Titre: Title:	Advancements in Tabu Search Algorithms: From Learning Approaches to Hyperparameter Optimization
Auteur: Author:	Nazgol Niroumandrad
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Niroumandrad, N. (2025). Advancements in Tabu Search Algorithms: From Learning Approaches to Hyperparameter Optimization [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/65804/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/65804/
Directeurs de recherche: Advisors:	Nadia Lahrichi
Programme: Program:	DR-Mathématiques

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Advancements in Tabu Search Algorithms: From Learning Approaches to Hyperparameter Optimization

NAZGOL NIROUMANDRAD
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques de l'ingénieur

Mai 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Advancements in Tabu Search Algorithms: From Learning Approaches to Hyperparameter Optimization

présentée par **Nazgol NIROUMANDRAD**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

DEDICATION

To my parents and family,
for their immense encouragement and support...

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisors, Prof. Nadia Lahrichi, and my co-advisor, Prof. Sébastien Le Digabel, for their unwavering support, guidance, and patience throughout my Ph.D. journey. Their invaluable insights and encouragement have helped me navigate the challenges of research and have been a constant source of inspiration.

I am also sincerely grateful to the members of my thesis committee for their time and accepting to be part of this journey.

In addition, I extend my heartfelt thanks to the CIRRELT and GERAD staff for their support and assistance throughout my research.

Finally, I am profoundly grateful to my parents for their unconditional support and countless sacrifices. Their belief in me has been my greatest motivation, and this achievement would not have been possible without their encouragement.

RÉSUMÉ

Cette thèse porte sur l'amélioration de l'algorithme de recherche tabou (TS) afin de résoudre des problèmes d'optimisation complexes de manière plus efficace. Bien que TS soit une méta-heuristique puissante pour une large gamme de problèmes d'optimisation, ses performances peuvent être considérablement améliorées. L'objectif principal de cette recherche est double: (1) améliorer le processus de recherche de TS et (2) développer des méthodes efficaces d'optimisation des hyperparamètres (HPO) dans divers contextes de problèmes.

Pour atteindre ces objectifs, nous avons intégré des techniques d'apprentissage automatique, telles que la régression logistique et les arbres de décision, dans le cadre de TS, afin d'améliorer son processus de recherche. Ces méthodes d'apprentissage ont permis d'affiner l'exploration de l'algorithme en réduisant l'espace de recherche et en améliorant la prise de décision au cours de la recherche. De plus, nous avons évalué la performance de TS en utilisant trois méthodes de HPO : MADS, IRACE et la recherche sur grille (Grid Search).

La thèse est structurée autour de deux études principales. Dans la première, nous avons intégré l'apprentissage automatique à TS pour résoudre un problème d'ordonnancement des médecins, démontrant des améliorations significatives des performances dans un environnement stochastique et des résultats comparables dans des conditions déterministes. La deuxième étude est relative à l'optimisation des hyperparamètres de TS à l'aide des méthodes mentionnées plus haut, appliquées à deux problèmes distincts : le problème d'ordonnancement des médecins et le problème généralisé d'affectation multi-ressources. Nos résultats montrent que ces techniques d'optimisation améliorent les performances globales de TS, offrant une approche adaptable et efficace pour résoudre divers problèmes d'optimisation complexes.

De manière générale, cette recherche contribue au développement d'algorithmes TS plus robustes et adaptatifs grâce à des améliorations basées sur l'apprentissage et des méthodes d'optimisation des hyperparamètres, proposant ainsi des solutions pratiques à une variété de défis d'optimisation.

ABSTRACT

This thesis focuses on advancing the tabu search (TS) algorithm to address complex optimization problems more efficiently and effectively. Although TS has proven to be a powerful metaheuristic for solving a wide range of optimization problems, its performance can be significantly improved. The primary objective of this research is twofold: (1) to improve the search process of TS, and (2) to develop effective methods for hyperparameter optimization (HPO) across various problem contexts.

To achieve these objectives, we introduce machine learning techniques such as logistic regression and decision trees into the TS framework to enhance its search process. These learning methods help refine the algorithm's exploration by reducing the search space and improving decision-making during the search. In addition, we evaluate the performance of TS using three HPO methods: MADS, IRACE, and Grid Search.

The thesis is structured around two key studies. In the first study, we integrate machine learning into TS to solve a physician scheduling problem, demonstrating significant performance improvements in a stochastic environment and comparable results in deterministic settings. The second study focuses on optimizing the TS hyperparameters using the mentioned tuning methods for two distinct problems: the Physician Scheduling Problem and the Multi-Resource Generalized Assignment Problem. Our findings reveal that these optimization techniques enhance the overall performance of TS, providing an adaptable and efficient approach to solving various hard optimization problems.

In general, this research contributes to the development of more robust and adaptable TS algorithms through learning-based enhancements and HPO methods, offering practical solutions to a variety of challenging optimization tasks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

TS	Tabu Search
ML	Machine Learning
L-TS	Learning Tabu Search
DT	Decision Trees
LR	Logistic Regression
HPO	Hyperparameter Optimization
MADS	Mesh Adaptive Direct Search
ATS	Adaptive Tabu Search
PTS	Probabilistic Tabu Search
RTS	Reactive Tabu Search
RL	Reinforcement Learning
RLTS	Reinforcement Learning Tabu Search
K-NN	K-Nearest Neighbor
ANN	Artificial Neural Network
BO	Bayesian Optimization
DoE	Design of Experiments
IDL	Intensification, Diversification, and Learning
MADS	Mesh Adaptive Direct Search
IRACE	Iterated Racing Algorithm Configuration Environment
BBO	Blackbox Optimization
VNS	Variable Neighborhood Search
LVNS	Learning Variable Neighborhood Search

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Optimization problems appear in many scientific and industrial fields, including scheduling, routing, and applications in machine learning and artificial intelligence. As these problems become more complex, there is a growing need for highly efficient and adaptable algorithms. Among various optimization techniques, metaheuristics and Tabu Search (TS) have been widely used for solving difficult combinatorial problems. First introduced by Fred Glover in the late 1980s [2], TS has been developed and refined to effectively solve a wide range of optimization tasks and applications.

In recent years, research has increasingly focused on combining different methods to enhance the performance of metaheuristics, such as TS. These combined approaches, known as hybrid metaheuristics, often integrate techniques from various optimization algorithms, including those outside of the traditional scope of metaheuristics [3, 4]. Hybrid methods have shown the potential to improve the quality of solutions for difficult optimization problems, although the development of an effective hybrid algorithm remains a challenging task. As highlighted in the literature, hybrid algorithms often face transferability issues, performing well in certain contexts while struggling in others.

1.1 Learning tabu search

Metaheuristics, including TS, explore the search space through an iterative process, generating large amounts of data during the search. Although some of this data, such as tabu lists and frequency-based information, is used to guide the search, much of the knowledge gathered remains unused. Recent research suggests that making better use of this information could improve search efficiency [5]. Machine learning (ML) techniques offer the potential to extract valuable insights from these data and use them to refine the search process, making it more effective in finding high-quality solutions.

The first article presented in Chapter 4 builds on these ideas by integrating ML into the TS framework and developing a learning-based Tabu Search (L-TS) algorithm. The goal is to improve adaptability and performance by reducing the search space.

Incorporating ML into TS provides two main advantages: it can approximate complex computations, reducing computational effort, and it can guide the search more effectively by identifying promising areas of the solution space. By analyzing patterns in high-performing solutions, ML techniques can enhance decision making within TS, leading to more efficient search strategies. This work is motivated by the strong performance of TS in solving difficult optimization problems, making it a suitable candidate for studying the benefits of ML integration.

1.2 Hyperparameter optimization

Many optimization algorithms require careful tuning of algorithm-specific hyperparameters to achieve the best performance. This applies to all heuristic and metaheuristic approaches. These algorithms often involve setting a large number of hyperparameters, as noted in several studies [6–9]. Although default hyperparameter settings are usually available for an algorithm, they are often optimized for specific problem types or application areas. As a result, optimizing the hyperparameters for a particular problem can significantly improve the algorithm's performance.

Traditionally, the design and tuning of optimization algorithms have been handled manually. Developers would experiment with various hyperparameter configurations, evaluate the results, and adjust based on trial and error. While this manual approach can produce effective algorithms, it has several limitations: it is time-consuming and heavily dependent on intuition and experience, making it subjective and challenging to reproduce. Typically, only a small subset of instances is tested, with a limited range of design options and hyperparameter settings explored.

To address these limitations, automated and systematic approaches for algorithm design and hyperparameter optimization have been introduced. Methods such as experimental design [10, 11], racing methods [12], and heuristic-based configuration techniques [13–17] have been developed to improve efficiency. In addition, statistical modeling approaches [18, 19] have further automated the optimization process, reducing the need for extensive manual intervention.

In metaheuristics, hyperparameter optimization requires identifying the best hyperparameters settings based on a set of instances. The goal is to optimize algorithm performance on these instances. This problem is called a hyperparameter optimization (HPO) problem and is commonly treated as a blackbox optimization (BBO) problem [20], formulated as

$$\max_{p \in \mathbb{P}} \quad \phi(p) \tag{1.1}$$

where $\mathbb{P}$ is the set of hyperparameters, and ϕ is the objective function, provided by a *blackbox*, i.e. a process.

A key distinction in hyperparameter optimization is between offline and online approaches. Offline methods determine the best settings before solving instances, whereas the online methods dynamically adjust hyperparameters during execution [21, 22].

The second article presented in Chapter 5 explores three approaches for optimizing the hyperparameters of TS: Mesh Adaptive Direct Search (MADS) [23], Iterated Racing Algorithm Configuration Environment (IRACE) [24] and Grid Search.

1.3 Contributions of the thesis

This thesis introduces several contributions aimed at enhancing the performance of TS by integrating ML techniques and applying HPO methods. These contributions focus on improving computational efficiency, guiding the search process more effectively, and refining algorithm hyperparameters to achieve high-quality solutions in different optimization problems. The key contributions are summarized below:

1. Enhancing the search process in tabu search:
   One of the primary contributions of this thesis is the improvement of the exploration and exploitation capabilities by narrowing the search space of TS through the use of ML techniques. Using logistic regression and decision trees, the algorithm can learn to identify promising neighborhoods within the search space. The ML models analyze patterns in past search behavior to predict which neighborhoods are most likely to contain high-quality solutions. By focusing on these promising areas, TS reduces unnecessary exploration, leading to:

   - A narrower and more efficient search space, reducing computational effort.
   - Improved decision-making, as the algorithm learns from past iterations.
   - Faster convergence, while maintaining or improving solution quality.

   We investigate both online learning, where ML models adapt dynamically during the search, and offline learning, where models are pre-trained before execution. These methods provide valuable information on the structure of the search space, leading to more informed search decisions.

2. Hyperparameter optimization for tabu search:
   Hyperparameter optimization plays an important role in enhancing the efficiency of TS. This thesis evaluates different approaches to optimize the key hyperparameters of TS, such as:

  - Tabu tenure (the size of tabu list).
  - Neighborhood size (the number of candidate solutions considered at each step).
  - Sequence of visiting neighborhoods or application of moves.

  We explore and analyze three HPO methods, MADS, IRACE and Grid Search. By comparing these approaches, we can provide insight into the strengths and weaknesses of each approach, offering guidance on selecting the most suitable method for different types of optimization problems.

3. Evaluation of hyperparameter optimization methods on different optimization problems:
   The effectiveness of MADS, IRACE, and Grid Search is tested on multiple optimization problems, demonstrating their applicability to real-world problems. We consider two key case studies:

   - Multi-Resource Generalized Assignment Problem [25], which focuses on efficiently allocating resources to tasks,
   - Physician Scheduling Problem [1], which entails optimizing the schedules of physicians while considering availability and workload constraints.

Together, these contributions refine the TS framework, making it more effective in solving complex optimization problems. These innovations improve the efficiency and adaptability of TS, making it more efficient in solving different optimization problems.

CHAPTER 2 LITERATURE REVIEW

This chapter provides a comprehensive overview of the literature surrounding TS, its advancements, the integration of ML techniques, and the important aspect of HPO. These sections serve as the foundation for understanding the contributions of this dissertation in enhancing the efficiency and performance of the TS algorithm.

2.1 Tabu search and enhancements

Since its introduction, TS received great attention in the literature and significant progress has been made in improving its robustness, efficiency, and adaptability. Various modifications have been introduced to balance exploration and exploitation while reducing computational complexity. Some of the most significant enhancements include Adaptive Tabu Search (ATS) [26], probabilistic Tabu Search (PTS) [27] or Reactive Tabu Search (RTS) [28]. Additionally, hybridization strategies have been explored, integrating TS with evolutionary algorithms, quantum-inspired techniques, and machine learning-based approaches to further improve its performance. These modifications have extended the applicability of TS to a wide range of combinatorial optimization problems.

ATS extends the classical TS by incorporating dynamic search strategies that enhance efficiency. Unlike traditional TS, which relies on a fixed-length tabu list, ATS introduces backtracking for diversification and an adaptive radius for intensification [29]. These strategies allow the algorithm to adjust its search behavior dynamically, leading to more effective exploration and exploitation. The theoretical foundations, convergence proofs, and performance evaluations of ATS have been well-documented in [30].

PTS, as introduced in [31], enhances computational efficiency by evaluating only a random subset $N'(S)$ of the full neighborhood $N(S)$ instead of exhaustively exploring all possible moves. This candidate list strategy significantly reduces the search time while still maintaining high-quality solutions. The probabilistic nature of PTS allows the use of a shorter tabu list, making it computationally attractive, though it may occasionally miss high-quality solutions.

RTS, first proposed in [28], introduces a self-adaptive mechanism that dynamically adjusts the tabu list size according to the characteristics of the problem. The algorithm stores visited configurations and their corresponding iteration numbers to track revisit frequencies. When the search repeatedly visits the same solutions, the algorithm expands the tabu list size to discourage cycles, while in less critical regions, it reduces the tabu list size to encourage further exploration. RTS has been widely applied in scheduling and routing problems due to its ability to dynamically adapt to different search landscapes.

Beyond these enhancements, TS has been successfully hybridized with other optimization techniques to further improve its performance. Several studies have explored the integration of evolutionary algorithms with TS, where TS refines solutions within an evolutionary framework, or evolutionary techniques enhance the diversification and intensification capabilities of TS. Studies such as [32–34] have demonstrated the effectiveness of these hybrid approaches. In [35], a genetic tabu search algorithm was developed to solve the distributed job shop scheduling problem, outperforming existing algorithms in both computational efficiency and solution quality. Similarly, in [36] a Tabu-Harmony hybrid algorithm was proposed for task scheduling in cloud computing, leveraging the strengths of both TS and Harmony Search to optimize task allocation.

TS has also been successfully applied to logistics and supply chain optimization. In [37], a Mixed-Integer Linear Programming model was proposed to solve the Green Inventory-Routing Problem with Time Windows. The study implemented an enhanced version of TS alongside heuristic methods, such as the improved Clarke-Wright algorithm and the Push-Forward Insertion Heuristic, to optimize routing and minimize total costs.

Quantum-inspired approaches have also influenced the TS methodologies in recent years. In [38], an amplitude-ensemble quantum-inspired TS algorithm was introduced to solve the 0/1 knapsack problem, demonstrating superior search efficiency. Similarly, [39] explored quantum-classical hybrid methods for solving capacitated vehicle routing problems, showing promising results. The integration of quantum computing principles with TS is an emerging area that has the potential to further enhance the algorithm's performance. Other studies, such as [40], have further examined the potential of quantum-inspired TS for complex optimization problems.

Further adaptations of TS have been developed for multi-objective optimization problems. In [41], an extension of TS for handling multiple objectives was proposed, incorporating path relinking strategies commonly used in discrete optimization problems. Additionally, an Advanced and Adaptive TS algorithm was introduced in [42], designed for dynamic parking allocation problems. This algorithm utilizes an adaptive neighborhood generation mechanism with bi-operator competition, achieving high-quality solutions with improved computational speed.

Efforts to refine TS have also focused on identifying key solution attributes that define prohibited moves, leading to more effective search strategies. The study in [43] emphasized the importance of selecting these attributes to improve solution quality, while studies in [44, 45] explored how balancing these attributes can optimize the search process. Another important development is the introduction of multi-objective TS variants.

In summary, the evolution of TS, from adaptive and probabilistic modifications to hybrid and quantum-inspired methodologies, demonstrates a continuous effort to improve its performance across a wide range of applications. These advancements not only enhance the algorithm's search efficiency but also extend its applicability to increasingly complex optimization problems. In the following section, we explore studies that have integrated learning-based methods to further enhance the performance of TS.

