection has already received extensive attention and examination in prior studies (Kerschke et al. [2019], Kotthoff [2016]). Algorithm Selection can be considered as a subset of Algorithm Configuration with a notably compact search space. That is because it primarily involves the selection of a single categorical parameter, which determines the choice of algorithm for a given instance. Algorithm Selection can also be categorized as a form of instance-specific configuration. In essence, it revolves around the task of configuring this single categorical parameter, the algorithm choice, based on the unique characteristics of each input instance.

Online and offline Offline AAC involves optimizing algorithm configurations before the execution of the target algorithm. It explores a predefined set of configurations and returns the best one. In contrast, Online AAC optimizes configurations during the runtime of the algorithm, adapting to changing conditions or problem instances. It dynamically selects and adjusts configurations, offering real-time adaptability and potential for improved performance. In this thesis, offline Algorithm Configuration (AC) techniques are predominantly employed for optimizing algorithm configurations.

5.2.2 Methods

Automatic configuration is an extensive research field that focuses on finding the most efficient methods for problem-solving. Numerous algorithms and tools have been developed and proposed to tackle this challenge. This section provides a concise overview of selected methods in this domain, as well as discusses the configurator we have chosen for our work and the reasons behind our selection.

In this research field, in addition to the previously mentioned types of automatic algorithm configuration methods, we can introduce the concept of model-based approaches. In our study, we exclusively employ a model-based offline automatic algorithm configuration technique. The key distinction between these two approaches lies in how the configurator operates. Model-based configurators utilize statistical or machine models to assist in identifying promising configurations, while model-free configurators do not rely on any models and instead employ alternative techniques.

In the literature, numerous configurators and algorithms have been developed to facilitate automatic configuration.

One of the earliest approaches to tune algorithms with a limited number of parameters was Calibra (Adenso-Diaz and Laguna [2006]). The idea involved combining Taguchi's fractional factorial experimental designs with a local search procedure. However, the main drawback of this method is that it accepts too few parameters, which makes it unsuitable for our work. Nowadays, Automatic Algorithm Configurations typically involve a large configuration space. Nevertheless, this method highlights the strategy of using local search to guide towards promising solutions while employing a design of experiments methodology to compare configurations.

The majority of the initial approaches developed for tuning in the literature adopted a similar idea to Calibra and utilized the design of experiments methodology (DOE) (Coy et al. [2001], Ruiz and Maroto [2005], Bartz-Beielstein [2003]). DOE is a statistical technique used to systematically

plan and conduct experiments to gather data and gain insights into the behavior of a system or process. It is commonly applied in algorithm tuning to efficiently explore the parameter space and optimize algorithm performance.

Following the methods developed using local search, the literature has also explored the use of evolutionary algorithms, particularly genetic algorithms. In the context of genetic algorithms, one tuning method is the Gender-Based Genetic Algorithm (GGA) (Ansótegui et al. [2009]). To provide a simple overview, GGA is a population-based approach where configurations are represented as individuals in two populations: competitive and non-competitive. GGA combines a strong intensification procedure, which involves racing individuals from the competitive population, with diversification from the non-competitive population. The competitive population undergoes a racing process on a random subset of instances, and this subset increases linearly with each generation in GGA. This approach is designed to achieve a specific goal or outcome within the algorithm tuning process.

In summary, a multitude of tuning methods exists within this field. Several comprehensive reviews, including one by Schede et al. [2022], have cataloged and presented a summary table of these tuning approaches by class. Some well-known methods, such as ROAR (Hutter et al. [2011]), GPS (Pushak and Hoos [2020]), and BNT (do Nascimento and Chaves [2020]), have demonstrated excellent results by employing diverse strategies to fine-tune algorithms. They use Racing and ILS, exploit the configuration landscape structure, and leverage Bayesian Networks, respectively.

The subsequent section focuses on the presentation of three methods: SMAC, ParamILS, and irace. These methods hold significance as they employ diverse approaches to explore configuration space. In this thesis, irace is the chosen approach due to its distinct advantages.

SMAC The Sequential Model-based Algorithm Configuration has been proposed by Hutter et al. [2011] and aims to predict the best configuration. SMAC uses a machine learning model that predicts the performance of configurations. In most cases, it utilizes the random forest model.

The algorithm begins by generating random configurations, which are then evaluated on a randomly selected set of instances. The results help to retrain the prediction model at each step. In each iteration, the prediction model generates new promising configurations based on the previous scores. Subsequently, these new configurations are tested, and statistical tests are applied to identify the most efficient configurations for random instances. At the end of each iteration, the model is retrained to enhance and update the knowledge about the configuration space and its relationship with performance.

This tuning algorithm is known to be very effective for tuning hyperparameters in machine learning methods. Auto-WEKA (Thornton et al. [2013]) and AutoSklearn (Feurer et al. [2015]), two tools that have been developed using SMAC for tuning machine learning methods.

ParamILS ParamILS, a highly efficient algorithm proposed by Hutter et al. [2009], stands out for its heuristic nature, employing neighbor-to-neighbor moves.

Understanding the operation of the algorithm is relatively straightforward. The process begins by generating random configurations and evaluating them to identify the most promising ones. Subsequently, new configurations are generated in close proximity to the previously selected ones, with each new configuration serving as a neighbor. In ParamILS, a neighbor of a configuration differs from it in the value of only one parameter.

These new configurations are generated through a random perturbation process, involving small changes from one configuration to another. This stage is referred to as the perturbation phase in

iterated local search (Lourenço et al. [2019]). Following the perturbation phase, a local search is performed. This iterative process aims to improve the current configurations by examining their neighboring configurations and replacing them with more efficient alternatives if found. The local search continues until no further improvements can be made.

At the end of each iteration, a set of new configurations is obtained, which are neighbors to the configurations selected during the perturbation phase. Once the budget is consumed, the algorithm returns the best configuration discovered throughout the process.

irace irace, Iterated Racing for Automatic Configuration, a dedicated automatic configurator for combinatorial optimization, uses racing competitions to efficiently discover well-tuned configurations. Employing a racing competition-based strategy, irace evaluates configurations and optimizes them based on statistical insights. Additionally, irace benefits from continuous development and regular updates (López-Ibáñez et al. [2016]).

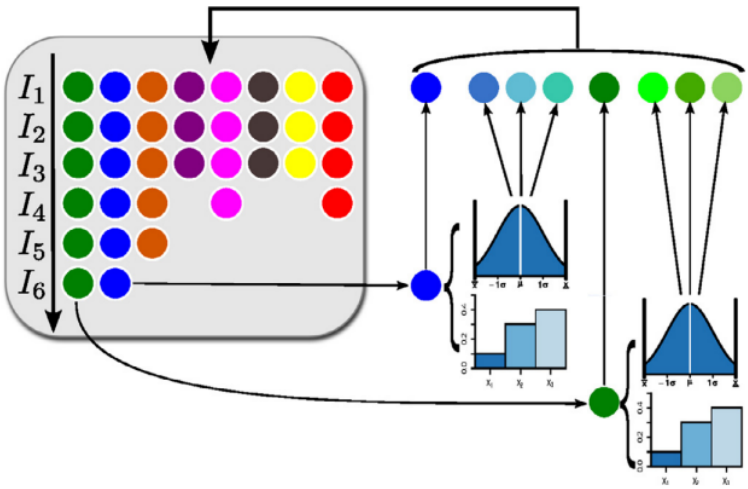


Figure 5.3: Scheme of the iterated racing algorithm (López-Ibáñez et al. [2016])

Firstly, we selected irace as our optimization tool due to its extensive documentation and user-friendly interface. irace provides a well-established framework for automated algorithm configuration. Some description of irace works can be found in López-Ibáñez et al. [2016].

The schematic process of irace is depicted in Figure 5.3. In the initial iteration, irace generates a set of random configurations or utilizes pre-defined configurations, ensuring a sufficiently large number of options. These configurations are then evaluated on a set of randomly chosen instances, serving as training instances. As the evaluation progresses, irace employs statistical tests, such as the Friedman test in most cases, to eliminate the worst-performing configurations.

With each subsequent iteration, irace consults the generated models to obtain new configurations, subjecting them to evaluation and subsequently eliminating configurations through statistical tests. The models assume that each parameter follows a normal distribution, with the aim of approximating the value of the best configuration. This iterative process allows the models to continuously refine their learning, incorporating the knowledge gained from the new best configuration

while removing less promising alternatives.

However, it is important to note that the figure does not explicitly illustrate one good feature of irace: its soft restart process when new configurations become too similar. This functionality is integral to irace objective of achieving convergence without prematurely converging to local optimum configurations. This capability serves as an additional advantage of using irace as an algorithm configurator.

5.3 AAC of ISLS for the CB-CTT

This section builds upon the previous context-setting section, demonstrating the application of ISLS AAC to the Curriculum-Based Course Timetabling (CB-CTT) problem.

We present the ISLS configuration space provided to irace, the AAC tool. This configuration differs from the global configuration space of Iterated Sequential Local Search in Chapter 4. The focus is on improving the Hybrid Local Search (HLS) structure rather than developing a new algorithm using HLS components.