2.2 Learning tabu search

The emphasis on adaptive memory within TS highlights its inherent learning capabilities. However, recent research efforts aim to further enhance this learning behavior by integrating advanced learning mechanisms. Most of the work in this area has focused on clustering strategies [46] and intensification-diversification mechanisms [47], but significant progress has also been made in incorporating machine learning techniques into TS to improve solution quality and efficiency.

In [48], the authors introduced Learnable Tabu Search, a novel hybrid approach that integrates TS with estimation of distribution models. Their method dynamically learns and updates a probabilistic model of high-quality solutions while guiding TS in exploring the search space. By integrating learning-based probability distributions, the algorithm adapts its search strategy over time, improving solution diversity and robustness.

Intensification and diversification strategies are critical components in metaheuristics, and learning-based TS approaches have been developed to refine their effectiveness. A recent study [49] introduced a relaxation-based adaptive memory programming algorithm for the resource-constrained project scheduling problem. This method leveraged primal–dual relationships to dynamically adjust the search process within an Intensification, Diversification, and Learning framework. By incorporating Lagrangian-based heuristics into a simple TS model, the approach effectively integrated dual information into the search process, improving efficiency and convergence rates.

In another study [50], a learning-based TS algorithm was proposed for a truck allocation problem. This approach introduced a trail system to track critical solution characteristics and employed a diversification mechanism to explore new regions of the solution space. By encouraging moves that were less frequently executed in previous cycles, the algorithm aimed to broaden its search capabilities. Similarly, [51] presented a hybrid intelligent-guided adaptive search combined with path-relinking for the maximum covering location problem. Path-relinking, originally proposed in [52] as an intensification strategy, was used to refine the search for high-quality solutions by blending features from an elite set of solutions, while simultaneously diversifying the search space. Additionally, [53] proposed the Modular Abstract Self-Learning TS algorithm, which employed dynamic neighborhood selection to adaptively explore the search space. This method facilitated the use of both intensification and diversification strategies to improve performance.

2.2.1 Classification and feature selection in tabu search

Classification is another learning method that can be integrated within a TS framework. Classification in this context refers to target analysis, which identifies the characteristics of promising moves based on historical search data. However, when the feature space is large, accurately predicting the classification of new instances becomes increasingly challenging.

Feature selection and weighting techniques have been incorporated into TS to improve classification accuracy while reducing computational complexity. In [54], a hybrid approach was introduced for simultaneous feature selection and feature weighting in the K-nearest neighbor (K-NN) rule, based on TS. Feature selection involves identifying the best subset of features, while each selected feature is assigned a weight to distinguish between pattern classes. Feature selection simplifies model understanding and reduces computational time, but using classifiers to build models can be time-consuming. A related study [55] extended this approach by developing an optimization model for handling large datasets. This model integrated TS with a learning mechanism that selectively evaluated only promising feature subsets, reducing the overhead associated with computationally expensive classifiers.

2.2.2 Neural networks and reinforcement learning in tabu search

Artificial Neural Networks have been widely used for pattern recognition and optimization problems, making them valuable for enhancing TS. Typically, TS has been used to optimize ANN training, with fewer studies exploring how neural networks can enhance TS procedures. One such study by [56] introduced a neural-based TS for the unit commitment problem. Their algorithm, based on an annealing neural network to generate initial solutions, which were then refined using TS. This hybridization improved solution quality while reducing computational costs.

A further enhancement was proposed in [57], where a hybrid TS-neural network model was developed. In this approach, the neural network processed input data that had not been previously explored by the TS algorithm, allowing for more informed move selection. The study demonstrated that this method significantly improved search performance in high-dimensional spaces. Similarly, [58] explored deep learning-aided TS in large multiple-input multiple-output detection, using deep neural networks to guide the search process. By integrating Convolutional Neural Networks to rank promising search regions, their method significantly reduced computational costs while maintaining competitive detection accuracy.

More recently, [59] introduced the DeepEigen-Tabu, a deep learning-enhanced TS scheme that significantly reduced computational costs while maintaining high-quality solutions. This enhancement leveraged DeepEigNet, a neural network-based approach for effective initialization, along with a PTS model that used early stopping and efficient candidate selection mechanisms.

Recent developments have also focused on reinforcement learning (RL) strategies to improve TS performance. In [60] introduced a Reinforcement Learning-based Tabu Search algorithm (RLTS), which integrated reinforcement learning mechanisms with TS to solve the max–mean dispersion problem. By dynamically updating reward structures, RLTS demonstrated superior convergence properties and solution quality. Moreover, [61] also proposed a RLTS to solve the Minimum Load Coloring Problem. Their approach adapts the tabu list size dynamically by learning from past search patterns, allowing RL to make informed decisions about which solutions to revisit and which to avoid. This adaptive mechanism significantly improves convergence speed and solution quality by avoiding premature cycles in the search process.

Learning-based approaches have enhanced the capabilities of TS, allowing it to intelligently adapt its search behavior based on previous experience. In the following section, we explore studies that have considered optimizing the hyperparameters of TS to further enhance its performance.

2.3 Hyperparameter optimization in metaheuristics

HPO is a critical component in optimizing the performance of metaheuristic algorithms. There are two main approaches to setting hyperparameter values: hyperparameter optimization (also known as offline tuning) and hyperparameter control (or online tuning). HPO involves determining best hyperparameter values before the algorithm's execution, with these values remaining fixed throughout the execution. In contrast, hyperparameter control begins with initial hyperparameter values that are adjusted dynamically during execution. Both approaches play an important role in ensuring that the metaheuristic performs well on the problem, as each algorithm has a predefined set of hyperparameters that must be calibrated to the specific characteristics of the problem.

Many studies have focused on addressing the metaheuristic tuning problem. These include techniques such as beam search [62], experimental design [11, 63], genetic programming [64] and the application of racing algorithms [12, 65]. Some approaches also combine fractional experimental design with local search [10] to efficiently explore hyperparameter configurations. More recently, model-based approaches such as Bayesian Optimization (BO), which efficiently explores the search space using surrogate models, have gained attention as well [66].

These methods are generally classified into blackbox or whitebox optimization methods [67]. Blackbox optimization methods treat the metaheuristic as a "blackbox" focusing on automated tools that systematically search for the best hyperparameter values or combinations of algorithm components. CALIBRA [10] and F-RACE [65] are both examples of the blackbox methods. The advantage of blackbox methods is that they require minimal human intervention and can explore large parameter spaces efficiently.

In contrast, whitebox optimization methods allow the user to inspect the algorithm's inner workings, providing opportunities for the designer to guide the tuning process. Examples include statistical analysis, such as fitness distance correlation and run time distribution analysis [68–70], as well as human-guided search approaches [71, 72]. The study by [73] can be placed in this line of research. Another example is the visualization of search algorithm behavior [74] which can help users better understand the algorithm's performance. However, white-box methods still rely heavily on human input, and tuning results may vary depending on the individual conducting the tuning.

Some alternative HPO approaches have been explored beyond the traditional blackbox and whitebox methods. Agent-based methods, such as +CARPS [75], use decentralized decision-making frameworks where multiple agents adjust hyperparameters collaboratively. Similarly, self-adaptive algorithms [28] employ mechanisms that allow hyperparameters to evolve dynamically based on feedback from the optimization process. These approaches can be categorized within blackbox or whitebox methodologies depending on their implementation.

The choice between blackbox and whitebox approaches depends largely on the nature of the objective function. Whitebox methods are effective when the mathematical form or gradients are known, making them suitable for structured problems such as convex or differentiable optimization. However, when the analytical form of the objective function is unknown or too complex to express explicitly, blackbox optimization becomes essential [76]. This distinction highlights the importance of selecting appropriate HPO techniques, as different optimization problems require customized strategies.

HPO frameworks have also been developed to improve solvers' efficiency. For instance, [77] highlights the importance of selecting optimal hyperparameter values customized to specific problem instances. Classical techniques such as grid search, random search, and BO remain widely used, each presenting distinct advantages. A comprehensive survey by [78] categorized offline automatic algorithm configuration methods into four main groups; Model-Free methods, Racing methods, Design of Experiments (DoE) methods, and Model-Based methods. These categories provide a structured way to analyze and compare different HPO techniques.

2.3.1 Hyperparameter optimization in tabu search

In this dissertation, we focus on hyperparameter optimization for the TS algorithm. Over the years, various approaches have been proposed to optimize TS hyperparameters, including the use of metaheuristic techniques such as evolutionary algorithms, simulated annealing, and particle swarm optimization [79–82]. For instance, [52] introduced an ATS algorithm that adjusts hyperparameters such as tabu list size and neighborhood size based on the problem's characteristics. This adaptive approach outperformed several state-of-the-art algorithms on benchmark problems, demonstrating the effectiveness of dynamic hyperparameter tuning in TS.

Each of these HPO methods presents unique strengths in exploring hyperparameter spaces, optimizing performance, and reducing computational costs. In this dissertation, we investigate how MADS, IRACE, and Grid Search can be applied to efficiently optimize TS hyperparameters. Although IRACE, Grid Search and MADS [83,84] have been used in the literature to optimize hyperparameters for various algorithms and metaheuristics, to the best of our knowledge, this is the first time MADS is being applied to optimize the hyperparameters of the TS algorithm and compared against the other two methods.

CHAPTER 3 GENERAL ORGANIZATION OF THE DOCUMENT

This thesis is structured into seven chapters, followed by supporting information in the appendices. Chapter 1 provides an introduction to general research, highlighting the motivation, objectives, and scope of the study. It sets the context for improving the TS algorithm, emphasizing the importance of solving complex optimization problems efficiently. This chapter also defines the contributions and objectives that guide the rest of the thesis.

In Chapter 2, we perform a detailed review of the existing literature. This includes exploring new studies to improve the performance of TS, as well as the integration of learning methods with TS. We also provide studies on HPO methods for metaheuristics and TS.

Chapter 4 presents the first article, titled "Learning Tabu Search Algorithms: A Scheduling Application" and published in *Computers & Operations Research*. This article explores the integration of ML techniques into the TS algorithm. Specifically, it aims to improve both exploration and exploitation during the search process.

Chapter 5 presents the second article, titled "Blackbox Optimization of Tabu Search Hyperparameters" and submitted to the *Journal of Heuristics* for publication. This article addresses the importance of efficient methods to optimize the hyperparameters of TS. It explores the role of blackbox optimization in optimizing TS hyperparameters to enhance algorithm performance in two multi-objective problems. In this context, three HPO methods, including MADS, IRACE, and Grid Search, are evaluated. The results highlight the substantial impact of well-defined hyperparameters on the efficiency and effectiveness of TS in solving complex optimization problems.

Chapter 6 offers a general discussion of the key findings of the previous chapters, highlighting the main contributions of the research. This section summarizes the findings of the integration of ML techniques and HPO methods into TS, highlighting their impact on improving optimization methods. The discussion also addresses the potential impact of these contributions on real-world applications.

In Chapter 7, we provide the concluding remarks of the thesis. This section summarizes the contributions made to the field, acknowledges the limitations encountered during the research, and suggests potential areas for future work.

Appendix A includes a published conference article, titled "A Learning Metaheuristic Algorithm for a Scheduling Application". This article explores the integration of Logistic Regression within the TS algorithm, with a specific application to a scheduling problem. The objective is to improve solution quality while reducing computational effort. The article identifies key areas where ML techniques can be applied effectively. The findings of this study provide the foundation for the subsequent research presented in the first article in Chapter 4. In addition, supporting information for Chapter 4 is provided in Appendix B, providing further details and additional data related to this article. Appendix C provides additional notes on Chapter 4, including further explanation of the integration of learning algorithms with TS and a discussion of model accuracy.

CHAPTER 4 ARTICLE 1. LEARNING TABU SEARCH ALGORITHMS: A SCHEDULING APPLICATION

Nazgol Niroumandrad
Department of Mathematics and Industrial Engineering, Polytechnique Montréal
Nadia Lahrichi
Department of Mathematics and Industrial Engineering, Polytechnique Montréal
Andrea Lodi
Department of Mathematics and Industrial Engineering, Polytechnique Montréal

Abstract Metaheuristics provide efficient approaches for many combinatorial problems. Research focused on improving the performance of metaheuristics has increasingly relied on either combining different metaheuristics, or leveraging methods that originate outside the field of metaheuristics. This paper presents a learning algorithm for improving tabu search by reducing its search space and evaluation effort. The learning tabu search algorithm uses classification methods in order to better motivate moves through the search space. The learning tabu search is compared to an enhanced version of tabu search that includes diversification, intensification and three neighborhoods in a physician scheduling application. We use the deterministic case to test the design of the algorithm (features and parameters) and as a proof of concept. We then solve the stochastic version of the problem. The experimental results demonstrate the benefit of using a learning mechanism under stochastic conditions.

Keywords Learning tabu search; learning metaheuristics; combinatorial problems; logistic regression; decision trees

History This article is published in *Computers & Operations Research,* 170 (2024): 106751

4.1 Introduction

In recent years, there has been extensive research on combining various methods to improve the performance of metaheuristics. Some of this research originates from other optimization algorithms, while many others originate from outside the field of metaheuristics. These approaches are referred to as hybrid metaheuristics [3, 4]. Choosing an adequate combination of methods and algorithmic concepts can be a critical factor in achieving high-quality performance when solving hard optimization problems. Developing an effective hybrid approach is generally difficult, and the wealth of literature on the subject demonstrates that it is non-trivial to generalize a particular hybrid algorithm. They indeed suffer from transferability; an algorithm may work well for some problems, but perform poorly for others.

Metaheuristics explore the search space using an iterative process, during which they generate a large amount of dynamic data. Some of the information is collected and leveraged, for example the frequencies and tabu information that is used during the search process. However, we argue that metaheuristics do not fully exploit the explicit knowledge discovered during the search process, and that making better use of this information to guide the search space will make the search process more efficient. The main objective of this paper is to demonstrate that metaheuristics, particularly tabu search, can behave more intelligently and efficiently by gathering and using the data generated during the search process. We believe that more advanced techniques such as machine learning (ML) can best extract and use this valuable knowledge. These techniques facilitate a better exploration of the search space, and are capable of finding solutions that are unattainable without these learning algorithms. To measure the performance of the learning method we run each algorithm and report the amount of effort it takes to reach the best solution, where effort is in terms of number of evaluations, number of iterations and computation time.

Machine learning techniques can help improve an algorithm in two ways [5]. During the search phase of metaheuristics, learning can be used to build approximations that replace heavy computations. The learning methods can also explore the space of decision variables and improve the ability to make algorithmic decisions. To achieve this improvement, the relevant knowledge is extracted from the behavior of the best-performing variables.

Tabu search (TS) has been successful in solving many hard optimization problems, which is why we believe it is a good candidate for studying the impact of incorporating our learning methods into a metaheuristic. We propose a novel learning tabu search algorithm using classification methods in the context of a physician scheduling problem. To the best of our knowledge, there is no comprehensive study on integrating ML techniques into TS to explore the search space. Most studies on learning metaheuristics evolve in the context of clustering or intensification and diversification strategies. In particular, [46] studied a clustering method that depends on frequency-based memory applied to high-quality solutions. In comparison, [47] studied a diversification-based learning method.

Tabu search has been hybridized with numerous methods, most commonly with evolutionary algorithms. The hybridization can either be an evolutionary method that employs tabu search to generate improved solutions, or a tabu search method that benefits from an evolutionary search as a diversification or intensification strategy. References [32–34] are examples of this evolutionary/tabu search hybridization. Other examples using heuristics to modify the search space of an optimization problem [85], constraint programming [86] and exact methods [87] are available.

This paper provides an intensive study on the use of ML techniques for the design of TS. We believe this paper is beneficial for both academic and industry experts engaged in solving hard combinatorial problems. Our proposed method is used to study a physician scheduling problem, and our results are compared with those of [1]. Tabu search presented by [1] uses most of the techniques known to improve the efficiency of the algorithm, specifically: diversification, intensification and multiple neighborhoods. The algorithm proved to be very efficient when compared to CPLEX. The algorithm is adapted to solve the problem under both deterministic and stochastic conditions; under stochastic conditions the search space is larger and requires more computation effort. This paper focuses on designing a learning TS. We use the deterministic case as a proof of concept and to experiment with the design of the method. We use the number of evaluations, the number of iterations and running time to evaluate the amount of effort required to reach the best solution. We then assess the performance of L-TS in the stochastic environment. L-TS reaches comparable results for the deterministic case and outperforms TS in the stochastic one.

The rest of the paper is organized as follows. In Section 4.2, we present the algorithm and review the related studies. We present the baselines for the classification methods, and explain the proposed methods in detail in Section 4.3. We describe the instance generation procedure and present the results in Section 4.4, and Section 4.5 provides concluding remarks.

4.2 Algorithm definition and related literature

This section provides a brief introduction to Tabu Search, followed by a literature review.

4.2.1 Algorithm definition - Tabu search

Tabu search [2] moves through the search space in an iterative procedure starting from an initial solution x_0 (possibly infeasible). Let X be the solution space, each solution $x' \in X$ is selected from the neighbors of previous solution x. A neighborhood $N(x)$ is defined as all the solutions that can be reached from $x \in X$ after applying a specific move. At each iteration, all (or only a subset of) the neighbors are evaluated, and the best non-tabu solution is selected. The aspiration criteria makes an exception for a tabu move when it improves the current best solution x^*. Each solution is evaluated by the function $f(x)$. A tabu list (T_{list}) containing attributes of recently visited solutions (or attributes of moves) is maintained to prevent cycling. Solutions on the tabu list cannot be revisited for a specific number of iterations. Different strategies help search the neighborhood solutions. The strategies are intended to explore the entire relevant search space, and having an adequate combination of moves and strategies has a large impact on the quality of the results. The moves and strategies used to search the solution space are the main ingredients of tabu search methods. Over the years, tabu search has been adapted to solve many applications beyond handling uncertainty. The adaptation process commonly considers several scenarios; scenarios that are typically generated using historical data or a probability distribution. Instead of moving to the best solution in the neighborhood (deterministic environment), the best solution on average (stochastic environment) is typically preferred.

The classical TS method evaluates all (or a subset of) neighbor solutions of x and selects the best $x' \in N(x)$ as a move. This approach is computationally expensive, and often costly for large problems. Instead, we can choose a learning method as a guiding principal of our search process within the feasible space; such a learning method can help us find good solutions faster. This is how ML techniques allow us to extract knowledge, in the form of a set of rules or patterns, and from good solutions that can be used to generate even better solutions [88].

4.2.2 Literature

Leveraging machine learning techniques to solve combinatorial problems has received a lot of attention recently, but the study of learning within metaheuristics has not been given the attention it deserves. Specifically, [89] and [90] demonstrated that employing machine learning during the search process can improve the performance of heuristic algorithms. Studies that employ machine learning to enhance heuristics have pursued the following objectives:

- Algorithm selection and analysis [91],

- Learning generative models of solutions [92],
- Learning evaluation functions [93],
- Understanding the search space [94, 95].

Algorithms that improve the search performance by learning an evaluation function for predicting the outcome of a local search algorithm using features of states visited during the search are described by [93]. Other studies on learning evaluation functions are presented in [96–98]. In the same domain, [99] is another example of a recent study on learning metaheuristics. This study proposed a learning variable neighborhood search (LVNS) that identifies quality features simultaneously. This information is then used to guide the search towards promising areas of the solution space. The LVNS learning mechanism relies on a set of trails, where the algorithm measures the quality of the solutions. This idea comes from ant colony optimization, and the trail system is inspired by the pheromone trails ants use to mark a path.

Machine learning has also been employed to prune the search space of large-scale optimization problems by developing pre-processing techniques [100], [101]. Some other studies used ML-based methods to directly predict a high-quality solution [102,103]. Moreover, [104] and [105] provided a review of literature on the subject of the benefits of using ML with metaheuristics, and also discussed potential future areas of research.

We propose a learning tabu search algorithm that builds on these previous studies with enhanced classification methods that guide the search through the solution space of hard combinatorial problems. We observe that the emphasis on adaptive memory within tabu search represents the nature of its learning mechanism. In contrast, most previous works focused on clustering [46] or intensification/diversification [47] strategies. In metaheuristics, intensification and diversification strategies play important roles in the quality of solutions. In a recent study, [49] presented a relaxation-adaptive memory programming algorithm on a resource-constrained project scheduling problem. In that approach, primal-dual relationships help to effectively explore the interplay between intensification, diversification, and learning (IDL). The algorithm is designed to integrate the most cutting edge Lagrangian-based heuristic with a simple tabu search. The authors goal was to present a study of the IDL relationship when dual information is added to the search. In [50], the authors presented a learning tabu search algorithm for a truck allocation problem in which they considered a trail system for weighing combinations of important characteristics. In this study, a diversification mechanism is introduced to help visit new regions of the solution space. The authors proposed diversifying the search by performing “good” moves that were not often performed in the previous cycles.

There are also numerous studies on improving the performance of tabu search. In particular, [43] emphasized improving solution quality by selecting particular attributes of solutions and determining conditions that help to find the prohibited moves. Following that work, [44] and [45] employed a balance among more commonly used attributes. Their experiments showed that considering these attributes can significantly outperform all other methods. These outcomes underpin researchers' ongoing strategy of identifying attributes that lead to more effective methods. However, to the best of our knowledge, our study in favor of learning the characteristics of the search space during the tabu search algorithm's search procedure, is a novel contribution to the literature. We use classification models to fill this gap, and demonstrate how ML can help learn the best neighborhoods to build or change a solution during the search process.

4.3 Learning tabu search algorithm

Before getting into the details of our proposed learning TS (L-TS), we provide a brief explanation of, and the baselines for, our chosen classification methods (namely logistic regression (LR) and decision trees (DT)). Next, we present our L-TS, followed by the important characteristics of the case study we chose in order to evaluate the performance of L-TS.

4.3.1 The basics of LR and DT

In this paper, we study the impact of deploying logistic regression as a multinomial classification method, and the decision trees learning method to guide the tabu search. Here, we give a basic overview of these methods, and define some terms used throughout the paper.

The logistic regression model is one of the most important models for analyzing categorical data, and the multinomial logistic regression model is a simple extension of the binomial one. It uses predictive analysis to explain the relationship between one nominal dependent variable, and one or more independent variables.

In a binomial logistic regression model, let n denote the number of predictors for a binary response Y by the features matrix $X = (x_{ij})_{n\times d}$, where each column represents a feature and each row represents an observation. The numerical value of x_{ij} denotes the measurement of a specific feature j ($j = 1, ..., d$) in a specific observation i ($i = 1, ..., n$). Given a training dataset $(x_i, y_i)_{i=1}^{n}$, where $x_i = (x_{i1}, x_{i2}, ..., x_{id})$ represents the vector feature values of the i^{th} observation, and $y_i \in \{0, 1\}$ for $i = 1, ..., n$, where $y_i = 1$ if observation i is in class 1 and 0 if i is in class 2. In general, the aim is to classify the new observation and identify the relevant features with high classification accuracy. Assume $p(x_i)$ represents the probability for observation i when $y_i = 1$, $p(x_i) = Pr(y_i = 1|x_i)$. Logistic regression determines this probability by learning, from a training set, a vector of weights w, and a bias term [106]. More precisely, each weight w_j is a real number, and is associated with feature j of observation i.

The weight w_j represents how important that input feature is to the classification decision. The bias term b, also called the intercept, is another real number that is added to the weighted inputs. Moreover, z in equation (4.1) expresses the weighted sum of the evidence for the class, which can also be represented by the dot product notation in (4.2):

$$z_i = \sum_{j=1}^{d} w_j x_{ij} + b \tag{4.1}$$

$$z_i = w \cdot x_i + b \tag{4.2}$$

To create a probability, we pass z through the sigmoid (logistic) function, $\sigma(z)$,

$$y_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}} \tag{4.3}$$

Now, we have an algorithm that, given an observation x_i, can compute the probability $P(y_i = 1|x_i)$. Hence, we can classify observation x_i based on:

$$\hat{y}_i = \begin{cases} 1, & \text{if } P(y_i = 1|x_i) > \alpha \\ 0, & \text{otherwise.} \end{cases}$$

where α is a parameter (e.g., 0.5) and the vector w is learned by iteratively minimizing the classification error over the observations in the training set.

Now we turn to the decision tree, a supervised machine learning technique that develops a tree of decision paths from training data [107]. A decision tree is a predictive model that maps observations of an event (training data) to conclusions about its target value (label). Decision trees classify data using a set of hierarchical decisions based on features. In the tree structure, leaves represent classifications (also referred to as labels), non-leaf nodes are features, and branches represent combinations of features that lead to the classifications.

Decision trees are a widely-used predictive model for many reasons:

- They are easy to interpret and explain to non-technical audiences,

- They require minimal data preparation,
- They can handle mmany varities of data, including numerical and categorical,
- Assumptions of linearity in the training data are not required, and nonlinear relationships between parameters do not affect tree performance.

As for logistic regression, the tree is constructed by iteratively minimizing a loss function corresponding to misclassification in the training set [108].

Using a predictive method significantly reduces the number of evaluations performed during the tabu search, and both logistic regression and decision trees are powerful classification methods with their own advantages.

4.3.2 Learning tabu search

Data is an essential component of a learning algorithm; in our context data is generated during the search procedure of TS. The learning procedure contains three main phases: collecting data ($T_{collect}$), training the model (T_{train}), and applying predictions using the trained model (T_{apply}). Phase $T_{collect}$, starts with the TS and collects data related to the exploration of the search space. After collecting enough data, we start phase T_{train}, training the learning model for a specific number of iterations. Note that in phase T_{train}, the training set is updated at each iteration using the data generated. The duration of phases $T_{collect}$ and T_{train} needs to be carefully tuned to reduce the risk of issues such as over-fitting. We evaluate the impact of different settings for $T_{collect}$ and T_{train} in Section 4.4.2. I_{train} represents the iteration that phase T_{train} (training) starts and I_{apply} represents the end of T_{train} and the start of applying the prediction algorithm. After collecting enough data and training the model, the application phase T_{apply} starts. In the application phase an action is taken based on the prediction of the trained model, and after validating the elements of a tabu list (T_{list}).

The learning procedure (i.e., the three phases) may be applied in different settings, we refer to those as the online and offline learning approaches. The main difference between the two is the set of instances used to collect and to train data (i.e., to learn from). Figure 4.1 illustrates the flow of the phases. For the online learning, *for each instance to be solved*, we apply $T_{collect}$, T_{train} *on the instances* followed by T_{apply}. For the offline learning, *for each instance to be solved*, we apply $T_{collect}$, T_{train} *on a set of similar instances* (to be described in more details later) followed by T_{apply}. More details on how the application phase was used in online and offline learning are provided in Section 4.4.2.

The learning methods are very sensitive to the input information. Thus, extracting the

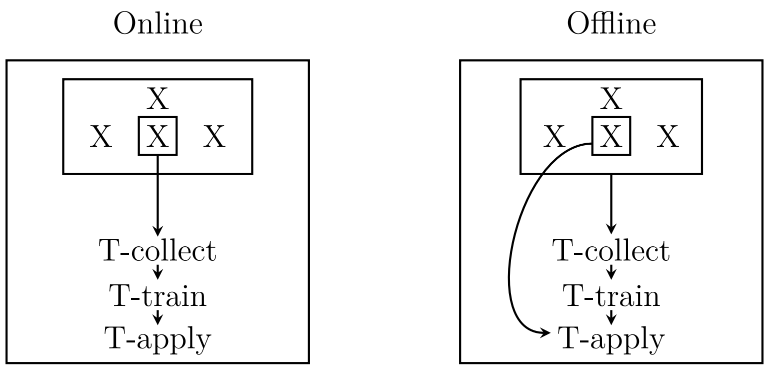


Figure 4.1 Online vs. Offline learning

features that have the greatest impact on the solution space is the first and most important step. These features represent important characteristics of the solution space and moves. We propose 10 features in total which were carefully chosen and evaluated in different settings. More details on the design of our experiments is presented in Section 4.4.2. All features are independent from the application, and only concern the data collected during the search. Table 4.1 presents these features in different categories along with the prediction output. While it will be discussed later, it should describe the level of granularity of the neighborhood to be explored, e.g., one neighbor or a subset of neighbors that should be evaluated.

Table 4.1 Input/Output features for the training model

	Input		Output
Category	Features	Format	Label
Cost improvement	Δ_x	$\mathbb{R}$	Observed Output
	$\Delta_x > 0$	$\mathbb{B}$	
Tabu	$x_i \in T_{list}$	$\mathbb{B}$	
	$\lvert t_i \rvert = T_{list}(x)$	$\mathbb{Z}$	
	$Freq_{-I^D}$	$\mathbb{Z}$	
Solution	x	$M_{\mathbb{Z}}$	
	M^*	$M_{\mathbb{B}}$	
	M'	$M_{\mathbb{B}}$	
Move	Attractiveness	$\mathbb{Q}$	
	Trail of the moves	$M_{\mathbb{Q}}$	

Let x be the current solution, x^* the best known solution, and x' the next solution. Each solution is represented as a matrix of integers. We measure cost improvement as $\Delta_x = f(x') - f(x)$. The first feature is the value of Δ_x, and the second reflects whether or not the solution is improved ($\Delta_x > 0$ in the context of maximizing). In the tabu category, we have a binary variable that determines if the accepted move belongs to the T_{list}, whether or not it has already been visited, and the tabu value for the move (when it will be free to be considered again). The frequency of the move is also considered, which indicates how many times the move has been visited so far. The next category represents the characteristics of solution. First, the solution x itself. A binary matrix (M^*) denotes the difference between the obtained solution and the last best known solution. A 1 indicates if the corresponding entry is equal, 0 otherwise. Another binary matrix (M') denotes the difference between the obtained solution and the previous solution, with the same meaning for 1's and 0's. The final category is related to the move characteristics, i.e., the attractiveness of the move and a trail matrix of the moves. These features were inspired by the recent work of [99]. Here, a trail system influences the decisions made by the ants in the Ant Algorithms, where move with high attractiveness values have a higher chance of being performed. The same list of features is used for the decision trees model.

To use tabu search in a stochastic environment, the set of input features in the stochastic learning algorithm is modified by considering the average of $f(x)$ over all scenarios at each iteration. In other words, we find the move that is the best in average over all scenarios. Thus, our objective is to find the solution $x' \in N_v(x)$ that minimizes the average solution over all scenarios $\left(\frac{\sum_r f_r(x'_{r)}}{|R|}\right)$.

The L-TS model differs from the TS mostly in the search space. Our goal is to reduce the number of evaluations in the search process. To accomplish this, we predict how promising each move is, and evaluate only highly promising neighbors, instead of every possible neighbor, of $N(x)$. This is the output of the prediction.

4.3.3 Case study: Physician Scheduling

We studied the impact of our proposed learning mechanism on a physician scheduling problem previously studied by [1]. In this section, we introduce the main characteristics of the problem, but we encourage those seeking a thorough explanation to review [1]. The goal is to find a weekly cyclic schedule for physicians in a radiotherapy department, and to assign the arriving patients to the best possible available specialist for their cancer type. Figure 4.2 shows a schematic diagram of the typical stages of the pre-treatment phase for a radiotherapy center (for patients with different types of cancer, some stages may vary).

In most physician scheduling problems, each day is divided into different slots (one or more), and each slot is assigned to a task [109, 110].

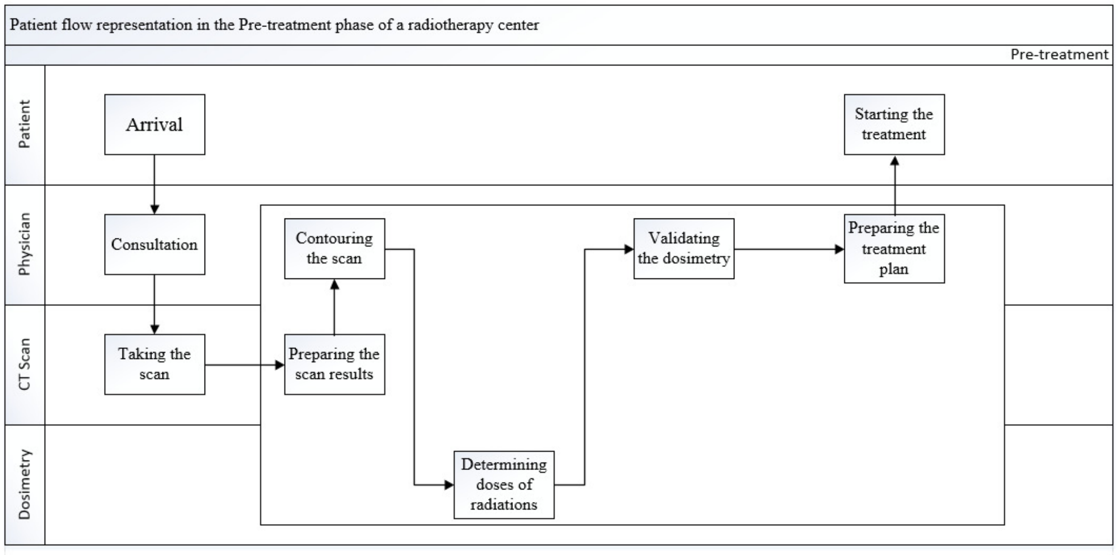

Patient flow representation in the Pre-treatment phase of a radiotherapy center
Pre-treatment
Patient
Physician
CT Scan
Dosimetry
Arrival
Consultation
Taking the scan
Preparing the scan results
Contouring the scan
Determining doses of radiations
Validating the dosimetry
Preparing the treatment plan
Starting the treatment

Figure 4.2 Stages of pre-treatment phase in a radiotherapy center [1].