Following the configuration space presentation, we explain the experimental protocol for all irace executions, detailing the instances used and their corresponding time constraints.

This section outlines the setup used to conduct AAC for ISLS in the context of CB-CTT.

5.3.1 Configuration Space

This section presents the selection of parameters used in this study.

Group	Parameter	Value
Neighborhood Operators	Op1 (`TimeMove`)	Boolean
	Op2 (`RoomMove`)	Boolean
	Op3 (`RsMove`)	Boolean
	Op4 (`MwMove`)	Boolean
	Op5 (`LMove`)	Boolean
Hill Climbing	`HC`	Boolean
	`NeutralStrict`	Boolean
	`IterHC`	$(10^4; 5 \cdot 10^4; 10^5)$
Great Deluge	`GD`	Boolean
	`UpperBoundRate`	(1.10; 1.15; 1.50; 2)
Simulated Annealing	`SA`	Boolean
	`Reheat Coeff`	(1; 7; 14)

Table 5.1: Parameters set

Table 5.1 lists the twelve parameters used in this study and their ranges. Notably, the full parameterization capability of the Iterated Sequential Local Search is not used here. The objective is to question the structural complexity of the strategies implemented in the Hybrid Local Search and not to create an overall best solver. It is also important to note that all the other parameters

are set to the default HLS values. For example, the initial temperature of the SA, if it is activated, is always 2.5.

Table 5.1 classifies them into 4 groups: the neighborhood operators and one group per heuristic. Each neighborhood operator and each heuristic can be activated (1) or not (0), so it is managed with a boolean value. If a heuristic is deactivated, HLS then moves to the next heuristic activated heuristic. Note that a security function prevents from deactivating all heuristics together. In addition, the parameter `NeutralStrict` that controlled the criterion acceptance during the neighborhood exploration is a Boolean value where 0 means that equivalent neighbors are accepted while 1 means that only strictly better neighbors are accepted. The other parameters are numerical. In order to control the size of the configuration space, we decided for this first study to limit the possible values. For each parameter, we allow at least one smaller and one larger value than the value set in the original HLS.

These parameters can be categorized into two types: boolean activation parameters, which determine the profitability of a particular tool in the Hybrid Local Search, and numerical parameters that provide flexibility to the configurator. Even with limited ranges, increasing the length of a heuristic can lead to the creation of more powerful configurations. This idea is inspired by the findings from the previous section, indicating that a longer exploration length for SA, for instance, can potentially outperform the original Hybrid Local Search.

Finally, the configuration space contains 2 449 configurations.

5.3.2 Experimental Protocol

Our goal is to investigate if specific configurations of HLS can improve the performance of the original HLS on the 77 real-world instances used in the literature (Chapter 1). In order to find these specific configurations, a configurator is run over the configuration space presented in Table 5.1. The process of automatic configuration needs training instances independent of the ones used to evaluate the final performance of the configurations. The framework used is the Iterated Sequential Local Search (Chapter 4) tuned to simulate a basic HLS and only parameters in Table 5.1 are tunable by the configurator.

Unlike the preliminary study presented in Chapter 4, here we use a part of all available instances (Chapter 1). Firstly, this study uses about 3 000 artificial instances artificially generated from real-world instances. The process of generation and how we selected them has been detailed in Chapter 3. The artificial instances are used as the training set. That means irace tests new configurations on some of these instances. Concerning the real performance, we use two sets: ITC and New real-world instances.

For both real-world and artificial instances, we applied the construction heuristic of the original HLS (Müller [2009]) to build and obtain solutions that are used as initial solutions of the optimization process.

We utilize irace (López-Ibáñez et al. [2016]) as the automatic algorithm configurator in this study. irace employs an iterated racing procedure and statistical tests to identify the best configuration from the configuration space. In the initial iteration, irace runs sampled configurations on a set of instances, typically around five instances. After this iteration, configurations that are statistically dominated are removed from consideration. For the subsequent iterations, irace generates new configurations based on the previously selected configurations from the last iteration. This iterative process continues until the allocated budget for the configurator is used up.

For our experiments, we allocated a budget of 2 000 runs to irace, using selected artificial in-

stances. Each run corresponds to the execution of one configuration on a single instance. Considering the conditional values between parameters, there are a total of 2 449 possible configurations. In the first iteration, we provided irace with the configuration corresponding to the original HLS (as shown in Table 5.2). When the budget is consumed, irace generates elite configurations that are statistically equivalent to the training instances used. We select five of these elite configurations for performance validation on the test instances. To assess the performance, both the original configuration and the five elite configurations are executed 30 times, each with different seeds, on each of the 77 real-world instances. The best fitness value reached for each run after 5 minutes is recorded as the final result.

5.4 Experiments

This section focuses on the practical application of irace, the automatic algorithm configurator introduced earlier in this thesis.

Specifically, we explore its application to selected artificial instances, as detailed in Chapter 3, to improve the configurations of the Iterated Sequential Local Search (ISLS) framework.The work presented here involves reducing the configuration space of ISLS to a structure based on HLS: a sequence of Hill Climbing, Great Deluge, and Simulated Annealing, while allowing the configurator to deactivate local search and neighborhood operators and manipulate certain parameters.

The primary goal is to expand upon the work presented in Chapter 4 by incorporating additional parameters and evaluating their performance across various real-world instances. This study aims to provide a detailed analysis of the obtained configurations and then to determine the best-simplified version of HLS for real-world scenarios.

An additional objective is to validate the selected artificial instances, assessing if configurations trained on this set consistently outperform HLS on real-world scenarios. Such validation would enhance our confidence in the representativeness of the artificial instances.

5.4.1 Analysis of Configurations

Parameter	$\text{HLS}_{\text{original}}$	θ_1	θ_2	θ_3	θ_4	θ_5
Op1	1	1	1	1	1	1
Op2	1	1	1	1	1	1
Op3	1	1	1	1	1	0
Op4	1	0	0	0	0	0
Op5	1	1	1	1	1	1
`HC`	1	0	1	1	0	1
`Neutral0Strict1`	0	NA	0	0	NA	0
`IterHC`	5	NA	5	1	NA	1
`GD`	1	1	1	1	1	1
`UpperBoundRate`	1.15	2	1.50	1.50	1.50	2
`SA`	1	0	0	0	0	0
`Reheat Coeff`	7	NA	NA	NA	NA	NA

Table 5.2: Parameter values of the original HLS and the five elites.

Table 5.2 presents the configurations obtained from the irace executions, along with the reference starting point. Each column represents a configuration, and irace returns several elite configurations ranked from θ_1 to θ_5, considering θ_1 as the best. Table 5.2 offers an overview of the converged configurations.

Recall that an activation parameter set to 1 means the related mechanism is activated. The first 5 rows correspond to each operator: `TimeMove`, `RoomMove`, `RsMove`, `MwMove`, and `LMove`. `RsMove`, `MwMove`, and `LMove` correspond respectively to Room Stability Move, MinWorkingDays Move, and Lecture Move presented by Chapter 1. The subsequent groups of lines correspond to different mechanism groups, with the next three linked to Hill Climbing, followed by two related to Great Deluge, and the last two related to Simulated Annealing. As mentioned earlier, Table 5.2 displays the elite configurations obtained from multiple irace runs with different seeds to ensure convergence. To simplify the presentation, we have selected a run that best represents the results, although slight variations in the numerical values may occur between runs.

One evident observation regarding these elite configurations is that irace appears to deem the use of SA and Op4 as unworthy. Surprisingly, none of the five elite configurations includes Simulated Annealing. That is unexpected, as our previous manual evaluation of SA alone revealed its competitive potential, nearly rivaling HLS on the ITC instances, evaluation done in Chapter 4. However, Table 5.2 indicates that irace does not consider SA alone to be a winning strategy. This difference in SA performance may be due to the instances, in fact, the addition of instances provided by De Coster et al. [2022] adds instances where SA may be less efficient. To test this hypothesis, SA is a baseline in the configuration tests.

Concerning Op4, i.e. MinWorkingDays Move, its systematic deactivation could mean that this operator has no impact and is even too slow to be worthwhile. Op4 focuses on a subpart of a timetable that violates one constraint and generates only improving solutions regarding this constraint. It is thus more complex than others. Operators Op1, 2, and 3 are much simpler. We hypothesize thus it is slower because of the complexity of its task. Indeed, it must find the lectures of a course scheduled on the same day and place them on different days. It may seem more efficient to perform a succession of simple random kicks.

Moreover, Table 5.2 underlines that only Great Deluge with a strong perturbation is always privileged by irace. Indeed, all the proposed configurations use 1.5 or 2, the two largest values for the GD `UpperboundRate` parameter. Great Deluge is a heuristic which, like Simulated Annealing, accepts non-improving solutions if they have a fitness lower than a bound. The latter decreases during the iterations. The larger the `UpperboundRate`, the more time the Great Deluge takes and the more tolerant it is of large perturbations. That represents a clue to understanding what makes better solvers, but should not be considered only individually. That may be due to several combined effects.

5.4.2 Performance

This section evaluates the performance of configurations produced by irace. Two methodologies are employed to assess these configurations against one another and against four baselines representing HLS and its simplified variants without behavioral parameters and neighborhood operator deactivation.