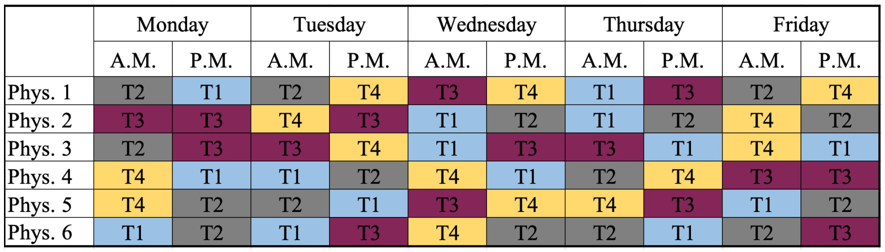
In practice, all of the tasks are scheduled each week, and some tasks are repeated. In this physician scheduling problem, the goal is to minimize the duration of the pre-treatment phase for patients. This is defined as the time from the patient's arrival day, to the day the final task is finished before treatment starts. This is usually one week for curative patients, while it should be three days at most for palliative patients. Scheduling may also take into account physician preferences.

Let I be the set of physicians, J the set of patients, T, the set of tasks and D the set of days. Each day is divided into δ time slots. The binary variables are defined as follows: p_{ti}^d is 1 if physician i performs task t, q_{ij} is 1 if patient j is assigned to physician i, and z_{ijt}^d is 1 if patient j undergoes task t on day d when assigned to physician i.

The objective is twofold: maximize the physician preferences and minimize the pre-treatment duration. Each physician has a preference for each day for each task (s_{ti}^d), and the pre-treatment duration is defined as the time from the arrival day a_j to the day the final task $|T^{o_j}|$ is performed.

$$\text{maximize} \quad \sum_{i\in I}\left(\sum_{t\in T}\sum_{d\in D} s_{ti}^d p_{ti}^d - \sum_{j\in J}\sum_{d\in D} d z_{ij|T^{o_j}|}^d - a_j\right) \tag{4.4}$$

Like most large combinatorial problems, this problem is intractable for exact methods. Thus, it is a good candidate for a tabu search metaheuristic. An exact mathematical model for this problem was presented in [1]. An example of a schedule is illustrated in Figure 4.3. In this figure, $T1$, $T2$, $T3$ and $T4$ represent the four main tasks for physicians in the pre-treatment phase in a radiotherapy center.



	Monday		Tuesday		Wednesday		Thursday		Friday	
	A.M.	P.M.	A.M.	P.M.	A.M.	P.M.	A.M.	P.M.	A.M.	P.M.
Phys. 1	T2	T1	T2	T4	T3	T4	T1	T3	T2	T4
Phys. 2	T3	T3	T4	T3	T1	T2	T1	T2	T4	T2
Phys. 3	T2	T3	T3	T4	T1	T3	T3	T1	T4	T1
Phys. 4	T4	T1	T1	T2	T4	T1	T2	T4	T3	T3
Phys. 5	T4	T2	T2	T1	T3	T4	T4	T3	T1	T2
Phys. 6	T1	T2	T1	T3	T4	T2	T2	T1	T2	T3

Figure 4.3 An example of the solution for 6 physicians, 4 tasks, 5 days and 2 time slots per day.

In practice, the arrival days, profiles and cancer types for patients are not known in advance and vary from one day/week to another. We took this uncertainty into account in the tabu search procedure by considering several scenarios (different arrival days, profiles and cancer types), typically built using historical data or a probability distribution. This strategy is common when uncertainty must be considered.

To summarize the tabu search algorithm for our physician scheduling problem, the search space is explored with three types of moves that are closely related to the decision variables (which task should be assigned to a physician on which day, and which patient should be assigned to which physician). Namely,

- *Move 1*: We seek the best sequence of task assignments achievable by swapping the assignments of a physician's current schedule from one day to another in the planning horizon.
- *Move 2*: We introduce more flexibility in a schedule by changing the repeated task. In the simple form of the problem, where the planning horizon is five days, and we have four main tasks, each physician repeats just one task. This task could be replaced by any other. In the case of two time slots, the neighborhood to explore is larger since it is likely that all tasks are repeated. Five tasks can be allocated to each of the ten time slots, and we evaluate all these options.
- *Move 3*: The third move focuses on the second decision variable, i.e., the assignment of patients to physicians. We explore all possible re-assignments.

Combinations of these three moves were used as different strategies for diversification and intensification criteria during the TS procedure. According to [1], Move 1 and 3 have the greatest impact on the quality of the solution. Therefore, the TS starts with Move 1 and applies Move 3 whenever the best solution is not improved. Move 2 is applied as a diversification strategy to encourage the method to search unvisited regions of the search space. The current solution and the best solution are updated at the end of each iteration. An aspiration criterion is also considered whenever a best solution is found that is tabu. In Algorithm 1, we refer to the TS algorithm as RunTS().

Having described the learning algorithm in Section 4.3.2, we will now demonstrate its application to the physician scheduling problem. Preliminary experiments show that applying the learning algorithm on *Move 1* has the greatest impact on the performance of the TS. In the L-TS method, we consider a subset of neighbors to evaluate, instead of evaluating all possibilities at each iteration. We predict the physician to which *Move 1* should be applied. Since data is the essential component of any learning algorithm, we collect behavioral data about the solution space (stage $T_{collect}$) before we employ the learning methods (phase T_{train} along with the application phase, T_{apply}) within the TS. The L-TS algorithm starts with the TS to collect the necessary data for I_{train}^{det} iterations, as demonstrated in Figure 4.1. (Note that superscript *det* stands for deterministic and *stoc* will be used for the stochastic environment). We re-train the learning method for a specific number of iterations (stage T_{train}), evaluate the result of the learning algorithm, and update the set of input features to encourage the method to search more promising regions. In the application phase, we use the last trained model at iteration $I_{apply}^{det} - 1$ to identify (by prediction) promising moves to build $N'(x) \subset N(x)$. *Move 2* and *Move 3* are used to diversify the search during I_2^{det} and I_3^{det} iterations respectively, and the total procedure ends when the stopping criterion ($Stop_{max}$) is reached. Our criterion is 1h of CPU time. Details of the parameter initialization step are documented in Section 4.4. Algorithm 1 presents the pseudocode of our learning tabu search algorithm.

Algorithm 1 Learning tabu search algorithm

```
1:  Generate initial solution $x^0$:
      a) Assign each task (at least once) to physicians randomly; and,
      b) for each patient: assign to the physician who performs task A on the arrival day if
         the quota is not reached, and to any other available physician otherwise;
2:  Calculate initial cost $f(x^0)$ using function 4.4 defined in Section 4.3.3. Create an empty
    $T_{list}$;
3:  $x$, $x'$, $x^* \leftarrow x^0$, $f(x) = f(x') = f(x^*) = f(x^0)$
4:  $it$, $it^*$
5:  **while** $Stop_{max} < 1h$ **do**
6:      **Stage** $T_{collect}$:
7:      **for** $it < I^{det}_{train}$ **do**
8:              $Do \quad RunTS() \quad with\ Move\ v \in \{1, 2, 3\}$
9:          Update the feature set
10:     **end for**
11:     **Stage** $T_{train}$:
12:     **for** $I^{det}_{train} < it < I^{det}_{apply}$ **do**
13:         Train the learning model
14:         Predict next neighbor
15:             $Do \quad RunTS() \quad with\ Move\ v \in \{1\} \in N'(x)$
16:         Update the feature set
17:     **end for**
18:     **Stage** $T_{apply}$:
19:     **for** $it > I^{det}_{apply}$ **do**
20:             $Do \quad RunTS() \quad with\ Move\ v \in \{1, 2, 3\}$:
21:         **if** $v = \{1\}$ **then**
22:             Predict
23:         **end if**
24:     **end for**
25:     $it \leftarrow it + 1$
26: **end while**
```

4.4 Experiments

In this section, we answer some important questions: Is learning possible during the TS procedure? Does online or offline learning have the greatest impact (these concepts will be explained later in this section)? Finally, how should we choose the parameters and features of each method?

4.4.1 Experimental Setup

We compare the results with a benchmark previously published by [1] for both deterministic and stochastic environments where performance was compared with CPLEX. We refer to this previous work as TS for the remainder of the paper. We use 21 generated pseudo-real instances where we vary both the number of new patients arriving each week, and the number of available physicians. The number of physicians varies from 6 to 10, and number of patients from 7 to 60; the problem sizes range from small instances to real-world applications. Each instance is labeled $pr-$(# of physicians, # of patients). In the deterministic case, we select one scenario to obtain a typical schedule. In the stochastic situation, we consider a subset of R for different scenarios. We refer the readers to [1] for more details on instance generation. The experiments were conducted on clusters provided by the Digital Research Alliance of Canada. To implement our learning methods, we utilized an open-source library, OpenCV/3.4.3, which was integrated into our C++ TS implementation.

4.4.2 Experimental Results

We report and analyze results in deterministic and stochastic cases based on 1) performance, 2) efficiency, 3) scalability and 4) sensitivity of the algorithm.

- **Performance** We evaluated the performance of the learning algorithm by comparing the cost value defined in equation (4.4).
- **Efficiency** The efficiency of the algorithm was validated by measuring three criteria, this is meant to assess the computation effort. First, we use the primal integral value [111]. It can be interpreted as a normalized average of the incumbent value over time (see B.2 for more information). The second criteria is the beneficial rate, that compares the improvement rates of L-TS and TS. The third and last criteria is the convergence rate (number of evaluations and the number of iterations) to assess the search space exploration and how quickly the algorithm approaches the best solution.

- **Scalability** The scalability of the algorithm was evaluated by testing in small, medium and large-scale instances.
- **Sensitivity** The sensitivity analysis was performed by evaluating the impact of different training features and methods (online/offline) on the performance of the algorithm.

All results were compared and evaluated with respect to the TS method and a random approach (in which $N'(x) \subset N(x)$ was randomly chosen) to test the performance of the L-TS method. Comparing these three approaches helps us to see the performance of each, and confirm the advantage of choosing TS over the random approach, and the advantage of choosing L-TS over TS. Thus we clearly demonstrate that learning during the process is useful, and the results are not a fluke.

The first three of the mentioned criteria (performance, efficiency and scalability), are discussed in Section 4.4.2 (*Experimental performance on the Deterministic case*) and Section 4.4.2 (*Experimental performance on the Stochastic case*). The fourth criterion, sensitivity, is analyzed in Section 4.4.2 (*Fine-Tuning and Feature Selection*).

Experimental performance on the Deterministic case

In this section, we use the experiments in the deterministic case to experiment with the design of the method, mainly for parameter tuning purposes and as a proof of concept. When not otherwise specified, we refer to online learning as our primary method of learning and will use L-TS for short (see Section 4.3.2). First, we validate our L-TS algorithm and determine the value of the parameters, i.e., the size of the T_{list}, the number of iterations in each neighborhood, the iteration to start the training phase and application phase. The values tested are all related to the size of the instances (i.e., number of patients, number of physicians and number of time blocks). For the deterministic case, we use $I_1^{det} = 1$, $I_2^{det} = 2|J| + |I| + \delta \times |D|$, $I_3^{det} = 1$, $I_{train}^{det} = 3\sqrt{|J| \times |I| \times (\delta \times |D|)}$, $I_{apply}^{det} = 2|I_{train}^{det}|$, $Stop_{max} = 1h$, and we set $\theta^{det} = 2|J| + |I| + \delta \times |D|$; the reason for these choices is discussed in Section 4.4.2. Note that the stopping criterion is computation time. This approach is best suited for assessing efficiency, as it ensures a fair comparison across all methods, regardless of their complexity or computational requirements.

Performance Table 4.2 compares the cost values of different methods and the gap columns show improvements with respect to the TS. In this table, "GAP - Best" compares the best solution obtained from 10 different runs of each method, and represents the improvements over the best solution obtained from the TS. Conversely, "GAP - Avg" represents the average

Table 4.2 Results for generated instances in the deterministic case

	Tests	GAP - Best (%)			GAP - Avg (%)		
		Random	L-TS-LR	L-TS-DT	Random	L-TS-LR	L-TS-DT
Small	pr-(6,7)	0.5	**-1.0**	**-0.7**	2.3	**-0.7**	0.0
	pr-(6,9)	1.0	0.8	1.3	1.7	**-1.4**	**-0.1**
	pr-(6,11)	2.4	0.3	**-0.8**	2.1	**-0.2**	0.4
	pr-(6,12)	2.9	0.5	1.1	3.1	0.3	0.5
	pr-(8,7)	2.7	**-0.7**	0.7	2.4	**-0.1**	0.7
	pr-(8,9)	2.2	**-0.4**	0.5	2.5	**-0.2**	0.1
	pr-(10,7)	1.5	0.0	0.4	2.3	0.1	0.7
Medium	pr-(6,20)	1.2	0.9	0.6	3.6	1.3	0.5
	pr-(8,11)	3.0	0.6	0.6	2.9	0.4	0.5
	pr-(8,12)	1.5	0.0	**-0.6**	2.3	0.0	0.1
	pr-(8,20)	3.3	0.8	**-0.8**	3.7	**-0.1**	0.8
	pr-(10,9)	2.3	0.3	0.4	2.1	0.1	0.2
	pr-(10,11)	1.7	**-0.4**	**-0.1**	2.2	0.2	0.7
	pr-(10,12)	1.5	0.1	0.6	3.0	0.4	0.5
Large	pr-(6,40)	2.6	**-4.0**	**-3.5**	4.5	**-0.4**	2.1
	pr-(6,60)	9.4	**-0.7**	1.4	5.8	**-3.6**	**-8.7**
	pr-(8,40)	2.3	**-0.5**	0.0	4.4	1.7	1.7
	pr-(8,60)	8.2	1.3	5.2	8.5	3.3	5.3
	pr-(10,20)	3.9	0.0	1.2	4.1	0.4	1.2
	pr-(10,40)	3.4	0.4	0.2	4.4	0.1	2.3
	pr-(10,60)	1.5	0.0	**-1.1**	6.2	2.2	1.5
	Average:	2.81	**-0.08**	0.31	3.53	0.18	0.52

values from ten different runs. This speaks to the methods consistency. A negative value in the GAP columns indicates that the learning tabu search has improved the solution on average. We observe that logistic regression succeeded in reaching the best solutions (0.08% on average over all the instances): 7/21 instances are improved, 4/21 same solution is reached and for the remaining 10/21, the gap is between 0.1% and 1.3%. We see more improvement on average in large instances (0.5%). When comparing the averages (second set of columns), performance of TS and L-TS-LR are very similar (0.18% GAP). Same observations can be made for the decision tree based learning algorithm : 7/21 instances are improved, 1/21 reaches same solution and the gap is between 0.2% and 5.2% for the remaining 13/21 instances. Solutions are very similar when looking at the averages over ten runs, which speaks to the consistency of the algorithm. Finally, both learning methods, L-TS-LR and L-TS-DT, perform much better than the Random algorithm. This confirms that the learning algorithms receive valuable information that guides the exploration.

Efficiency The value of the cost function does capture the benefits of a specific method only from that aspect. In what follows, we provide comparisons based on the primal integral, the beneficial rate and the convergence rate.

Primal integral value. Figure 4.4 illustrates the primal integral value for all instances. The premise of the primal integral comes from [111]. It is defined such that the smaller the primal integral value, the better the expected quality of the solution if we stop the solver at an arbitrary point in time. The value is calculated based on ten runs for each instance and shows the improvement of the best solution vs time.

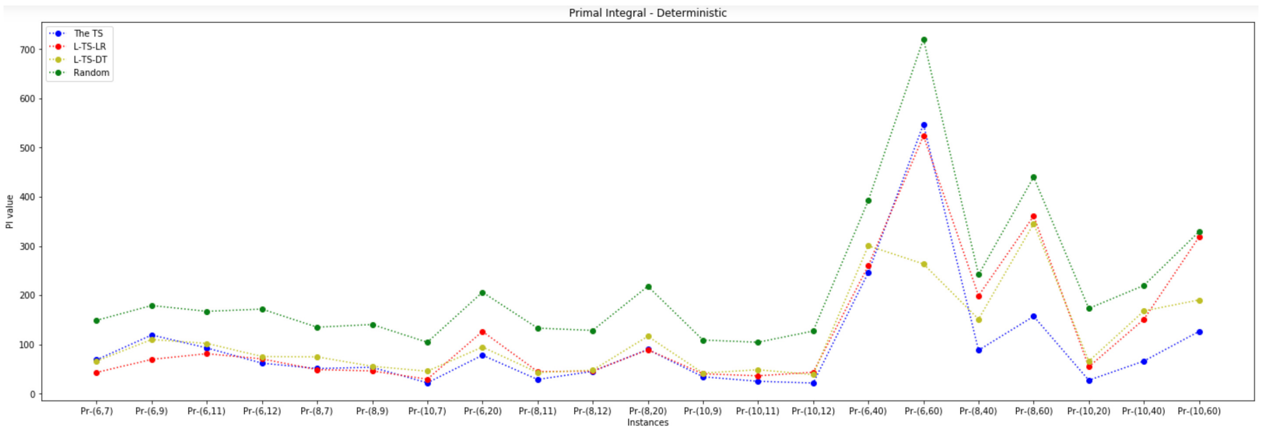
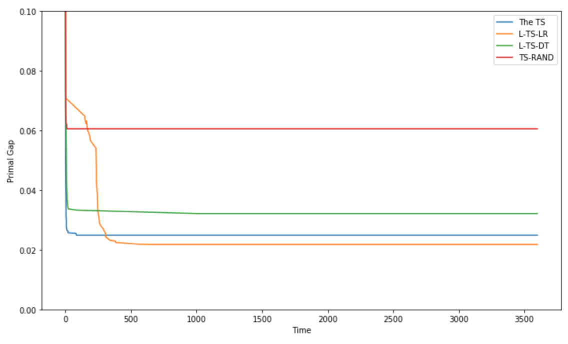


Figure 4.4 Comparing the primal integral value for different methods in the deterministic case

We observe from Figure 4.4, that the learning methods consistently exhibit comparable primal integral values across the majority of instances, except from four instances ($pr-(8,40), pr-(8,60), pr-(10,40)$, and $pr-(10,60)$). This means that the effort to reach the best solution is comparable. To provide further insight, the primal gap values over time are presented in Figure 4.5 for two specific instances: $pr-(8,20)$ and $pr-(10,40)$. In the case of $pr-(10,40)$, the primal value P for the L-TS-LR method is notably lower than TS method, and the two values are similar for $pr-(8,20)$. For these instances, the GAP between TS and L-TS-LR is 0.8% and 0.4% respectively. The figure shows that both L-TS-LR and the TS methods achieve similar results in terms of the value of the objective function, but the key driver here is the time. L-TS-LR method required more time to converge to the best solution in both cases. Running time is a key element and we discuss it through the time for each iteration and to reach the best solution.

The computation time required by each algorithm varies from one method to another and from one instance to the other. Figure 4.6 illustrates the running times per iteration for each algorithm to reach $Stop_{max}$ (Figure 4.6a), and the running times for the *Training* phase more specifically (Figure 4.6b). All box-plot graphs show the median (line) and the mean (pointer). Figure 4.6 shows that the logistic regression and decision tree algorithm require on


(a) $pr\_(8, 20)$

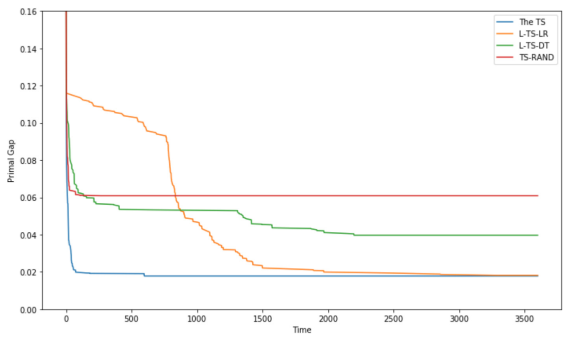
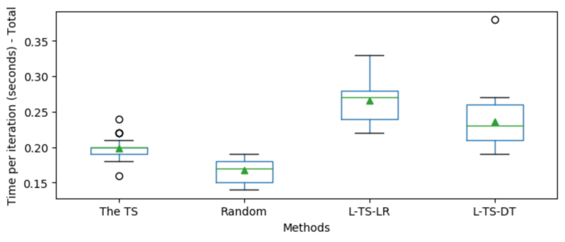

(b) $pr\_(10, 40)$

Figure 4.5 Comparing the primal gap for all methods in the deterministic case with online learning

average 0.28 and 0.25 seconds per iteration (0.20 for TS). This is due to the time needed to manipulate the training data and train the models (Figure 4.6b). Note that the learning algorithms were employed within the search process, so the pre-processing procedure, including data preparation and training time, have an impact on the total computing time. Computation time varies significantly based on several factors, including hardware specifications and the chosen implementation approach [112]. To highlight this variability, we illustrate three distinct implementations of the L-TS-LR method, aiming to demonstrate its impact on computation time (see Appendix B.3).

Figure 4.6a also shows that the time spent for each iteration was nearly identical for all methods, with an average difference of around 0.08 seconds (which is rather insignificant).


(a) Total

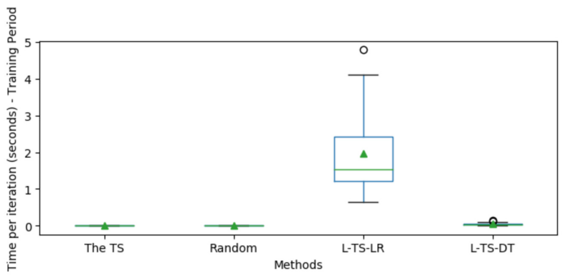

(b) Training Period

Figure 4.6 Comparing computing time per iteration for all methods in the deterministic case with online learning

Beneficial rate. The second criterion we use to compare the efficiency of the methods is the beneficial rate B. Let b_1 be the improvement rate and b_2 be the time spent increasing the rate compared to the TS for each learning method. $B = \frac{b_1}{b_2}$ is the overall beneficial rate. This rate is demonstrated in Figure 4.7.

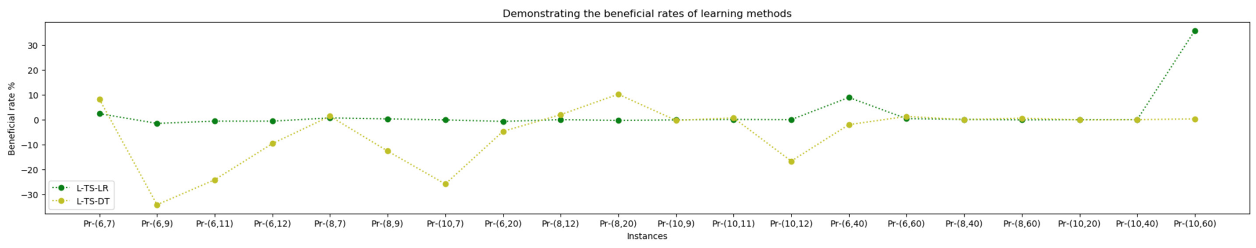


Figure 4.7 Comparing the beneficial rate for learning methods in the deterministic case

B(L-TS-LR) $= 2.1\%$ on average for all instances. This shows the advantage of L-TS-LR over the TS. Despite the efficiency of the TS method, our L-TS-LR method outperformed the TS for 7 instances (33% of our test cases). The maximum B value we obtained is 35.7%, and is from Pr-(10,60). The minimum value obtained was -1.5% from Pr-(6,9). This analysis shows that even in the worst case, pr-(6,9), we did not negatively impact the results of the TS method. For most instances, the rate B is close to 0, with a median of 0.0 for the deterministic case. This confirms that the performance of L-TS is very close to that one of the TS in the deterministic case.

Convergence rate. The last criteria to compare the different methods in terms of efficiency is the convergence rate. We define two measures : number of evaluations and number of iterations.

Figure 4.8 provides a comparison of two critical metrics: the number of evaluations conducted at each iteration and the total number of iterations required to reach the predetermined stopping criterion ($Stop_{max}$). The "number of evaluations" metric shows the number of neighboring data points that must be assessed during each iteration. It is obvious that, the figure distinctly illustrates a reduction in the number of evaluations within the learning methods, reflecting the evaluation of a smaller subset of neighbors. Also, the decrease in the total number of iterations shows that we successfully reduced the overall search space in both learning methods.

We also compare the number of iterations required to reach the best solution. Figure 4.9 illustrates that the L-TS-LR model is more consistent over different instances in terms of number of iterations needed to reach the best solution. Furthermore, L-TS-LR finds the best solution in 1378 iterations on average, while the TS finds the best solution in 1930 iterations on average. When comparing the number of iterations for the different instances, number of iterations to reach the best solution for small and medium instances is 663 and

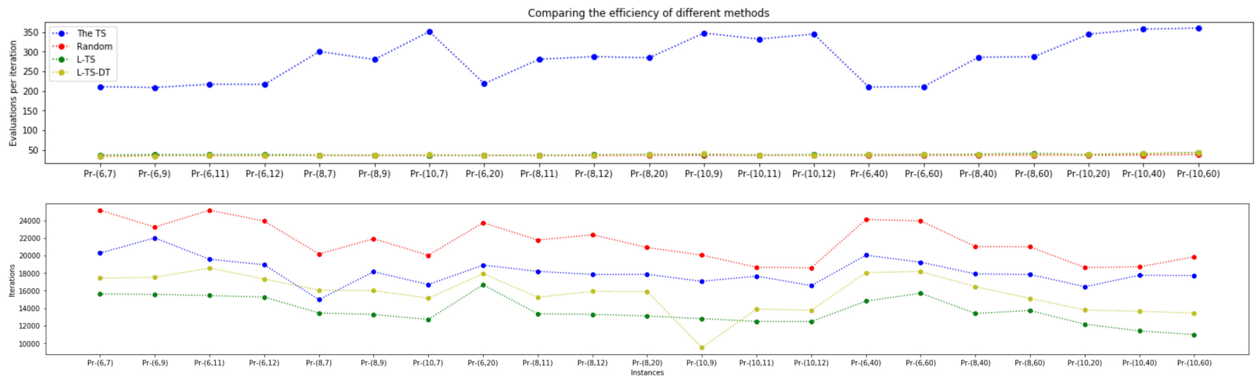


Figure 4.8 Comparing computing performance for all methods in the deterministic case with online learning

756 iterations on average, while it is 4462 and 2621 for large instances. We also note that L-TS-LR method required 1266 iterations in average to reach to the comparable solution while the TS needs 1883 iterations in average. More specifically, in 13 out of 21 instances, L-TS-LR method obtained a comparable best solution in less number of iterations comparing to the TS method. This is in concordance with previous observations; the learning phase requires time but helps reach a better solution in less number of iterations and evaluations, especially for the larger instances.

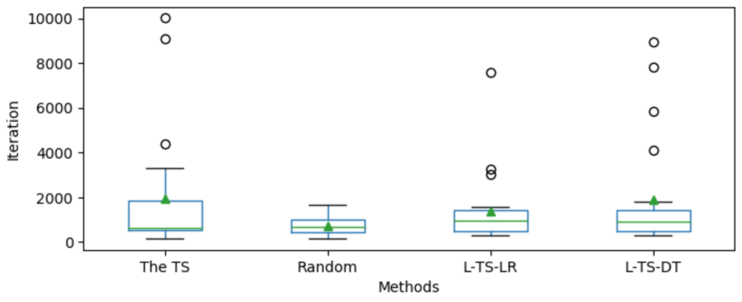


Figure 4.9 Number of iterations to reach the best solution

All of these illustrations, including the improvement in the objective value presented earlier in Table 4.2, show that the learning tabu search is able to learn during the search and is able to reach good quality solutions. Because we can enhance solutions for certain instances and typically require fewer iterations and evaluations in general, it demonstrates that the search mechanism incorporates interesting features.

In the next section, we present these experiments within different settings, first for parameter settings, and second for different testing strategies.

Fine-Tuning and Feature Selection

In this section, we explain how we chose the parameters and features for our learning method. Fixed parameters and parameters based on the size of each instance, were considered while designing our experiments. We also had different sets of parameters for L-TS-LR and L-TS-DT models. Table 4.3 gives a detailed view of the different parameter settings. We experimented with two different sets of parameters for I_{train} and I_{apply} for the L-TS method. With the first set of parameters, we used fixed values ($\{30, 60, 100\}$), and for the second set we used parameters based on the size of each instance ($\{\sqrt{|I| \times |J| \times (\delta \times |D|)}, 2\times\sqrt{|I| \times |J| \times (\delta \times |D|)}, 3\times\sqrt{|I| \times |J| \times (\delta \times |D|)}\}$). For the remaining instances, we use $\alpha = \sqrt{|I| \times |J| \times (\delta \times |D|)}$. Setting values of parameters based on the size of instances is more efficient for parameter tuning. For I_{apply}, we test the values $\{30, 60, 100\}$ and I_{train}. L-TS-LR is evaluated with different parameter settings for the regularization methods, training methods, and learning iterations. We also tested the L-TS-DT method by varying the settings for “Min sample count” and “Max depth”.

Table 4.3 Design of experiments for parameter settings

Methods	Parameters	Values
L-TS-(.)	I_{train}	$train = \{30, 60, 100, \alpha, 2\alpha, 3\alpha\ \}$
	I_{apply}	$apply = \{30, 60, 100, 2 \times train\}$
LR	Regularization method	{L1, L2}
	Training method	{Batch, MINI-Batch}
	Learning iterations	{1000, 500, 100, 10}
DT	Min sample count	$\{\ |I|\ ,\ |I| \times |J|\ ,\ |I| \times |J| \times (\delta \times |D|)\ \}$
	Max depth	{50, 10}

$\alpha = \sqrt{|I| \times |J| \times (\delta \times |D|)}$

As an illustration, in Table 4.4 we present a portion of the results from varying the sizes of I^{det}_{train} and I^{det}_{apply} parameters. This table presents the “GAP- Best” results by varying parameters and comparing the best solution obtained from 10 different runs for each parameter to the best solution obtained from the TS. The best values are in bold to highlight the difference between the settings. Observe that for problems with small sizes, the L-TS-LR method obtained better results in columns $I_{30} - I_{60}$ and $I_{30} - I_{90}$ with a 0.13% gap on average. We observe better performance with $I_{60} - I_{120}$ for Medium size problems with a gap of 0.04% on average. For large problems, $I_{60} - I_{160}$ performs better than other settings with a gap of -0.2% on average. This demonstrates the sensitivity of these parameters. Thus, these parameters should be optimized to avoid issues, such as lacking enough data to learn from, or the risk of over-fitting. We also evaluated L-TS-LR method by varying I_{train} and I_{apply} based on the size of each problem. The best improvements were -0.06%, 0.03% and -0.40% on average for small, medium and large problems respectively. By analyzing the standard deviation for each set of parameters, we observe more robustness in the last three column (i.e., when considering the size of the problem, standard deviation varies between 0.15 and 0.18).