The first methodology directly analyzes fitness performance to determine the most efficient method in terms of average runs. The second methodology focuses on ranking the efficiency of methods based on the average rank per instance, rather than average fitness.

5.4.2.1 Analysis by Fitness Scores

The experiment resulted in 13,860 fitness scores. These scores were obtained from the execution of 9 solvers (original HLS, HC, GD, SA, and the 5 elites) on 77 real-world instances, each run with 30 different seeds. To compare the performance globally and make fair comparisons, we normalized the fitness scores for each real-world instance using the Min-Max scaling method. Consequently, each fitness value is in a range from 0 to 1. In this subsection, we compare the fitness scores of the configurations, including baselines and the five elites. Subsequently, in the next subsection, we focus on analyzing the computed ranks to re-rank the configurations and reduce outlier behaviors.

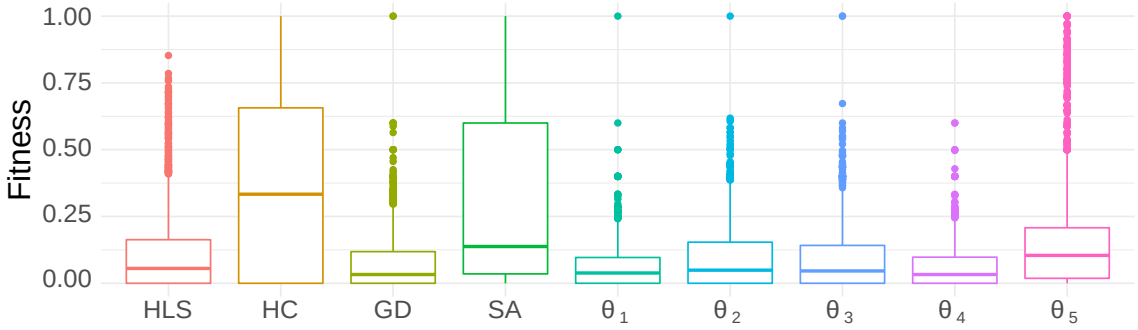


Figure 5.4: Distribution Fitness Scaled per configuration over all instances.

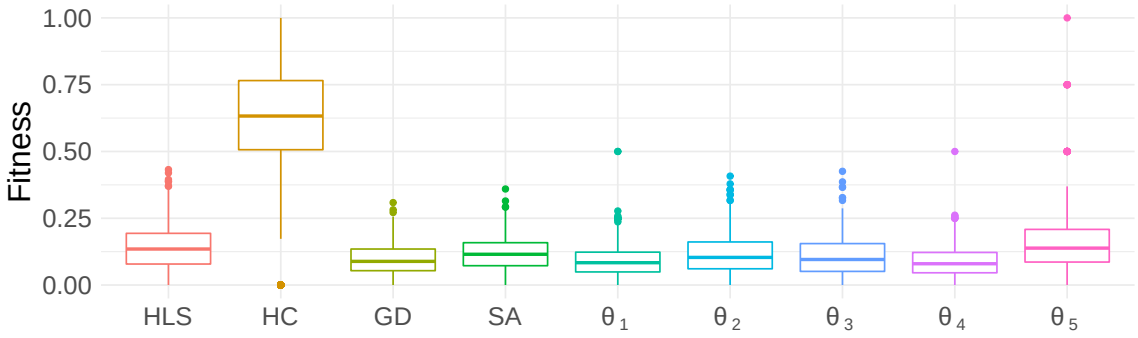


Figure 5.5: Distribution Fitness Scaled per configuration over ITC instances.

Figure 5.4, 5.5 and 5.6 show the distributions of normalized fitness scores of the configurations, respectively over all, ITC, and newly added instances.

Figure 5.4, 5.5 and 5.6 highlight the fact that most elite configurations outperform baselines. HLS, HC, GD, and SA are here to get reference points. The HC, GD and SA algorithms correspond to their HLS implementation.

Firstly, Figure 5.4 shows that the five best methods are GD, θ_1, θ_2, θ_3, and θ_4, with a slight advantage for θ_1 and θ_4. After this visual analysis, we use statistical tests to validate this observation. Without these, it is hard to make a definitive ranking. But the conclusion Figure 5.4 brings is that

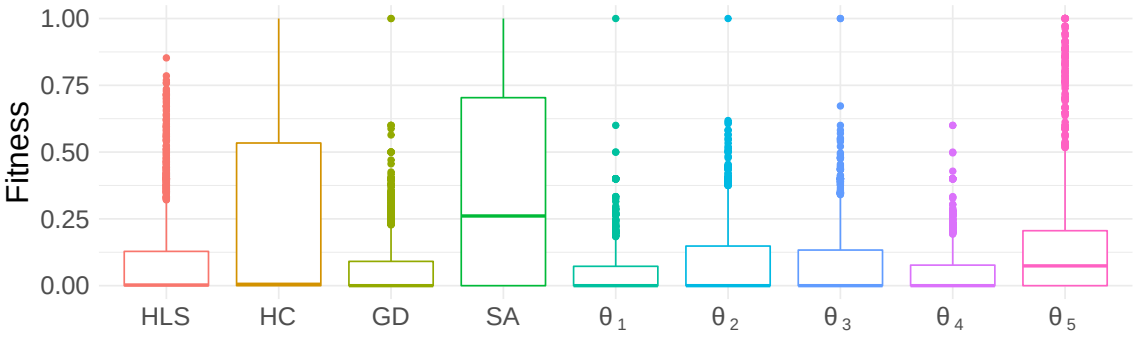


Figure 5.6: Distribution Fitness Scaled per configuration over New real-world instances.

irace has managed to generate simpler configurations than HLS. They outperform it and are more stable. We can see that, once again, SA shows disappointing results, with a behavior close to HC.

The other two figures focus on the instance group distributions (Chapter 1). Figure 5.6 shows the same behavior as Figure 5.4, so recall that the New group has 56 instances of the 77 real-world. The behavior of this group is expected to influence the overall behavior. Figure 5.5 focuses on the initial 21 instances provided by the ITC. Figure 5.5 still shows the same behavior for the best configurations: GD, θ_1, θ_2, θ_3, and θ_4. However, there is a big difference concerning SA. Simulated Annealing is very good on ITC instances. That corroborates our conclusions in the previous section. So it could be the New instances that represent a higher difficulty for the Simulated Annealing structure.

In order to rigorously compare the configurations, we calculate the ranks of the methods following the procedure below. We first use a Friedman statistical test, with a significance level of 0.05, to check that the distributions of the normalized fitness scores for each method are different. The Friedman test obtains a p-value $< 2.2 \times 10^{-16}$, which means there is a statistical difference in global performance between configurations. This test only tells us that a difference is present. To obtain actual ranks, we proceed as follows. The mean of scaled fitness values for each solving method is computed in order to sort configurations by ascending order of mean fitness. Recall that CB-CTT is a minimization problem, so we look for the lowest values. The sorted algorithms are compared using a Wilcoxon test (also known as the Mann-Whitney test) to check for actual statistical difference. Again a significance level of 0.05 is used. The first ordered configuration (the one with the best mean) is ranked 1 and is statistically compared with the following ordered configurations until the Wilcoxon test rejects the equality hypothesis for one configuration. Then, this configuration is ranked 2 and the previous ones are ranked 1. Then, we test the configuration ranked 2 with the ordered configurations that follow. The procedure is detailed in Algorithm 12. Table 5.3 reports the ranks calculated for each configuration. The activated heuristics are specified between parentheses.

Table 5.3 (line *All*) confirms that elite configurations θ_1, θ_2, θ_3 and θ_4 outperform the other two. Moreover, θ_1 and θ_4 are the elites that give the best performance to solve real-world instances. We can remark that the order of elites given by irace is not confirmed here. This is due to a difference between training and test instances. However, the main important result here is that it is possible with fine-tuning to find configurations of HLS that better perform than the original one. To support this conclusion, we compute the frequencies of the ranks of each configuration. Figure 5.7 shows

Algorithm 12 Ranking Procedure

```
1: rank ← 1
2: index_ref ← 1
3: methods ←list of methods sorted by mean fitness
4: rank_list[methods[1]] ← rank
5: for index in index_ref + 1 to |methods| do
6:     method1 ← methods[index_ref]
7:     method2 ← methods[index]
8:     result ←Wilcoxon.test(method1, method2)
9:     if result : methods are not equivalent then
10:        rank ← rank + 1
11:        index_ref ← index
12:    end if
13:    rank_list[method2] ← rank
14: end for
15: return rank_list
```

Instance Set	HLS	HC	GD	SA	θ_1	θ_2	θ_3	θ_4	θ_5
All	4	6	2	6	1	3	3	1	5
ITC	4	5	2	3	1	3	2	1	4
New	3	4	1	5	1	2	2	1	4

Table 5.3: Ranks on Normalized Fitness values.

the computed frequencies. If we focus on HLS ranks, we see that it is outperformed by at least one other configuration on 42 instances $(77 - 35)$.