Table 4.4 Comparing the performance of the learning algorithm by varying the size of training set and training duration

	Tests		GAP - Best (%)											
		I_{train}	I_{30}	I_{30}	I_{30}	I_{60}	I_{60}	I_{60}	I_{100}	I_{100}	I_{100}	I_{α}	$I_{2\alpha}$	$I_{3\alpha}$
		I_{apply}	I_{60}	I_{90}	I_{130}	I_{90}	I_{120}	I_{160}	I_{130}	I_{160}	I_{200}	$I_{2\alpha}$	$I_{4\alpha}$	$I_{6\alpha}$
Small	pr-(6,7)		-0.7	-0.7	-1.2	**-1.7**	**-1.7**	0.0	0.0	-0.2	0.0	-0.7	-0.7	-1.0
	pr-(6,9)		1.3	1.3	1.3	2.3	2.3	**0.8**	**0.8**	**0.8**	1.0	**0.8**	1.5	**0.8**
	pr-(6,11)		**-0.8**	-0.8	0.5	0.8	**-0.8**	-0.3	1.3	1.1	1.1	-0.5	-0.3	0.3
	pr-(6,12)		0.5	0.5	1.6	1.1	1.1	1.1	0.8	0.8	0.8	**0.3**	**0.3**	0.5
	pr-(8,7)		0.0	0.0	-0.4	-0.2	-0.2	0.7	-0.5	-0.5	-0.5	0.0	-0.5	**-0.7**
	pr-(8,9)		0.0	0.0	0.7	0.2	0.6	0.9	0.0	0.4	0.4	0.0	**-0.4**	**-0.4**
	pr-(10,7)		0.6	0.6	0.3	0.4	0.4	0.4	0.6	0.0	0.0	0.3	**-0.3**	0.1
	Average		**0.13**	**0.13**	0.40	0.41	0.24	0.51	0.43	0.34	0.40	0.03	**-0.06**	**-0.06**
Medium	pr-(6,20)		1.2	1.2	0.3	1.2	0.6	1.6	0.6	0.6	0.6	**0.0**	2.5	0.9
	pr-(8,11)		**-0.4**	**-0.4**	**-0.4**	0.2	-0.2	-0.2	0.2	0.2	0.2	0.6	0.0	0.6
	pr-(8,12)		0.2	0.2	0.0	-0.2	-0.2	-0.2	0.0	0.0	0.0	-0.2	**-0.6**	0.0
	pr-(8,20)		0.0	0.0	0.2	0.4	0.4	**-1.0**	0.6	0.6	0.6	-0.6	-0.2	0.8
	pr-(10,9)		0.6	0.6	0.9	0.4	0.0	1.1	**-0.1**	**-0.1**	**-0.1**	0.3	0.0	0.3
	pr-(10,11)		-0.3	-0.3	0.4	0.3	**-0.6**	0.1	0.0	0.0	0.0	-0.3	0.1	-0.3
	pr-(10,12)		**-0.4**	**-0.4**	0.3	0.3	0.3	0.1	0.7	0.7	**-0.4**	0.4	0.1	0.1
	Average		0.13	0.13	0.24	0.37	**0.04**	0.21	0.29	0.29	0.13	**0.03**	0.27	0.34
Large	pr-(6,40)		0.4	-0.4	-5.7	-2.6	-2.6	-5.7	**-6.6**	-6.2	-5.7	0.0	-1.3	-4.0
	pr-(6,60)		-0.7	-0.7	**-14.0**	-0.7	-0.7	-7.0	-3.5	-3.5	-3.5	-0.7	-2.2	-0.7
	pr-(8,40)		2.0	2.0	4.8	0.5	0.5	0.5	0.8	0.8	0.8	-0.8	**-1.3**	-0.5
	pr-(8,60)		**0.3**	**0.3**	8.3	2.6	2.6	6.9	5.8	5.8	5.8	2.0	4.9	1.3
	pr-(10,20)		**-0.2**	**-0.2**	0.9	0.3	0.3	1.7	2.0	1.2	1.9	0.6	1.2	0.2
	pr-(10,40)		1.1	1.1	4.4	1.4	1.1	0.6	3.1	3.1	0.7	1.1	1.1	**0.5**
	pr-(10,60)		**-1.3**	**-1.3**	2.8	-0.4	0.7	1.6	1.9	1.9	0.9	-0.2	-0.2	0.4
	Average		0.23	0.11	0.21	0.16	0.27	**-0.20**	0.50	0.44	0.13	0.29	0.31	**-0.40**
	Average:		0.16	0.12	0.29	0.24	0.15	0.18	0.4	0.35	0.21	0.1	0.18	**-0.03**
	Standard deviation:		0.14	0.14	0.64	0.51	0.39	0.40	0.75	0.68	0.49	0.16	0.18	0.15

Note : $\alpha = \sqrt{|I| \times |J| \times (\delta \times |D|)}$

Table 4.5 presents the results obtained from a subset of different groups for training features in our primary move, *Move 1*, with online learning. In this table, The "GAP - Best" columns show the improvements over the best solution obtained from the TS. We evaluated the L-TS-LR method with different subsets of features to identify how each feature contributes to the performance of the algorithm. For this purpose, we removed any noise or randomness in the algorithm (e.g., no diversification). Each column represents the set of features used. The first one displays the results from when all 10 features were considered. In the following columns, we present the most interesting results, using other combinations of features. Considering All features performs better than all other subsets, with a 0.05% gap on average for all instances. The results with "All" features are also more stable for all instances (standard deviation = 0.6). Columns Δ_x and $\Delta_x > 0$ show that considering features in the "Cost improvement" group perform the worst, with a 1.01% gap on average. However, we see more improvement with the "Solution" and "Move" related features in columns x, M^*, M', and Attractiveness and Trail of the moves ((A, T) for short where the gap is 0.34%, 0.46%, 0.48% and 0.36% respectively. We also evaluated the combination of "Solution" and "Cost improvement" groups in columns (x, Δ_x) and (M^*, Δ_x) where the gap was 0.44% and 0.41% on average. In general, we observe that the differences are more significant for the large size instances.

Table 4.5 Comparing the effect of different feature sets in learning - without noise

	Tests	GAP - Best (%)								
		All	Δ_x	$\Delta_x > 0$	x	M^*	M'	(x, Δ_x)	(M^*, Δ_x)	(A, T)
Small	pr-(6,7)	0.0	0.5	0.5	0.3	0.3	0.5	0.5	0.3	0.0
	pr-(6,9)	1.1	0.0	0.0	0.5	1.1	1.1	0.5	0.3	0.0
	pr-(6,11)	**-0.3**	**-0.3**	**-0.3**	0.0	0.0	0.0	0.0	0.0	0.0
	pr-(6,12)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	pr-(8,7)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	pr-(8,9)	0.0	1.0	1.0	0.0	0.4	0.4	0.4	0.4	0.2
	pr-(10,7)	0.1	0.3	0.3	0.3	0.3	0.3	0.1	0.1	0.1
Medium	pr-(6,20)	1.1	1.1	1.1	0.0	1.1	1.1	1.1	1.1	0.0
	pr-(8,11)	0.0	0.0	0.0	0.0	**-0.2**	**-0.2**	0.0	0.0	-0.2
	pr-(8,12)	0.4	0.0	0.0	**-0.2**	**-0.2**	**-0.2**	**-0.2**	**-0.2**	0.0
	pr-(8,20)	0.0	**-0.7**	**-0.7**	0.0	0.0	0.0	0.0	0.0	0.0
	pr-(10,9)	0.2	0.2	0.2	0.0	0.0	0.0	**-0.3**	**-0.3**	0.0
	pr-(10,11)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	pr-(10,12)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Large	pr-(6,40)	0.7	0.7	0.7	0.0	0.7	0.7	0.7	0.7	0.7
	pr-(6,60)	0.0	22.0	22.0	9.8	9.8	9.8	9.8	9.8	9.8
	pr-(8,40)	**-0.7**	**-1.7**	**-1.7**	**-1.0**	**-1.0**	**-1.0**	**-1.0**	**-1.0**	-1.0
	pr-(8,60)	**-0.7**	**-0.7**	**-0.7**	**-0.7**	**-0.7**	**-0.7**	**-0.7**	**-0.7**	-0.7
	pr-(10,20)	0.0	**-0.5**	**-0.5**	0.0	0.0	0.0	0.0	0.0	0.0
	pr-(10,40)	0.7	1.3	1.3	0.2	0.2	0.2	0.2	0.2	0.7
	pr-(10,60)	**-1.6**	**-1.9**	**-1.9**	**-1.9**	**-1.9**	**-1.9**	**-1.9**	**-1.9**	-1.9
	Average:	**0.05**	1.01	1.01	0.34	0.46	0.48	0.44	0.41	0.36
	Standard deviation:	**0.6**	4.9	4.9	2.2	2.2	2.2	2.2	2.2	2.2

The last set of experiments for testing the algorithm design is the setting: online vs offline. This answers the questions: can we learn from different instances? can we learn in advance? or is learning instance based? If pre-training is possible, then it would save training time whenever a problem is solved. It would mean that phases $T_{collect}$ and T_{train} could be done beforehand. We would only need to run T_{apply} when solving a problem. In addition to different sets of parameters, we also experimented with different learning settings, i.e. online and offline learning. Furthermore, we investigated different prediction outputs. This relates to the level of granularity of the prediction: the move that needs to be performed or the subset of neighbors to be evaluated.

We use different strategies to run the offline model. First, we determine the set of instances that can be used for learning. Each instance is based on a number of physicians and a number of patients. Each instance is run 10 times in our experiments. Can we use different instances of the same problem pr-(β, γ) to solve instance i of problem pr-(β, γ)-i, where β is the number of physicians and γ is the number of patients? Can we learn from similar instances i with different problem sizes (i.e., sharing the same number of physicians β or the same number of patients γ). Another option could be to use the instance i regardless of problem size. Recall that we are only using the learning phase for $Move1$ (swapping tasks for physicians). In the following, we investigate these three strategies and proceed as follows. To solve pr-(β, γ)-i:

- Strategy 1: collect and train on pr-(6,γ)-1. Here we learn from an instance that is similar in terms of the number of patients γ, but use a fixed number of physicians. This number is set to the smallest number over all problems, which is equal to 6. For example, we used the trained model from problem pr-(6,7) to predict the output for instances pr-(8,7) and pr-(10,7).
- Strategy 2: collect and train on pr-(β,7)-1. This is similar to the previous strategy, in that the learning uses a similar number of physicians, but uses a fixed number of patients (7 since it's the smallest). For example, we used the trained model from problem pr-(6,7) to predict the output for instances pr-(6,9) and pr-(6,11).
- Strategy 3: collect and train on pr-(β,γ)-1. For this strategy, we use one instance of a specific problem pr-(β, γ) to learn and predict for all other instances ($i > 1$). For example, we used the trained model from problem pr-(6,7)-1 to predict the output for instances pr-(6,7)-2 and pr-(6,7)-10.

The performance of these experiments on the L-TS-LR model is shown in Table 4.6. The "GAP - Best" and "GAP - Avg" columns present the improvements from the best solution obtained over the TS. We can observe from Table 4.6 that the strategies related to "Initial Sol." perform better than other strategies with an overall gap of 1.02%. It can also be observed that the strategies based on the "Physicians" were poorly designed; we can see the impact of randomness specifically in large problems where the gap is 2.61% on average. This shows that we need to use the right number of physicians and patients in order to predict the best moves, but we can learn information from solutions.

Finally, we also compared the performance of online versus offline learning based on computation time. Figure 4.10 compares the time per iteration for the L-TS-LR method in online and offline settings. Figure 4.10 also shows that the offline method performs faster than the online method (≈0.05 seconds per iteration). This is due to the fact that there is no time spent on the collect and train phases. This is in concordance with what was observed earlier: most of the time is spent for these two phases. The difference in the computing time (time per iteration) between the L-TS-LR offline model and the TS model is also negligible (≈0.02 seconds per iterations).

Table 4.6 Comparing the performance of logistic regression in the deterministic case with offline learning

	Tests	GAP - Best (%)			GAP - Avg (%)		
		Physicians	Patients	Initial Sol.	Physicians	Patients	Initial Sol.
Small	pr-(6,7)	0.0	0.1	**-0.2**	**-0.1**	**-0.2**	**-0.1**
	pr-(6,9)	1.5	1.9	1.6	**-0.9**	**-0.7**	**-1.1**
	pr-(6,11)	1.8	1.9	0.7	0.7	0.6	**-0.5**
	pr-(6,12)	2.1	2.4	1.2	1.0	1.4	0.2
	pr-(8,7)	0.9	1.0	0.6	0.9	0.8	0.5
	pr-(8,9)	0.7	1.1	0.4	0.6	0.7	0.2
	pr-(10,7)	0.1	0.4	0.5	0.2	0.2	0.3
Medium	pr-(6,20)	26.0	1.3	0.8	1.8	1.5	0.9
	pr-(8,11)	0.3	0.2	0.4	0.0	**-0.1**	0.2
	pr-(8,12)	**-0.2**	**-0.1**	**-0.4**	0.0	0.0	**-0.2**
	pr-(8,20)	**-0.8**	**-0.7**	0.4	**-1.1**	**-1.1**	**-0.3**
	pr-(10,9)	1.1	1.2	0.8	0.5	0.6	0.1
	pr-(10,11)	**-0.1**	**-0.2**	0.1	0.3	0.1	0.3
	pr-(10,12)	0.9	0.9	0.7	0.6	0.6	0.4
Large	pr-(6,40)	5.3	4.8	1.5	5.4	4.9	2.1
	pr-(6,60)	2.9	3.3	6.5	**-8.0**	**-8.7**	**-5.7**
	pr-(8,40)	**-0.3**	0.0	1.4	**-0.6**	1.0	0.2
	pr-(8,60)	3.6	3.8	4.4	6.7	5.7	3.4
	pr-(10,20)	1.3	1.2	0.7	0.9	0.9	0.3
	pr-(10,40)	2.0	2.2	1.3	1.6	1.7	0.5
	pr-(10,60)	0.7	0.7	0.1	1.0	0.9	0.8
	Average:	2.61	1.23	1.02	0.55	0.52	0.14
	standard deviation:	5.35	1.30	1.45	2.46	2.42	1.48

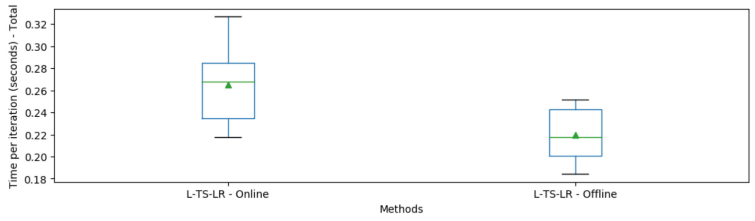


Figure 4.10 Comparing computing time per iteration for online vs. offline learning in the deterministic case

Our experiments show that the TS and L-TS perform similarly in terms of solution quality, but behave differently when it comes to running time, the number of evaluations and number of iterations. The impact is expected to grow as the size of the problem increases. In the following section, we present the stochastic version of the problem, where each scenario (arrival and profile of patients), needs to be evaluated.

Experimental performance on the Stochastic case

In this section we evaluate the L-TS method with uncertainty conditions. The main difference between the deterministic and the stochastic setting are the characteristics of the patients.

For each problem $pr - (\# \text{ of physicians}, \# \text{ of patients})$, the arrival days and the profile (palliative or curative) of the patients are known. In the stochastic version, the aim is to find the best weekly schedule for the physicians that will fit different patients arrivals and profiles. These uncertainty conditions make the search space larger. In a deterministic environment, where we have $|I|$ physicians, $|J|$ patients, and $\delta \times |D|$ time blocks in a physician scheduling problem, the complexity of the problem is $O(n^3)$, and the size of the neighborhood is $|I| \times |J| \times (\delta \times |D|)$. The stochastic conditions enlarge the problem by$\times$(#of scenarios). For this purpose, we follow the methods used by [1]. A set of 50 scenarios (with the stochastic setting) with different arrival days, etc. was generated. This is similar to what we referred to as an instance earlier. For each problem, the algorithm was computed using 10, 30 or 50 instances. These instances were defined based on the arrival day of patients (Poisson distribution) and their profiles.

For the stochastic case, the values of the parameters I_1^{stoc}, I_3^{stoc} and θ^{stoc} are the same as we used for the deterministic case, except that I_2^{stoc} is now equal to $|J|$, the number of patients, $I_{train}^{stoc} = \sqrt{|J| \times |I| \times (\delta \times |D|)}$, and $Stop_{max} = 5h$. We evaluated the same three criteria of ***Performance***, ***Efficiency*** and ***Scalability*** for the stochastic cases.

Performance Table 4.7 presents the results of experiments conducted in stochastic cases with online learning. Notably, both learning methods, L-TS-LR and L-TS-DT, demonstrated significant enhancements to the TS on a global scale. The most substantial improvements were observed in cases involving 10 scenarios, where L-TS-LR achieved an average improvement of -1.67%. Specifically, L-TS-LR enhanced the solution in 18 out of 21 instances and reached the same solution in 1 out of 21 instances. L-TS-DT also outperformed the TS, showing an average improvement of -3.02%, with enhancements in 17 out of 21 instances and identical solutions in 2 out of 21 instances. For the stochastic case, L-TS-DT exhibits more notable improvements although L-TS-LR method demonstrates greater overall robustness, with an average standard deviation of 5.52, compared to 9.28 for L-TS-DT.