Table 5.3 also provides the rank if we separate the real-world instances into ones from the ITC 2007 competition and the others (called *New*). The ranks are globally the same, but, surprisingly, HLS gets better performance on the new instances even though it was manually tuned on the ITC ones.

We notice that elite configurations θ_1 and θ_4, the best ones, do not have a Hill Climbing phase. The three other configurations use a relatively quick HC according to their value of `IterHC`, and all of them use neutral acceptance.

5.4.2.2 Analysis by Ranks

The limitation of considering the normalized fitness values across all instances is that it fails to consider the variation in behavior for each instance. In this section, we address this limitation by incorporating per-instance rank analysis, which allows us to use instance-specific information. To achieve this, we perform a ranking procedure for each instance separately, utilizing the raw fitness scores.

The aim is also to obtain a ranking that values the average rank per instance, as opposed to the lowest average fitness. We therefore want a method that performs better on the largest number of

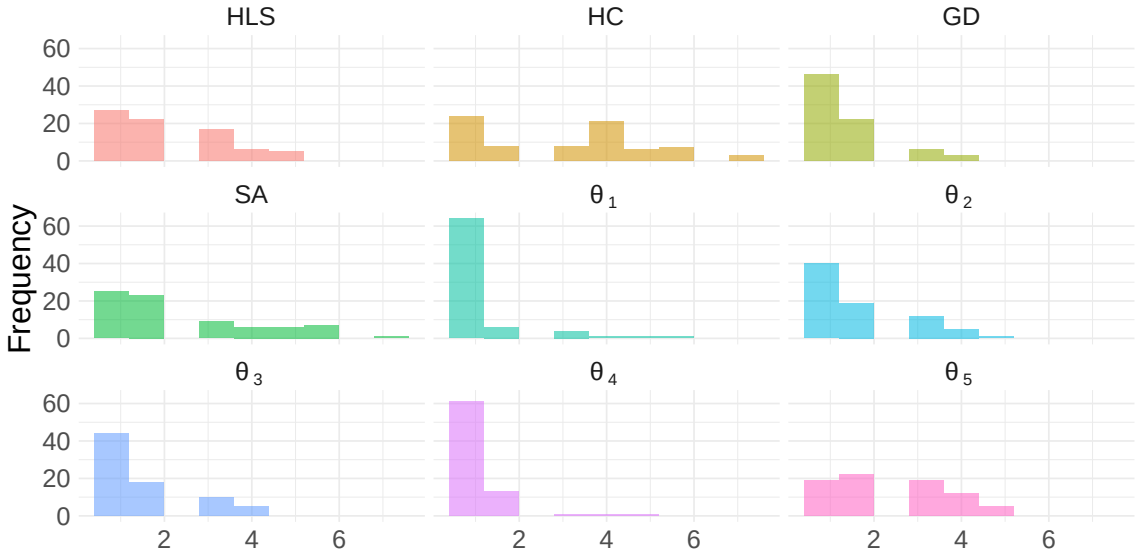


Figure 5.7: Frequency of Rank by Config.

instances, not the largest number of runs.

We first analyze the distribution of per-instance ranks. Figure 5.8, 5.9, and 5.10 show these distributions for each configuration, respectively over all instances, ITC instances, and newly added real-world instances. Unlike the previous section, these figures display the distribution of ranks rather than the scaled fitness values. Figure 5.8, 5.9, and 5.10 show similar behavior. The first four elites and the original HLS have a median rank equal to 1. These figures show that in at least 50% of instances θ_1 and θ_4 are first in terms of rank. If we analyze the raw data, there are many real-world instances where the configurations are equivalent, especially on the easier instances.

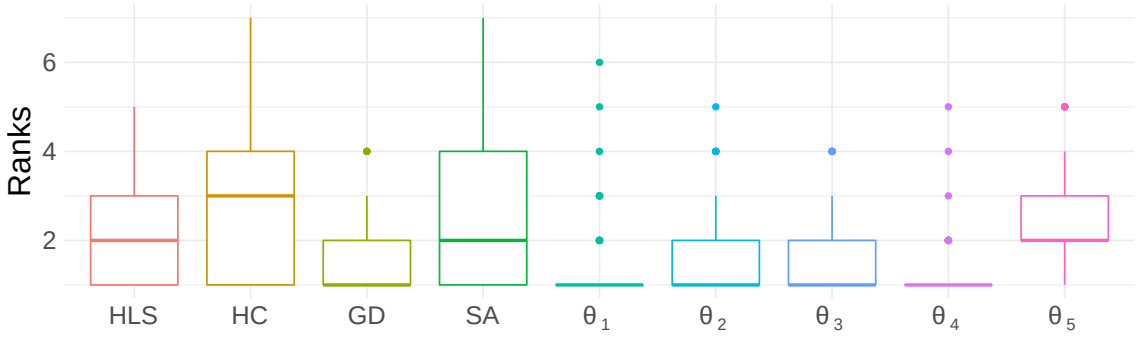


Figure 5.8: Value of Ranking by Method over all real-world Instances.

We observe three main distinct behaviors. The first one relates to θ_5. Its wider boxplot and median at 2 indicate poorer performance compared to the other variants, which aligns with the

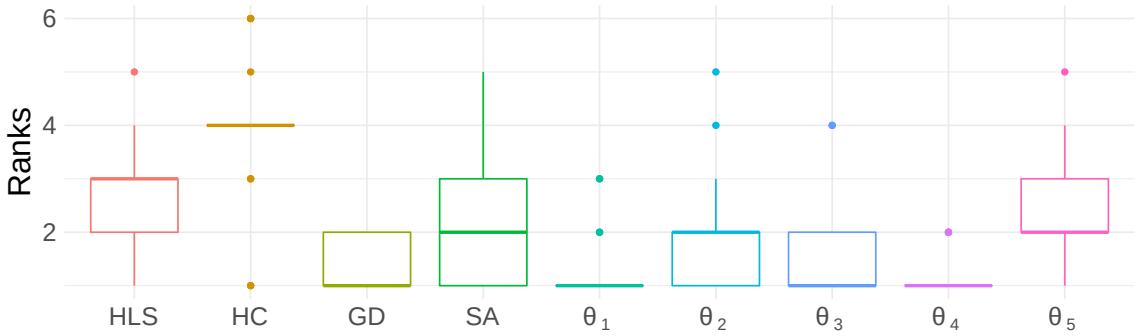


Figure 5.9: Value of Ranking by Method over ITC real-world Instances.

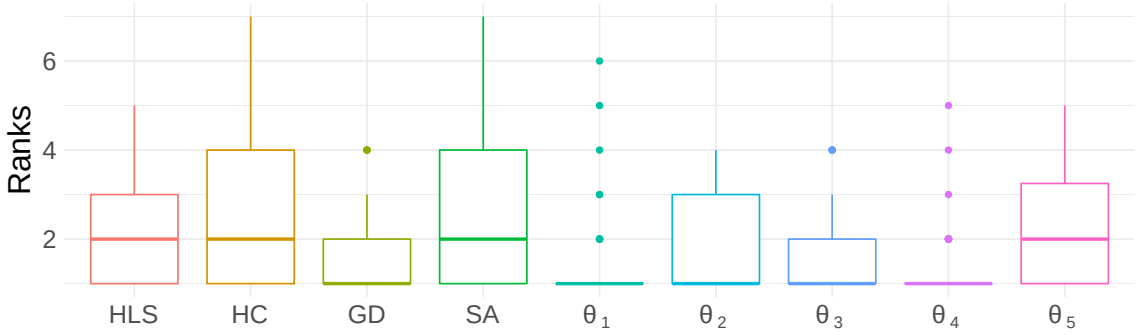


Figure 5.10: Value of Ranking by Method over New real-world Instances.

rank obtained from the normalized fitness values.

Original HLS, together with θ_2 and θ_3 have the same boxplots. However, this does not mean that the ranks are distributed in the same way, only that they share the same summary statistics. The two best variants are θ_1 and θ_4, both only use Great Deluge. These distributions of per-instance ranks seem to match to a certain extent the results obtained on the ranks of normalized fitness values. To further the observations from the analysis of Figure 5.8, we use a ranking method on the sum of ranks per instance by each solver. In addition, we explicitly isolate the ITC 2007 instances from the rest, to see whether HLS performs better on this subset of instances it was originally designed to solve.

A notable difference in the results from the fitness distribution is related to SA. This time, its behavior remains consistent across the sets of instances. It indicates that despite its performance on some instances in the ITC set, SA does not perform as well on others. When considering the fitness distribution of the previous section, SA showed a better fitness distribution on the ITC sets. However, in this case, the ranks remain the same, meaning that regardless of the subgroup of instances, SA performance varies significantly in terms of ranks and does not consistently excel.

Table 5.4 shows the ranking of each method considering ranks per instance. The ranking is also performed on subgroups of instances. The first subset, named ITC, includes the initial CB-CTT benchmark consisting of 21 instances from the University of Udine. The “New” group contains the

Instance Set	HLS	HC	GD	SA	θ_1	θ_2	θ_3	θ_4	θ_5
All	6	9	3	8	2	5	4	1	7
ITC	8	9	3	6	2	5	4	1	7
New	6	9	3	8	2	5	4	1	7

Table 5.4: Global Ranks on the ranks per instance.

remaining newer real-world instances (Chapter 1).