We observe the same behavior for 30 and 50 scenarios : L-TS-LR method improves results for 16/21 (14/21) instances, and the average improvement is -0.54% (-0.59%); L-TS-DT method improves results for 18/21 (17/21) with an average improvement of -0.65% (-0.91%).

Table 4.7 Comparing the performance of different methods in the stochastic case

	Tests	GAP - Best (%)									GAP - Avg (%)								
		10 Scenarios			30 Scenarios			50 Scenarios			10 Scenarios			30 Scenarios			50 Scenarios		
		Random	L-TS-LR	L-TS-DT	Random	L-TS-LR	L-TS-DT	Random	L-TS-LR	L-TS-DT	Random	L-TS-LR	L-TS-DT	Random	L-TS-LR	L-TS-DT	Random	L-TS-LR	L-TS-DT
Small	pr-(6,7)	1.3	**-0.3**	**-0.3**	1.7	**-0.7**	**-0.5**	1.0	**-0.7**	**-0.2**	1.1	0.2	**-0.3**	0.8	**-1.4**	**-1.1**	0.9	**-1.1**	**-0.2**
	pr-(6,9)	0.8	0.0	0.0	1.3	0.0	**-0.3**	1.5	0.3	0.3	0.9	**-1.2**	0.0	1.5	**-1.1**	**-0.8**	0.5	**-2.2**	0.3
	pr-(6,11)	2.2	**-0.3**	**-0.6**	2.2	**-0.3**	**-0.3**	1.1	**-1.3**	**-1.1**	1.0	**-1.1**	**-0.6**	**-0.5**	**-1.1**	**-0.9**	2.6	**-2.6**	**-1.1**
	pr-(6,12)	0.8	**-1.4**	**-0.8**	0.8	**-0.6**	**-0.3**	0.8	**-1.6**	**-1.3**	0.3	**-1.3**	**-0.8**	1.2	**-1.1**	**-0.4**	1.6	1.3	**-1.3**
	pr-(8,7)	1.3	**-0.5**	**-0.4**	1.3	**-0.4**	**-0.4**	**-0.4**	**-1.1**	**-0.5**	0.4	**-1.0**	**-0.4**	1.5	**-0.6**	0.0	2.1	**-1.3**	**-0.5**
	pr-(8,9)	1.7	**-0.4**	**-0.4**	1.1	**-0.4**	-0.7	1.4	**-0.4**	**-0.4**	1.6	**-0.6**	**-0.4**	1.6	**-0.5**	0.1	0.7	**-1.4**	**-0.4**
	pr-(10,7)	1.0	**-0.7**	0.0	1.8	0.1	0.3	1.0	0.1	**-0.1**	1.3	**-0.4**	0.0	1.0	**-0.5**	**-0.1**	0.7	**-0.9**	**-0.1**
	Average:	1.30	**-0.51**	**-0.36**	1.46	**-0.33**	**-0.31**	0.91	**-0.67**	**-0.47**	0.94	**-0.77**	**-0.36**	1.01	**-0.90**	**-0.46**	1.30	**-1.17**	**-0.47**
	Standard deviation:	0.51	0.45	0.29	0.47	0.29	0.31	0.63	0.71	0.56	0.46	0.54	0.29	0.73	0.36	0.48	0.81	1.25	0.56
Medium	pr-(6,20)	0.7	**-1.3**	**-2.0**	0.6	**-1.9**	**-0.6**	2.7	0.6	**-0.3**	1.6	**-1.4**	**-2.0**	1.5	**-4.4**	**-4.0**	0.5	0.6	**-0.3**
	pr-(8,11)	1.0	**-0.6**	**-1.0**	1.2	**-0.4**	**-0.4**	0.9	**-0.6**	0.0	1.3	**-1.3**	**-1.0**	0.5	**-1.1**	**-0.5**	0.3	**-1.6**	0.0
	pr-(8,12)	1.2	**-0.8**	**-0.2**	1.8	**-0.6**	**-1.0**	0.0	**-1.5**	**-2.1**	0.6	**-0.8**	**-0.2**	0.5	**-1.9**	**-0.7**	0.0	**-2.0**	**-2.1**
	pr-(8,20)	2.4	**-0.7**	0.4	0.8	**-1.3**	**-1.0**	1.8	**-0.4**	**-1.0**	**-1.3**	**-1.4**	0.4	0.8	**-3.1**	**-1.1**	1.7	**-2.1**	**-1.0**
	pr-(10,9)	1.5	**-0.1**	**-0.3**	1.7	0.1	0.3	1.0	**-0.4**	**-0.4**	1.5	**-0.4**	**-0.3**	0.9	**-0.8**	**-0.6**	0.3	**-1.6**	**-0.4**
	pr-(10,11)	0.4	**-1.0**	**-0.4**	0.7	**-0.7**	**-0.3**	0.9	0.3	0.3	1.5	**-1.0**	**-0.4**	0.0	**-0.7**	**-0.1**	**-0.4**	**-1.9**	0.3
	pr-(10,12)	1.3	**-0.3**	**-0.3**	1.8	**-0.4**	**-0.4**	0.4	**-0.4**	**-0.4**	1.2	**-0.3**	**-0.3**	0.8	**-1.2**	0.5	0.1	**-2.2**	**-0.4**
	Average:	1.21	**-0.69**	**-0.54**	1.23	**-0.74**	**-0.49**	1.10	**-0.34**	**-0.56**	0.91	**-0.94**	**-0.54**	0.71	**-1.89**	**-0.93**	0.36	**-1.54**	**-0.56**
	Standard deviation:	0.64	0.41	0.76	0.54	0.66	0.45	0.90	0.67	0.79	1.03	0.46	0.76	0.46	1.38	1.45	0.66	0.97	0.79
Large	pr-(6,40)	5.2	4.6	1.1	4.3	**-1.4**	**-1.0**	**-1.3**	**-3.5**	**-3.1**	2.2	1.9	1.1	0.3	**-1.8**	**-0.3**	4.3	**-0.8**	**-3.1**
	pr-(6,60)	**-10.7**	**-25.0**	**-42.9**	8.1	0.0	0.0	3.8	**-1.5**	**-3.8**	16.3	**-44.7**	**-42.9**	12.4	0.3	**-76.5**	0.1	**-3.2**	**-3.8**
	pr-(8,40)	1.3	**-0.6**	**-6.6**	1.6	**-0.5**	**-1.4**	0.8	0.0	**-1.8**	2.8	**-0.4**	**-6.6**	**-0.3**	**-0.9**	**-0.9**	0.5	-2.4	**-1.8**
	pr-(8,60)	0.0	**-3.2**	**-3.7**	0.5	**-1.4**	**-4.5**	1.7	**-0.3**	**-2.0**	3.3	**-4.8**	**-3.7**	3.1	1.6	**-27.7**	0.1	**-3.5**	**-2.0**
	pr-(10,20)	1.1	**-0.7**	**-0.8**	1.9	**-0.2**	**-0.3**	1.5	0.0	**-0.5**	1.2	**-1.0**	**-0.8**	0.2	**-2.3**	**-1.9**	0.2	**-4.6**	**-0.5**
	pr-(10,40)	0.2	**-2.1**	**-2.8**	1.5	0.0	**-0.6**	0.9	**-0.4**	**-0.5**	0.4	**-1.0**	**-2.8**	0.3	**-0.3**	0.0	0.1	**-1.5**	**-0.5**
	pr-(10,60)	2.0	0.3	**-1.7**	2.6	**-0.5**	**-0.3**	0.9	0.4	0.0	1.5	4.3	**-1.7**	0.6	**-1.1**	**-2.7**	2.8	**-1.9**	0.0
	Average:	**-0.13**	**-3.81**	**-8.20**	2.93	**-0.57**	**-1.16**	1.19	**-0.76**	**-1.67**	3.96	**-6.53**	**-8.20**	2.37	**-0.64**	**-15.71**	1.16	**-2.56**	**-1.67**
	Standard deviation:	4.97	9.66	15.49	2.57	0.60	1.55	1.51	1.35	1.43	5.53	17.06	15.49	4.56	1.32	28.59	1.70	1.30	1.43
	Average:	0.79	**-1.67**	**-3.02**	1.87	**-0.54**	**-0.65**	1.07	**-0.59**	**-0.91**	1.93	**-2.75**	**-3.02**	1.36	**-1.14**	**-5.7**	0.94	**-1.75**	**-0.91**
	Standard deviation:	2.84	5.52	9.28	1.65	0.54	0.97	1.03	0.93	1.10	3.42	9.75	9.28	2.65	1.20	17.28	1.17	1.27	1.10

Efficiency To evaluate the efficiency, we provide the primal integral, the beneficial rate and the convergence rate.

Primal integral value. We show the primal integral values in the stochastic environment with 30 scenarios in Figure 4.11.

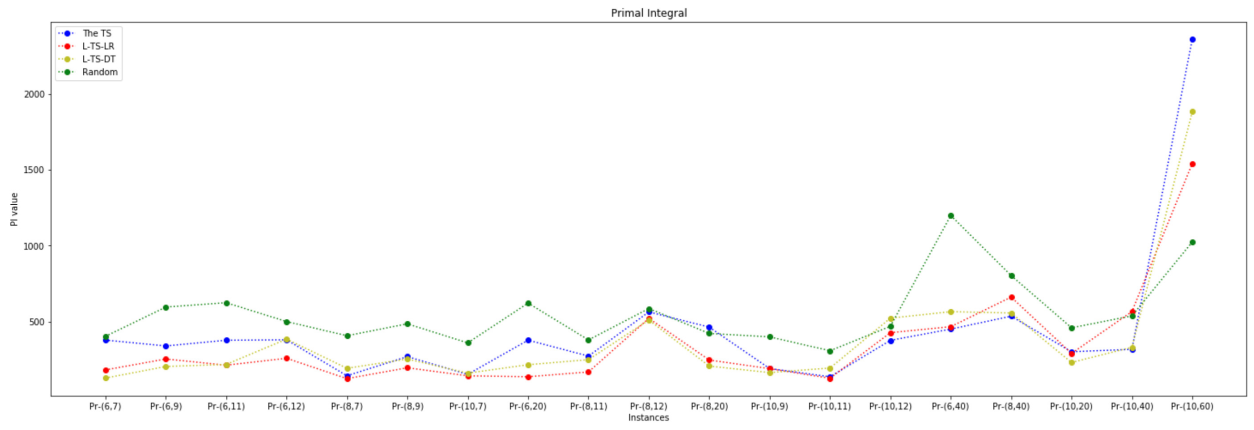
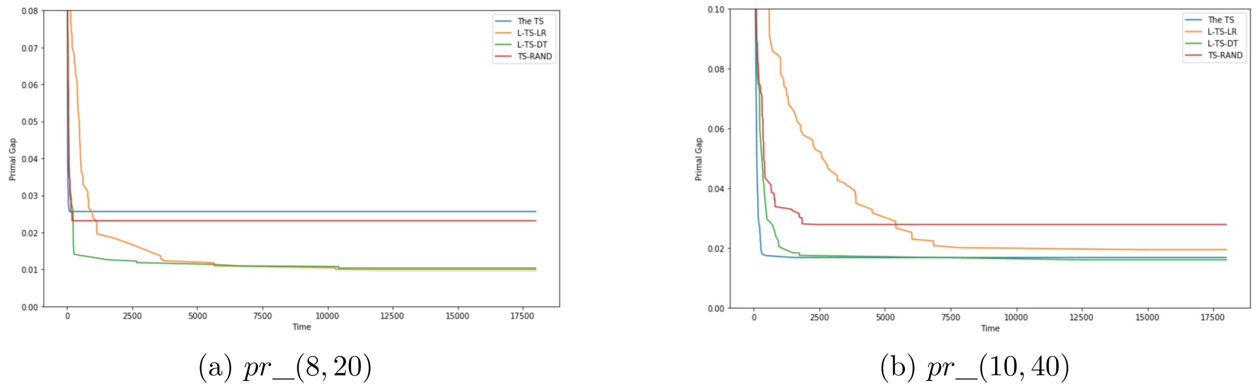


Figure 4.11 Comparing the primal integral value for different methods in the stochastic case with 30 scenarios

Figure 4.11 shows that the learning methods have better or comparable primal integral values for almost all instances. This figure shows that both learning methods, presented by red and yellow lines, perform better than the TS and Random algorithms.

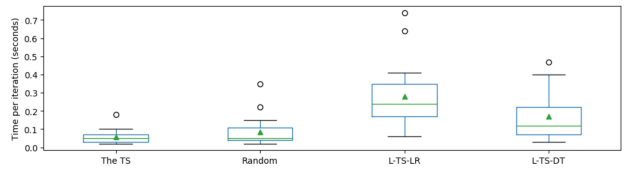


(a) $pr\_(8, 20)$ (b) $pr\_(10, 40)$

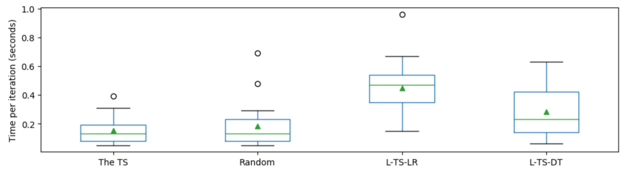
Figure 4.12 Comparing the primal gap for all methods in the stochastic case with 30 scenarios

The primal gap values over time are also presented in Figure 4.12 for two specific instances: $pr-(8, 20)$ and $pr-(10, 40)$ with 30 scenarios. In the case of $pr-(10, 40)$, Figure 4.11 indicates that the primal value for the L-TS-LR method is less favorable than that of the TS method. However, Figure 4.12b shows that both L-TS-LR and the TS methods achieved similar solution and the difference is where L-TS-LR requires more time to reach to the similar solution. Figure 4.12a clearly shows the advantage of both learning methods over the TS in $pr-(8,20)$.

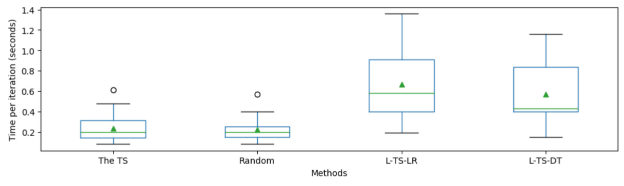
The performance of each learning algorithm in a stochastic environment was also evaluated based on its computation time. Figure 4.13 compares the time per iteration by method for 10, 30 and 50 scenarios. It is clear that increasing the number of scenarios increases the computation time.



(a) 10 Scenarios



(b) 30 Scenarios



(c) 50 Scenarios

Figure 4.13 Comparing computation time per iteration for all methods, in the stochastic case, with online learning

Beneficial rate. The second criteria we use to compare the efficiency of the methods is the beneficial rate B. This rate is demonstrated in Figure 4.14 for L-TS-LR and L-TS-DT methods in the stochastic case with 30 scenarios.

B(L-TS-LR) shows advantage of L-TS-LR method over TS in 16/21 instances while B(L-TS-DT) shows this advantage in 17/21 instances. For most instances, the rate B is close to 0, with a median of 0.0 for the stochastic environment and 30 scenarios. This confirms that the performance of L-TS is very close to that one of the TS in the stochastic cases as well.

Convergence rate. We compared the number of evaluations and number of iterations

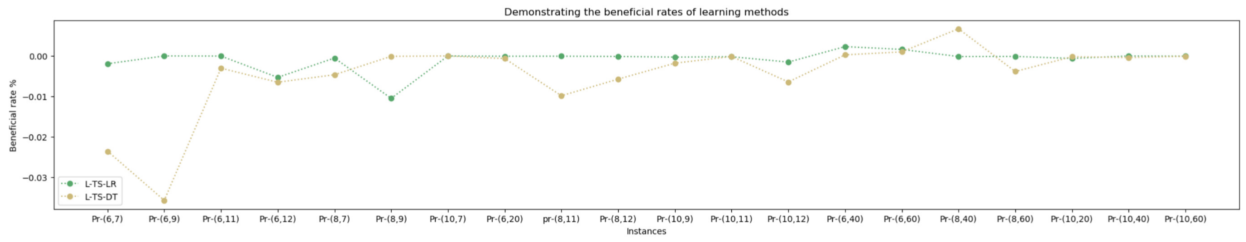


Figure 4.14 Comparing the beneficial rate for learning methods in the deterministic case

(convergence rate) as the last criteria to compare the different methods in terms of efficiency. Figure 4.15 provides a comparison of these two critical metrics to reach the predetermined stopping criterion ($Stop_{max}$). The metric "number of evaluations" represents the count of neighboring data points to be examined in each iteration. It is evident that the graph clearly indicates a decrease in the number of evaluations within the learning methods, which signifies the evaluation of a smaller subset of neighbors. Additionally, the reduction in the total number of iterations demonstrates our effective reduction of the overall search space in both learning methods.