The new ranks provide additional information to complement the boxplots and previous ranks. With this approach, the HLS, θ_2, and θ_3 are not considered statistically similar. Indeed, HLS is ranked worse than four of the five elite configurations proposed by irace, all of which are algorithmically simpler. This time it was θ_4 who came out on top.

5.4.3 Ablation

Ablation analysis (Fawcett and Hoos [2016]) is a method employed to investigate how modifying specific parameters of a solver impacts its overall performance. It quantifies the ranking of the importance of value changes in achieving improved configurations.

The ablation analysis takes a set of instances to test several configurations. Moreover, this process needs two initial configurations. A starting configuration and a target configuration. The method tests some configurations that are between these two initial configurations. The algorithm tests the starting configuration, here HLS, on the given instances, here the real-world, test set. During the first iteration, the method generates new configurations based on the HLS default configuration, the starting method. But these configurations have one parameter value that differs from the starting configuration, that value is set to the same value as in the target. After the performance tests, statistical tests determine the best of the new configurations. The improvement or degradation compared to the previous starting configuration is memorized. On the next iteration, the new starting configuration is the one that is considered the best in the previous iteration. So, on the second iteration, the starting configuration is a configuration that has the same parameters as HLS except for one whose value is equal to the target. The program generates new configurations similar in terms of parameters to this new starting configuration except for one of them. The iterations continue until the target configuration is obtained again. At the end of the analysis, we get information about the change of values and parameters that allowed us to go from HLS to the best configuration.

Here we consider θ_1 as the target. Indeed, we keep θ_1, and not θ_4, because it is the most efficient according to irace on the train set. Anyway, θ_1 and θ_4 are very close as detailed previously.

Ablation analysis highlights the most important parameters and their values to improve performance. In this paper, ablation analysis takes the set of instances called *All* in the previous section, which contains all of the 77 feasible real-world instances. The ablation compares two solvers that both have a Great Deluge phase, which is why this parameter is not studied by the algorithm. We previously highlighted that the best configurations all activate this feature.

Figure 5.11 shows the order in which the parameters were changed. This is equivalent to giving an order of importance for these two configurations. Furthermore Figure 5.11 gives the average

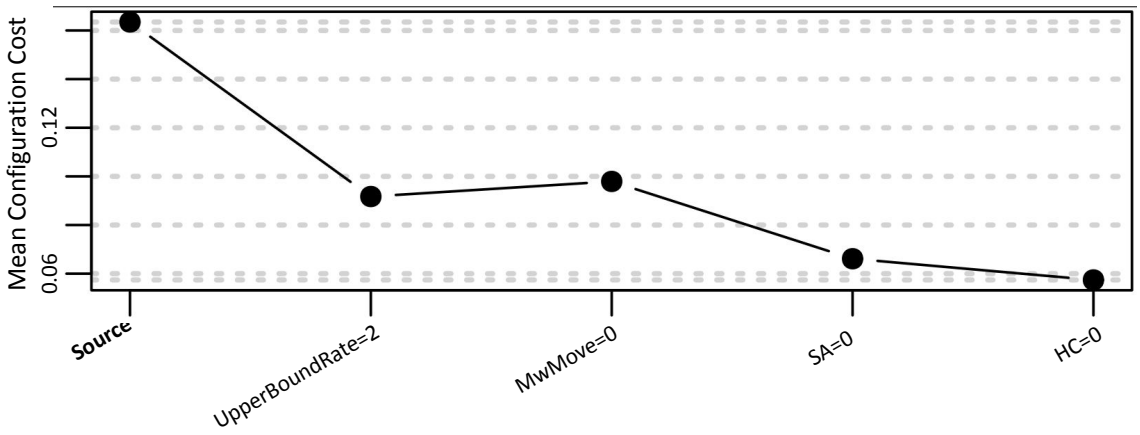


Figure 5.11: Mean configuration cost computed by the ablation analysis.

scaled cost gain on the 77 real-world instances.

The ablation method shows that the most improving change of value from HLS is to set the `UpperBoundRate` to 2. That means increasing the power of perturbation of the Great Deluge phase contributes to the efficiency of GD. Additional runs showed that setting `UpperBoundRate` to 1.43 would increase performance in the case of θ_4 configuration.

The second most important feature highlighted by ablation is `MwMove`, which corresponds to the activation of the MinWorkingDays Move operator (Section 5.3.1). Ablation results conclude that the improvement due to the activation of this operator is less significant than for `UpperBoundRate`. However, it is still significant compared to the other two. That fact validates our analysis in the previous section. This operator must slow down the solver a lot or not be efficient enough to be worth it. The importance of deactivating SA is finally minimal. In Section 5.3.1, SA was like MinWorkingDays move, always deactivated. And yet, these are not the most important parameters, i.e. chosen first. So there may be combinations of parameters where SA and GD are effective. This would require further experimentation to ascertain.

The last parameter chosen by ablation is HC. That means Hill Climbing has no concrete impact on the performance when changing HLS to θ_1. Ablation results advise setting this parameter to 1. Moreover, it considers that if HC is deactivated that decreases the performance of the solver. However, our performance tests say that variants with HC are worse than GD only. In summary, the ablation analysis showed that `UpperBoundRate` was the most important parameter in the success of θ_1 and θ_4. That offers the opportunity to work in the future on SA and HC, and their parameters. Consequently, these features become significantly important in this kind of study.

5.5 Conclusion

Experimental results have shown us three facts. The first is that irace seems to be able to use the train set selected in Chapter 3, as the configurator returns configurations that perform better on the real-world data. This shows that if a method is good on a sample of the training set then it is also good on the test set. We therefore keep this test set for the future. However, it is important to note that irace also returned θ_5 which is very bad. So even if we have four statistically better

elite configurations, it is important to note that there are differences between the training and the test set, but these do not prevent us from obtaining new and better configurations.

The second discovery is that the structure of Hybrid Local Search is not optimal. It is possible to create a method that outperforms it in number of runs and on a majority of instances. This method uses fewer neighborhood operators and does not need Hill Climbing and Simulated Annealing. It is a simple iterated Great Deluge. That highlights the usefulness of questioning complex algorithm structures to ensure that each element is cost-effective to incorporate.

The latest discovery paves the way for the work that follows. Ablation analysis shows that the most important parameter change according to it locally is to increase the value of a so-called activation parameter. Then to deactivate the MinWorkingDays move and only then, with minimal gain, to deactivate the other two heuristics. So, these results show that increasing the available values of the behavior parameters increases performance. That is why the next step consists, this time, of having wider ranges for the behavior parameters but also of increasing the number of parameters. Ablation analysis has shown that these parameters have a strong importance and that, consequently, tuning of them must be carried out to obtain the best results. Furthermore, one of the limitations of our study could be overcome by using a larger number of parameters. SA and HC do not have much impact on performance according to the ablation analysis so perhaps tuning their parameters makes them more cost-effective to use.

Chapter 6

Conclusion

Contents

This thesis focuses on a university timetabling problem: the Curriculum-Based Course Timetabling problem (CB-CTT) and its state-of-the-art solver: Hybrid Local Search (HLS). This thesis has two main axes, the first being to study the search landscape to extract and understand the complexity of the problem and to use this data to improve the performance of the solvers. The second focuses on tuning HLS to improve its performance. This thesis conducts several studies to achieve these objectives.

After describing the scientific context of this thesis to understand where our work stands concerning the literature, this thesis details our analysis of the landscape of early CB-CTT instances and shows that proposed landscape representation models contain relevant information that could be used to create machine-learning models predicting HLS performance.

This thesis then details an analysis of instances using feasibility as a proxy to develop a selector that selects the most similar artificial instances to the real-world. That creates a selected dataset that is more appropriate for future tuning work.

Finally, this thesis explains how starting from HLS, we have developed and coded an Iterated Sequential Local Search (ISLS) framework that is fully parameterizable and integrates a generic local search that can instantiate HLS heuristics.

This thesis concludes with the first work using ISLS and the Automatic Algorithm Configurator irace that shows that an improved HLS can be created using the data selected in the previous work and using an ISLS forced to be HLS-like.

This chapter provides an overview of the primary contributions made in this thesis. Additionally, it highlights work conducted during this thesis that, while valuable, was not incorporated into the manuscript due to its preliminary nature. Lastly, the chapter outlines potential perspectives for further research, offering suggestions for the continuation of this thesis.

6.1 Contributions

This section provides a concise overview of the contributions made within this thesis. It outlines the primary contributions in alignment with the thesis plan structure, offering a clear and well-defined identification of these contributions.

6.1.1 Search Landscape Analysis

In this section, we introduce the novel concept of Fitness Networks as a key contribution to our work. These networks offer a way to represent the CB-CTT search landscape, providing valuable insights into its underlying characteristics. To create Fitness networks, we employ Iterated Local Search (ILS), running it multiple times on a given CB-CTT problem. During each ILS run, we record transitions between solutions with different fitness levels. These transitions are used to construct the initial Timeout Plateau Network, where nodes represent plateaus (set of solutions with the same best fitness during a Hill-Climbing phase), and edges are formed by transitions within a single run following a Hill Climbing and Perturbation phase. Subsequently, we contract the Timeout Plateau Network by fitness, resulting in a highly connected representation model.

In Fitness Networks, three distinct groups of patterns are discerned. Group A consists of nodes with low local density, representing the best solutions. Group B comprises interconnected nodes with intermediate fitness values, often encountered during exploration. Group C represents the poorest solutions, isolated from the rest, with arcs primarily connecting to Group B. An auto-

mated method identifies these different groups by detecting kinks in fitness distributions along the networks.

6.1.2 Performance Prediction Models Study

In the context of this work, a diverse set of features was computed from the Fitness Networks to support the development of a prediction model. These features covered various aspects of the network topology, including the count of nodes, plateaus, and sink nodes within the three groups or whole network: Groups A, B, and C. Additionally, statistical measures such as quantiles, means, quartiles, and other characteristics of the fitness distributions were computed for each of these groups.

This contribution consists of predicting the performance of solvers, specifically HLS, $\text{HLS}_{\text{strict}}$, and SA, by using landscape features as predictive indicators. By employing these landscape features, the study seeks to identify and validate the relevant factors that significantly influence solver performance. Predicting the minimum reachable fitness by a solving algorithm is the first step in optimizing solver performance and enables early termination when the optimal fitness is approached.

To determine the optimal setup, we applied predictive modeling techniques to predict the minimum setup required for solving instances efficiently. We evaluated multiple protocols involving different combinations of samplers, allocated exploration time, and features. This entailed using two samplers, $\text{ILS}_{\text{strict}}$ and $\text{ILS}_{\text{neutral}}$, and various time budgets. We also considered several features, including those derived from the network metrics and node fitness distributions.

Initially, the protocol relied on a single feature linked to the fitness of the best solutions discovered during exploration in zone A. However, to further enhance prediction power and robustness, we experimented with incorporating additional features.

Through this experimentation, we found that including more than one feature slightly increased prediction power and overall robustness. In most cases, these extra features could also compensate for the absence of the best single feature. This highlights the potential benefits of using a combination of features with a machine learning model to accurately predict the final fitness of a 5-minute run of a complex algorithm like Hybrid Local Search.

Our contribution demonstrates the potential of leveraging landscape data and multiple features for performance prediction, providing a promising avenue for optimizing resolution algorithms using prediction.

6.1.3 Feature Analysis for Instances

During the course of our work, a large number of benchmark instances were proposed by De Coster et al. [2022]. Most of these instances are artificial instances.

We analyzed the distribution of the instance features and used Principal Component Analysis (PCA) to identify different patterns in the data.

Our observations showed that the artificial instances exhibited a scattered distribution in the PCA-generated space, but they seemed to exhibit similar distributions as real-world instances in the feature space.

Experiments on apply the HLS construction algorithm to these new instances revealed a notable difference between the solutions obtained from artificial and real-world instances. Four real-world instances were found to be infeasible, but a higher proportion of artificial instances also faced the

same issue. Moreover, the distribution of instance features when comparing feasible and not feasible instances was significantly different.

6.1.4 Feasibility Prediction and Instance Selection

This contribution focuses on proposing a protocol for predicting the feasibility of instances based on their instance features. This protocol led to the training of the *Selector* model, which predicts whether an instance should be selected for use in the training set.

The analysis of feasibility using statistical tests, boxplot visualizations, and PCA revealed distinct distributions of instance feature values when comparing feasible and infeasible instances. That demonstrated the possibility of predicting instance feasibility. Predicting feasibility can help avoid wasted time in experiments. However, our primary interest lies in the comparison between artificial and real-world data and determining if artificial instances are suitable for training automatic algorithm configurators.

Consequently, this work produced three models, each trained on different datasets. The first, referred to as FP^R, is trained directly on real-world instances but is not intended for practical use due to the risk of overfitting and high instance-specificity.

The other two models employ artificial data. The one that uses all artificial data without discrimination exhibits poor performance and inefficacy. That indicates that many artificial instances behave differently from real-world data concerning feasibility. The second model, FP^{AC}, demonstrates good performance. It is trained on a subset of artificial instances selected for their similarity to real-world behavior. FP^R assists in selecting these instances based on their ability to predict their feasibility accurately.

These results highlight that artificial instances can be leveraged for training models or as a training dataset for techniques applicable to real-world data, provided that selection criteria are applied. Hence, we have introduced a specialized model, *Selector*, which uses well-predicted instances to specialize and determine whether to select new instances. This model proves valuable when newly generated instances are added to the benchmarks. Finally, we tested the entire pipeline. The resulting model, FP^{AS}, achieved good results with an average accuracy exceeding 0.74.

6.1.5 Iterated Sequential Local Search

During this thesis, our efforts to enhance the performance of HLS, the state-of-the-art method for CB-CTT, led us to focus on tuning field. Before this, performance analyses of HLS and its components had indicated that it was not optimal and could be outperformed. Starting from the premise that HLS is a highly efficient method, owing to its overall process and heuristics, we decided to create an HLS designed explicitly for tuning.

Tuning demands significant flexibility, especially when exploring highly diverse configurations. Therefore, we envisioned not merely an HLS framework but an Iterated Sequential Local Search framework (ISLS). This framework preserves the fundamental HLS concept of executing a sequence of several different heuristics in a loop, maybe more than 3.

Moreover, relying on fixed heuristics with predetermined acceptance and termination strategies would not have expanded the configuration space significantly. This led us to propose a generic local search (GLS) capable of instantiating the HC, GD, and SA from the original HLS. GLS can combine the acceptance criteria of one of the three basic heuristics with the termination criterion of another, thereby creating new local searches. ISLS is developed within ORKAD's MH-builder

platform. It allows for HLS instantiation while providing high configurability, all while utilizing HLS components exclusively.

6.1.6 Tuning ISLS

In this work, our focus was on tuning the parameters that enable or disable components within the global Hybrid Local Search (HLS) algorithm. Our analysis concentrated on the utility of adding or removing complexity from the solver.

To achieve this, we employed a configurator that could deactivate each neighborhood operator and HLS local searches. Surprisingly, the optimal configurations obtained through irace were found to be simpler than the basic HLS in various aspects.

The configurator, trained on artificial instances selected from a previous contribution, returned a method that utilized only one local search, i.e. Great Deluge, and deactivated one operator.

We validated this configuration on real-world instances and observed significant improvements in performance. The iterated Great Deluge outperformed the base HLS configuration in terms of both the lowest average fitness and best average rank per instance.

This validation provided support for using the artificial data as a training set for tuning. Moreover, it demonstrated that a simpler method, the Great Deluge in this case, could surpass the performance of HLS.

An ablation analysis further revealed that the most crucial factors for enhancing performance were increasing the power of the Great Deluge and deactivating a specific operator. As a result, there is a possibility of obtaining an improved HLS configuration by retaining the hybridization of the Great Deluge and Simulated Annealing components while deactivating a particular operator. This insight opens up avenues for refining and optimizing the HLS algorithm to achieve even better results.

6.2 Research Perspectives

This section outlines research avenues in the context of this thesis. It includes a first part on investigations already started and in need of further investigation. And secondly, perspectives on as yet uninitiated research topics that could be carried out in the future.

6.2.1 Short-term Perspectives

In this section, we detail research endeavors undertaken during the thesis that have not been integrated into the manuscript. These efforts encompass preliminary investigations into specific aspects, which may yet evolve into more extensive studies.

Trajectory Analysis The main goal was to apply different heuristics, including HLS, SA, HC, GD, and others, to ITC instances. The objective was to gather execution data in order to predict promising trajectories. Upon completing the experiments, we obtained data on the evolution of the best fitness, current fitness over time in seconds and iterations, and the number of evaluations for each run. Surprisingly, our findings revealed significant variations within the trajectories of the same method applied to the same instance. These trajectories displayed considerable variability. Despite our efforts to investigate time series analysis and other predictive methods for discerning the future behavior of data series, we encountered challenges. Unfortunately, we were unable to

develop a tool that would enable us to predict whether a run would result in a favorable trajectory or not. This limitation prevented us from drawing conclusive results from the trajectory analysis work.

Fitness Landscape Extension This work builds upon the works presented in Chapter 2 but with a broader scope, encompassing all feasible instances. The primary objective was to assess whether our machine-learning tools and predictors could continue to yield good results.

During our initial experiments, we observed a decrease in the predictive capacity of our models. However, this decrease was not drastic and could be attributed to the substantial increase in the number of instances considered. In response to these preliminary findings, we conducted an in-depth analysis of the instances to explore the possibility of employing clustering techniques. The goal was to create clusters that would enable the training of performance prediction models specific to each cluster.

The instance analysis resulted in a published contribution, which is detailed in Chapter 3. Despite being incomplete work, this endeavor led to other published work that proved to be quite time-consuming. Nonetheless, the clustering of instances to predict algorithm performance remains a promising avenue for future exploration and one of our potential perspectives.

More Real-World Instances Increasing the robustness of our work was a primary objective, driven by the need to accommodate new data. To achieve this, we initiated the process by retrieving the university timetables from the new ITC 2019 benchmark and from Polytech Lille engineering school. We initiated the process and created an in-development converter to transform these university timetabling problems into CB-CTT problems. However, the need for numerous manual adjustments due to the converter ongoing development made the process time-consuming, and its cost-effectiveness was questionable. The addition of 60 new real-world instances from other institutions by De Coster et al. [2022] has not only diversified our dataset but has also reduced the immediate need to further expand the volume of instances, contributing to the overall robustness of our research.

Distance and Heuristics One of our recent major projects was to quantify the changes experienced by a solution based on the solver used or the heuristics applied. We aimed to leverage the distance metrics introduced in our first published paper (Feutrier et al. [2021]) to compare the initial and final solutions produced by local searches in the HLS structure.

The intended outcome was to demonstrate a correlation between the distance measures and the solver employed. However, the results did not align with our expectations. Surprisingly, the distance did not appear to be correlated with fitness, and it did not exhibit consistent behavior across different solvers, including SA, HC, and GD. As a consequence, further investigation is required to comprehend and address these deviations.

6.2.2 Medium-term Perspectives

This section introduces several unexplored possibilities for future work, distinct from the previous section. The objective is to highlight the diverse perspectives that can be pursued as a consequence of the work conducted in this thesis.

Optimal Instance clustering The first idea for future work involves identifying the optimal combination of data for effectively clustering problem instances. This entails collecting both instance features and landscape data for each instance. Subsequently, various automatic clustering algorithms like Kmeans, Hclust, DBSCAN, and Spectral Clustering would be tested to determine the most suitable approach.

To give concrete application to instance clustering, two options are proposed. The first option is to explore adapting performance prediction models on instances and train them based on clusters, evaluating the differences in predictive power. This means reworking the work begun on the extension of Chapter 2 to prediction.

The second option involves focusing on tuning. Clustering instances and then tuning the best configuration for each cluster would allow for easy resolution of an instance by assigning it to the cluster with its optimal configuration. To validate the effectiveness of clustering, the chosen clustering methods should exhibit significantly better performance than the most efficient method on all instances without clustering. This validation process will necessitate specialized cluster methods that outperform the non-clustered approach across all instances.

The second option offers the chance to show the most suitable methods for various instance types. For instance, it might reveal that Hill Climbing and Simulated Annealing are effective for simpler instances, while Hill Climbing and the Great Deluge outperform in handling highly complex instances.

Analysis of Neighborhood Operator The second idea combines analysis and tuning to enhance algorithm performance. In the first part, we aim to analyze the generated neighbors and explore the direct relationship between the chosen neighborhood operator and the resulting solution. To achieve this, we will create a diverse set of initial solutions, including optimized solutions obtained after a specific time budget (e.g., the best solution after 1 minute of Great Deluge and after 3 minutes). This approach will provide us with a wide range of solutions. For each solution, we will generate a set of neighbors using different neighborhood operators.

The goal is to examine the results in terms of soft constraints and distance. We seek to establish a connection between each operator and the type of modification it introduces. For instance, does the CurriculumCompactness Move operator primarily reduce soft constraints? Does it lead to significantly different solutions, or are the changes minimal?

In the second part, we will use the insights from the analysis to select an adaptive online operator. This involves manipulating the operator selection probabilities based on solution quality and violation values for each soft constraint. The algorithm will dynamically adjust its neighborhood based on the current situation.

To evaluate the effectiveness of the approach, we will compare the results with those obtained from the Hybrid Local Search (HLS) and the basic Great Deluge (GD). Additionally, we will set up an adaptive HLS and GD for comparison, as HLS using different methods may exhibit varying behaviors due to their distinct strategies.

Instance Generation One of the promising future work perspectives is to shift the focus from optimization to instance creation. The objective would be to leverage the work of De Coster et al. [2022] presented in Chapter 3. This work enabled us to generate a large number of artificial data, where feature value instances of new instances were created using a generator inspired from the literature.

There are two main objectives for this perspective. Firstly, we aim to manipulate the instance generator code to ensure that instances are more frequently feasible and adhere to real-world rules. This could be achieved through the use of machine learning tools or verification algorithms to create feasible instances and manipulate the complexity within the generator. This would provide an opportunity to generate new feasible instances for the train set.

The second objective is to create a procedure akin to SMOTE (Chawla et al. [2002]) for real-world instances. This procedure would duplicate real-world instances while introducing simple mutations to make them unique. The intention is to increase the volume of test data for more significant testing. By duplicating and mutating instances, we can increase the dataset with diverse variations while retaining the underlying characteristics of the real-world instances.

The idea is to use instance clusters to guide the duplication process and increase the number of instances per cluster. As clusters are expected to group together similar instances, this approach would ensure that the generated instances remain representative of the patterns in the real-world data.

Automatic Algorithm Configuration with global ISLS Configuration Space and Time Budget variation This work aims to expand upon the work presented in Chapter 5 by exploring a more extensive configuration space with a strong emphasis on generating new local searches. This larger configuration space includes providing irace with a more generous budget for discovering the most effective ISLS configuration for real-world instances. After analyzing these configurations and assessing their performance, we intend to find this HLS-independent algorithm capable of outperforming it.

Subsequently, we will investigate the impact of time budget on the configurations generated by irace. Preliminary analyses have shown that for easy instances, HLS often finds the best fitness in under 1 minute. Thus, the following 4 minutes are useless.

To this end, we will experiment with different time budgets: 10, 30, 60, 420, 600, and 1200 seconds, mapping out the relationships between ISLS configurations and time budget. We will also determine which elements to add in irace to utilize the available time effectively. An analysis of these results will highlight the strategies and Hybrid Local Search that increase the performance with enough time.

Finally, we will compare the top configurations across different time limits to demonstrate the significant performance gains achieved with increased time allocation.

In summary, this work will harness the full range of ISLS flexibility to craft configurations finely attuned to specific time constraints. These diverse configurations will offer a tangible understanding of the most effective components within varying time frames and provide insights into which local search strategies should be employed to achieve optimal performance over short or extended durations. Furthermore, it will shed light on the performance gains obtained as we transition between different time budgets.

Bibliography

Salwani Abdullah and Hamza Turabieh. On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. *Information Sciences*, 191: 146–168, 2012.

Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 142–157. Springer, 2009.

Roberto Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, 218:71–91, 2014.

Mutsunori Banbara, Katsumi Inoue, Benjamin Kaufmann, Tenda Okimoto, Torsten Schaub, Takehide Soh, Naoyuki Tamura, and Philipp Wanko. teaspoon: solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, 275:3–37, 2019.

Thomas Bartz-Beielstein. Experimental analysis of evolution strategies: Overview and comprehensive introduction. Technical report, Universität Dortmund, Dortmund, Germany, 2003.

Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. A simulated annealing approach to the curriculum-based course timetabling problem. In *Proc. of the 6th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA-13)*. Citeseer, 2013.

Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, 65:83–92, 2016.

Antonio Benítez-Hidalgo, Antonio J Nebro, José García-Nieto, Izaskun Oregi, and Javier Del Ser. jmetalpy: A python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, 51:100598, 2019.

Jeremias Berg, Emir Demirović, and Peter J Stuckey. Core-boosted linear search for incomplete MaxSAT. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings 16*, pages 39–56. Springer, 2019.

Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.

Edmund K Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

Edmund K Burke, Adam J Eckersley, Barry McCollum, Sanja Petrovic, and Rong Qu. Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research*, 206(1):46–53, 2010a.

Edmund K Burke, Jakub Mareček, Andrew J Parkes, and Hana Rudová. A supernodal formulation of vertex colouring with applications in course timetabling. *Annals of Operations Research*, 179: 105–130, 2010b.

Valentina Cacchiani, Alberto Caprara, Roberto Roberti, and Paolo Toth. A new lower bound for curriculum-based course timetabling. *Computers & Operations Research*, 40(10):2466–2477, 2013.

Sébastien Cahon, Nordine Melab, and E-G Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

Chun Yew Cheong, Kay Chen Tan, and Bharadwaj Veeravalli. A multi-objective evolutionary algorithm for examination timetabling. *Journal of Scheduling*, 12:121–146, 2009.

Batuhan Mustafa Coşar, SAY Bilge, and Tansel DÖKEROĞLU. A new greedy algorithm for the curriculum-based course timetabling problem. *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, 11 (2):1121–1136, 2023.

Steven P Coy, Bruce L Golden, George C Runger, and Edward A Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2001.

Fabio Daolio, Arnaud Liefooghe, Sébastien Verel, Hernán Aguirre, and Kiyoshi Tanaka. Problem features versus algorithm performance on rugged multiobjective combinatorial fitness landscapes. *Evolutionary Computation*, 25(4):555–585, 2017.

Arnaud De Coster, Nysret Musliu, Andrea Schaerf, Johannes Schoisswohl, and Kate Smith-Miles. Algorithm selection and instance space analysis for curriculum-based course timetabling. *Journal of Scheduling*, pages 1–24, 2022.

Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5:65–89, 2006.

Luca Di Gaspero, Andrea Schaerf, et al. Easylocal++: An object-oriented framework for the design of local search algorithms and metaheuristics. In *MIC'2001 4th Metaheuristics International Conference, Porto, Portugal*, pages 287–292, 2001.

Marcelo Branco do Nascimento and Antonio Augusto Chaves. An automatic algorithm configuration based on a bayesian network. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

Gunter Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993.

Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

Chris Fawcett and Holger H Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22:431–458, 2016.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Methods for improving bayesian optimization for automl. In *Proceedings of the International Conference on Machine Learning*, 2015.

Thomas Feutrier, Marie-Éléonore Kessaci, and Nadarajen Veerapen. Investigating the landscape of a hybrid local search approach for a timetabling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1665–1673, 2021.

Thomas Feutrier, Nadarajen Veerapen, and Marie-Éléonore Kessaci. Exploiting landscape features for fitness prediction in university timetabling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 192–195, 2022.

Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the Theory of NP-Completeness.*, 1979.

Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.

Fred Glover and Manuel Laguna. *Tabu search*. Springer, 1998.

Fred W Glover, Mark Parker, and Jennifer Ryan. Coloring by tabu branch and bound. *Cliques, Coloring, and Satisfiability*, 26:285–307, 1993.

David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988. ISSN 1573-0565.

Oğuzhan Hasançebi, Serdar Çarbaş, and Mehmet Polat Saka. Improving the performance of simulated annealing in structural optimization. *Structural and Multidisciplinary Optimization*, 41: 189–203, 2010.

Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. The theory and practice of simulated annealing. *Handbook of Metaheuristics*, pages 287–319, 2003.

Sebastian Herrmann, Gabriela Ochoa, and Franz Rothlauf. Pagerank centrality for performance prediction: the impact of the local optima network model. *Journal of Heuristics*, 24:243–264, 2018.

Myles Hollander, Douglas A Wolfe, and Eric Chicken. *Nonparametric Statistical Methods*, pages 139–146. John Wiley & Sons, 2013.

Holger H Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous Search*, pages 37–71. Springer, 2012.

Holger H Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier, 2004.

Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, 52:2191–2233, 2019.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer, 2011.

Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2): 167–187, 2014.

James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial Optimization*, volume 1. Springer, 2011.

Lars Kotthoff. *Algorithm Selection for Combinatorial Search Problems: A Survey*, pages 149–190. Springer, 2016.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4:87–112, 1994.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

Arnaud Liefooghe, Fabio Daolio, Sébastien Verel, Bilel Derbel, Hernan Aguirre, and Kiyoshi Tanaka. Landscape-aware performance prediction for evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 24(6):1063–1077, 2019.

Leo Lopes and Kate Smith-Miles. Pitfalls in instance generation for udine timetabling. In *International Conference on Learning and Intelligent Optimization*, pages 299–302. Springer, 2010.

Leo Lopes and Kate Smith-Miles. Generating applicable synthetic instances for branch problems. *Operations Research*, 61(3):563–577, 2013.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Helena Ramalhinho Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. *Handbook of Metaheuristics*, pages 129–168, 2019.

Zhipeng Lü and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.

Gang Luo. A review of automatic selection methods for machine learning algorithms and hyperparameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5:1–16, 2016.

Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibánez, and Thomas Stützle. Automatic design of hybrid stochastic local search algorithms. In *Hybrid Metaheuristics: 8th International Workshop, HM 2013, Ischia, Italy, May 23-25, 2013. Proceedings 8*, pages 144–158. Springer, 2013.

Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

Tomáš Müller. *Constraint-based timetabling*. PhD thesis, PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005.

Tomáš Müller. Itc2007 solver description: a hybrid approach. *Annals of Operations Research*, 172 (1):429–446, 2009.

Mario A Muñoz and Kate A Smith-Miles. Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary Computation*, 25(4):529–554, 2017.

Ferrante Neri and Carlos Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

Gabriela Ochoa, Rong Qu, and Edmund K Burke. Analyzing the landscape of a graph based hyperheuristic for timetabling problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 341–348, 2009.

Gabriela Ochoa, Sébastien Verel, Fabio Daolio, and Marco Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. *Recent Advances in the Theory and Application of Fitness Landscapes*, pages 233–262, 2014.

Gabriela Ochoa, Nadarajen Veerapen, Fabio Daolio, and Marco Tomassini. Understanding phase transitions with local optima networks: Number partitioning as a case study. In *Evolutionary Computation in Combinatorial Optimization: 17th European Conference, EvoCOP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 17*, pages 233–248. Springer, 2017.

Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

José Antonio Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16:527–561, 2012.

Yasha Pushak and Holger H Hoos. Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 245–253, 2020.

Rubén Ruiz and Concepción Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005.

Rob A Rutenbar. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, 1989.

Andrea Schaerf. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence*, pages 1254–1259. Citeseer, 1997.

Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13:87–127, 1999.

Andrea Schaerf and Amnon Meisels. Solving employee timetabling problems by generalized local search. In *AI* IA 99: Advances in Artificial Intelligence: 6th Congress of Italian Association for Artificial Intelligence Bologna, Italy, September 14–17, 1999 Selected Papers 6*, pages 380–389. Springer, 2000.

Elias Schede, Jasmin Brandt, Alexander Tornede, Marcel Wever, Viktor Bengs, Eyke Hüllermeier, and Kevin Tierney. A survey of methods for automated algorithm configuration. *Journal of Artificial Intelligence Research*, 75:425–487, 2022.

David B Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning Proceedings 1994*, pages 293–301. Elsevier, 1994.

Kate Smith-Miles and Simon Bowly. Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113, 2015.

Ting Song, Mao Chen, Yulong Xu, Dong Wang, Xuekun Song, and Xiangyang Tang. Competition-guided multi-neighborhood local search algorithm for the university course timetabling problem. *Applied Soft Computing*, 110:107624, 2021.

Kenneth Sorensen, Marc Sevaux, and Fred Glover. A History of Metaheuristics, 2017.

Peter F Stadler. Fitness landscapes. In *Biological Evolution and Statistical Physics*, pages 183–204. Springer, 2002.

Jonathan M Thompson and Kathryn A Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research*, 63:105–128, 1996.

Sarah L Thomson, Gabriela Ochoa, Sébastien Verel, and Nadarajen Veerapen. Inferring future landscapes: Sampling the local optima level. *Evolutionary Computation*, 28(4):621–641, 2020.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

Marco Tomassini, Sébastien Vérel, and Gabriela Ochoa. Complex-network analysis of combinatorial spaces: The NK landscape case. *Physical Review E*, 78(6):066114, 2008.

Sébastien Verel, Gabriela Ochoa, and Marco Tomassini. Local optima networks of NK landscapes with neutrality. *IEEE Transactions on Evolutionary Computation*, 15(6):783–797, 2010.

Anthony Wren. Scheduling, timetabling and rostering—a special relationship? In *International Conference on the Practice and Theory of Automated Timetabling*, pages 46–75. Springer, 1995.

Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

Résumé Dans cette thèse, nous nous intéressons au problème du Curriculum-Based Course Timetabling (CB-CTT), un problème d'emploi du temps universitaire appartenant à la famille des problèmes d'ordonnancement. Le CB-CTT est donc un problème de recherche opérationnelle et plus précisément d'optimisation combinatoire. Les métaheuristiques sont des méthodes de résolution qui offrent de bonnes performances dans un délai raisonnable. Les métaheuristiques sont utilisées pour leur généricité qui leur permet de s'adapter et d'être appliquées sur un grand nombre de problèmes d'optimisation. Tout d'abord, nous analysons le paysage de recherche du CB-CTT pour caractériser les instances de la littérature. Différents indicateurs sont ensuite utilisés pour construire un modèle permettant de prédire la performance des algorithmes de résolution comme les métaheuristiques. De plus, nous proposons une généralisation de la méthode plébiscitée par la littérature pour résoudre le CB-CTT sous forme d'une recherche locale séquentielle itérée (ISLS : Iterated Sequential Local Search) qui permet la conception de nouvelles versions de la méthode originelle et qui surpasse ses performances. La prédiction de performance et la configuration automatique nécessitent de nombreuses instances d'entrainement. Ainsi, nous proposons également une analyse statistique des instances et définissons un modèle d'intelligence artificielle qui sélectionne les instances les plus adaptées en terme de faisabilité.

Abstract This thesis focuses on the Curriculum-Based Course Timetabling (CB-CTT) problem, a university timetabling problem belonging to the scheduling problems. The CB-CTT is, therefore, an Operational Research problem, and more specifically a combinatorial optimization problem. Metaheuristics are solving methods that offer good performance in a reasonable time. Metaheuristics are used for their genericity, which allows them to be adapted and applied to a large number of optimization problems. First, we analyze the search landscape of the CB-CTT to characterize the instances of the literature. Several indicators are then used to build a model for predicting the performance of solution algorithms such as metaheuristics. In addition, we propose a generalization of the state-of-the-art method called Iterated Sequential Local Search (ISLS) for solving CB-CTT, which allows the design of new versions of the original method and outperforms it. Performance prediction and automatic configuration require numerous training instances. We therefore also propose a statistical analysis of the instances and define an artificial intelligence model that selects the most suitable instances in terms of feasibility.