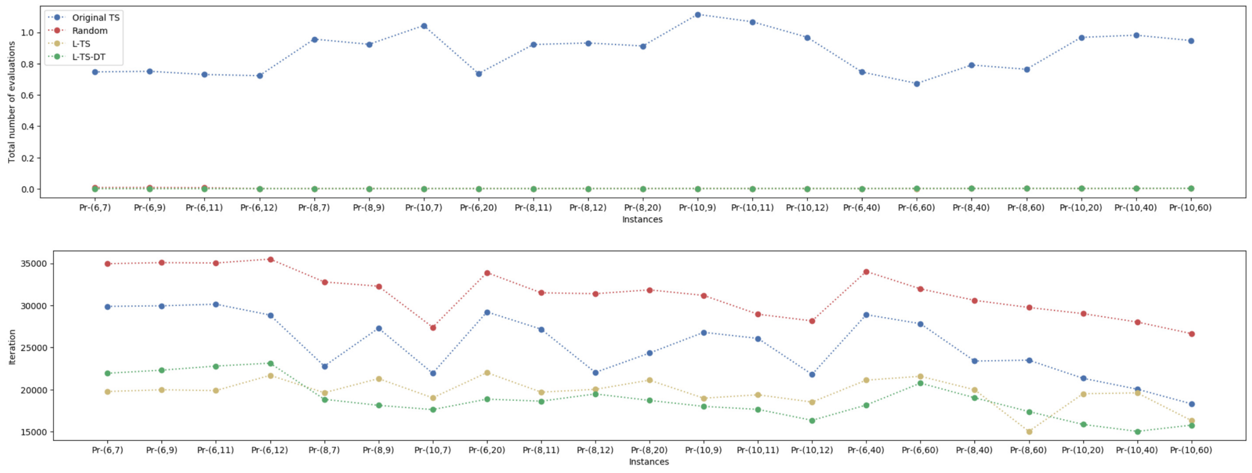


Figure 4.15 Comparing computing performance for all methods in the stochastic case with 30 scenarios

These illustrations, as well as the earlier-mentioned enhancement in the objective value presented in Table 4.7, collectively demonstrate the capacity of L-TS methods to learn and attain high-quality solutions during the search process. This ability is particularly beneficial for real-world problem instances, as it allows for improved solutions while generally demanding fewer iterations and evaluations. It underlines the algorithm's effectiveness in exploring a reduced search space while incorporating valuable features in the search mechanism.

4.5 Conclusion

In this paper, we proposed a novel learning tabu search method, and assessed its performance in the context of the physician scheduling problem. We evaluated the performance of our approach on several benchmark instances with promising results. Notably, our L-TS method leverages data generated during the tabu search procedure, and none of the 10 features used for learning are application-dependent. Based on this, we believe that L-TS has broad applicability for any optimization problem that has been solved using tabu search.

Our experimental design and evaluation of the new method encompassed both deterministic and stochastic environments. To illustrate the use of the algorithm, for parameter tuning and as a proof of concept, we used the deterministic case as a benchmark and compared the performance of the L-TS methods with the TS algorithm. We evaluated the L-TS methods in terms of Performance, Efficiency, Scalability, and Sensitivity.

Regarding “Performance”, the L-TS-LR algorithm showed a slight improvement over the TS algorithm, while L-TS-DT achieved very similar results with only a 0.31% gap over the TS. We assessed “Efficiency” by considering three rates, the “Primal Integral”, the “Beneficial Rate” and the “Convergence Rate”, and observed the effectiveness of both L-TS methods. To evaluate “Scalability”, we varied the size of the problems. Finally, we assessed the Sensitivity” by varying the design of our experiments. In Section 4.4.2 (*Fine-Tuning and Feature Selection*), we presented different settings for our L-TS methods, evaluated different sizes for I_{train} and I_{apply}, and assessed the effect of different feature sets on the algorithm’s performance. We also tested the algorithm with different learning strategies, online vs offline. Through these experiments, we validated the effectiveness of our L-TS algorithm and identified the best parameters for obtaining optimal results. We illustrated that both L-TS methods achieved very comparable results with the TS method in the deterministic cases. We extended these algorithms to the stochastic cases where both L-TS-LR and L-TS-DT methods showed great advantage over the TS.

In conclusion, we demonstrated that our method is highly efficient when compared to both the tabu search algorithm and a random method, particularly in its stochastic version. Across 21 instances, the average gap was improved by 1.67% with logistic regression and 3% with decision trees when 10 scenarios were considered. Additionally, the learning methods achieved better solutions faster than both the tabu search algorithm and random methods with respect to “number of evaluations” and “number of iterations” which confirms the reduction in the search space.

Although we focused on applying our method to a scheduling problem, tabu search has proven effective in solving a wide range of optimization problems. Therefore, the learning tabu search algorithm has the potential to be adapted for use in other applications. Future work could further extend the algorithm by generalizing several ingredients that we developed specifically for our application, so that they would work well on a more broad set of optimization problems.

CHAPTER 5 ARTICLE 2. BLACKBOX OPTIMIZATION OF TABU SEARCH HYPERPARAMETERS

Nazgol Niroumandrad
Department of Mathematics and Industrial Engineering, Polytechnique Montréal
Nadia Lahrichi
Department of Mathematics and Industrial Engineering, Polytechnique Montréal
Sébastien Le Digabel
Department of Mathematics and Industrial Engineering, Polytechnique Montréal

Abstract The performance and behavior of any metaheuristic algorithm largely depend on its hyperparameters. These hyperparameters need to be chosen carefully to achieve the best results. Their impact is even more important in large-scale and real-world size problems. The literature shows that hyperparameter optimization (HPO) is a nontrivial task and efficient methods are required to obtain the best possible results. We study the impact and performance of different HPO approaches and present how blackbox optimization (BBO) enables the efficient selection of tabu search (TS) hyperparameters. Computational experiments have been conducted on two real-world applications.

Keywords Blackbox optimization (BBO); hyperparameter optimization (HPO); tabu search (TS).

History This article is submitted for publication in *Journal of Heuristics*

5.1 Introduction

Metaheuristics are widely recognized as efficient approaches for many difficult combinatorial problems. All metaheuristic algorithms are associated with a set of hyperparameters that have a great impact on their performance. Hyperparameters are parameters whose values are predetermined before the start of an algorithm and remain fixed throughout its execution. While default hyperparameter settings are typically provided as part of the algorithm's design, these settings are often optimized for specific problems or application domains. Traditionally, the tuning of optimization algorithms have been handled manually. However, the best values for the hyperparameters cannot be identified through a trial-and-error approach when the search space is large. Literature shows that hyperparameter optimization is a non-trivial task.

The task of fine-tuning algorithms often takes longer than the design and implementation of the algorithms themselves. This is particularly true where the "right" choice of values for hyperparameters has a considerable effect on the performance of the algorithm. The performance of a heuristic or metaheuristic algorithm is typically evaluated based on both the quality of the solutions obtained and the time needed to find them, yet tuning time is often overlooked.

In this work, we study different approaches for optimizing tabu search (TS) hyperparameters. This problem is called a hyperparameter optimization (HPO) problem and is commonly treated as a blackbox optimization (BBO) problem [20], formulated as

$$\max_{p \in \mathbb{P}} \quad \phi(p) \tag{5.1}$$

where $\mathbb{P}$ is the set of hyperparameters, and ϕ is the objective function, provided by a *blackbox*, i.e. a process, typically a computer simulation, for which no analytical expression is available. BBO methods follow the framework described in Figure 5.1, where the blackbox is iteratively evaluated by the solver.

Before defining the BBO problem, several key questions must be addressed. The first question pertains to the objective measure of the problem. Different objectives can be considered, such as minimizing the time required to solve a set of problem instances or ensuring that all instances are solved within a reasonable time. The second question concerns the set of acceptable hyperparameter values ($\mathbb{P}$). For instance, some hyperparameters may have constraints, such as non-negativity or upper limits. Another key question is how to select a subset of instances for evaluation.

In this study, we investigate the use and the performance of the following methods for HPO:

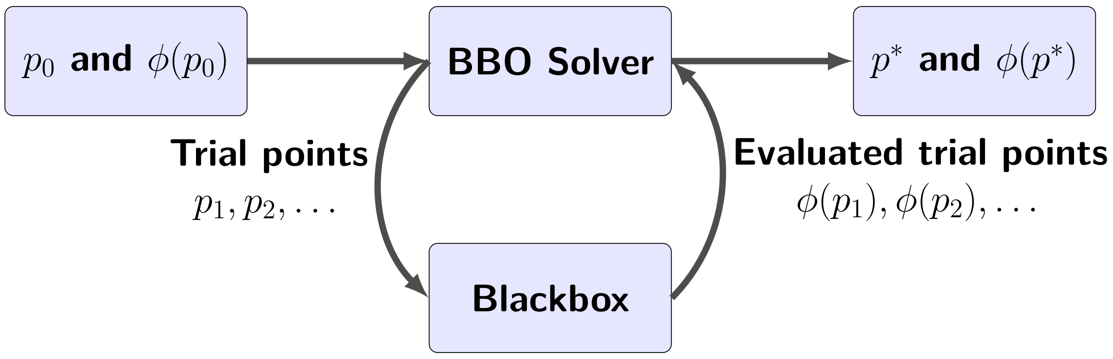


Figure 5.1 An illustration of the iterative process in a BBO algorithm, where trial points are generated by the solver and evaluated by the blackbox to converge to a solution.

MADS [113], IRACE [24], and Grid Search, across a diverse set of optimization problems. We consider the Multi-Resource Generalized Assignment Problem [25], which focuses on efficiently allocating resources to tasks, and the Physician Scheduling Problem [1], which entails optimizing the schedules of physicians while considering availability and workload constraints. Note that both problems are multi-objective. In both cases, a TS algorithm was used to solve the problems. We refer to these as the target algorithms in the remaining. Our experiments show that IRACE and Grid-Search (Grid-Heuristic) perform similarly, while MADS outperforms both by achieving faster convergence rates and superior overall solution quality.

The remainder of this study is organized as follows: Section 5.2 provides a review of related studies. In Section 5.3, we provide a detailed description about the problem following by describing the applications and instance generation. Section 5.4 explains the HPO methods in detail while Section 5.5 presents the results. Section 5.6 discusses the findings, and Section 5.7 provides concluding remarks.

5.2 Literature review

Hyperparameter optimization (also termed as offline tuning) and hyperparameter control (also known as on-line tuning) are two major forms of setting parameter values. Hyperparameter optimization is usually referred to as the approach of finding good values for the hyperparameters before the execution of the target algorithm. These hyperparameters remain fixed during the execution. In contrast, a hyperparameter control method starts with initial hyperparameter values that are adjusted during execution.

Each metaheuristic has a predefined set of hyperparameters that has to be initialized before execution or dynamically adjusted during the execution of the algorithm. The calibration of these hyperparameters is usually done with respect to the problem.

Many studies have addressed the problem of tuning the hyperparameters of metaheuristics, such as beam search [62], experimental design [11, 63], genetic programming [64], the application of racing algorithms [12, 65], and combinations of fractional experimental design and local search [10].

These methods can be classified as blackbox or whitebox tuning methods [67]. Blackbox tuning methods refer to those that treat metaheuristics as a “blackbox”. They are usually presented in the form of automated tools that systematically search for the best hyperparameter values or combination of metaheuristic components. CALIBRA [10] and F-RACE [65] are placed in this category. The whitebox tuning methods refer to those that allow the designer to inspect the inner-working of the algorithm and to assist in designing a better algorithm. These include, statistical analysis such as fitness distance correlation and run time distribution analysis [68–70], human-guided search such as studies performed in [71, 72]. Recently, this line of work resurfaced in [73]. Also visualization of search algorithm behavior [74] can be categorized in the whitebox methods. Whitebox optimization relies on the explicit knowledge of the objective function and constraints, making it effective for structured problems like convex or differentiable optimization. However, when the objective function is complex, non-differentiable, or inaccessible (i.e., proprietary software, simulations, or heuristic algorithms), blackbox optimization becomes essential [20]. These methods still require human effort for tuning, and results may vary as different users apply different strategies. While whitebox approaches use explicit formulas to guide optimization, blackbox methods rely on iterative and experimental evaluations, emphasizing the need for robust and tailored hyperparameter tuning techniques.

There are few other works around the HPO problem, such as, an agent based approach +CARPS [75] and self adaptive algorithms [28]. Recent advancements in HPO include model-based approaches such as Bayesian Optimization (BO), which efficiently explores the search space using surrogate models [66]. While BO has shown success in tuning continuous hyperparameters, it can be less effective for highly categorical and combinatorial spaces, such as those encountered in TS. Hybrid approaches, such as BO combined with local search or multi-fidelity HPO, aim to mitigate these limitations [114].

The literature also highlights the critical role of HPO in improving solver efficiency and effectiveness. For example, [77] discusses the importance of selecting the appropriate parameter configuration in order to achieve optimal performance. The authors emphasize that the choice of parameters can have a significant impact on the solver’s performance, and that different parameter settings may be optimal for different problem instances. The article presents a comprehensive overview of existing techniques for parameter tuning, including Grid Search, Random Search, and BO, and discusses their relative strengths and weaknesses. The authors also highlight several key challenges associated with parameter tuning, including the time and computational resources required to perform the tuning, the difficulty of choosing an appropriate performance metric, and the potential for over-fitting. In the recent study [78], the authors provided an inclusive survey on offline automatic algorithm configuration which is categorized in four main groups; Model-Free methods, Racing methods, Design of Experiments (DoE) methods, and Model-based methods.

In this work, we study the optimization of hyperparameters for the TS metaheuristic which emphasizes the importance of understanding how hyperparameter choices influence the behavior and outcomes of TS, in line with "scientific testing" proposed by [115] that investigates how specific factors affect algorithmic performance. Over the years, different approaches have been proposed to configure the TS hyperparameters. One common approach is to use metaheuristic techniques such as evolutionary algorithms, simulated annealing, or particle swarm optimization. These studies include [79–82].

In an early study, [52] proposed an adaptive TS algorithm that uses a set of rules to adjust the tabu list size, neighborhood size, and other parameters based on the characteristics of the optimization problem. The algorithm was evaluated on several benchmark problems and was shown to perform well compared to other state-of-the-art algorithms.

Each of fine tuning methods can be effectively leveraged to tune the hyperparameters of TS. Each offers unique strengths in exploring parameter spaces, optimizing performance, and reducing computational time. However, given the structured nature of TS hyperparameters, approaches like those used in this study (MADS and IRACE) remain competitive, offering greater flexibility in handling mixed-variable spaces. In the following sections, we outline how MADS, IRACE, and Grid Search can be applied as blackbox optimization methods to efficiently fine-tune the TS hyperparameters, providing a comparative perspective on their potential impacts and advantages. While IRACE, Grid Search and MADS [83,84] have been used in the literature to optimize hyperparameters for various algorithms and metaheuristics, to the best of our knowledge, this is the first time MADS is being applied to optimize the hyperparameters of the TS algorithm and compared against the other two methods. MADS is most effective for continuous optimization problems where local search and refinement of solutions are required, particularly in constrained environments. IRACE, with its global exploration and adaptive sampling, excels in automatic tuning of algorithms with complex, mixed-parameter spaces, offering a balance between efficiency and flexibility. Grid Search, while reliable and systematic, is better suited for small-scale parameter searches where exhaustive evaluation is manageable. For large or complex problems, IRACE and MADS provide the most robust and adaptable solutions.

5.3 Hyperparameter optimization for tabu search

In this section, we start with a short description of the fundamentals of the TS algorithm, focusing on its mechanism for escaping local optima using a flexible memory structure. We then describe the HPO problem we aim to address and introduce two case studies used for the experimentation and comparative analysis.

5.3.1 Tabu search principles

TS [2] is an advanced local search method designed to overcome the limitations of traditional local search methods. These methods explore potential solutions of a problem and their immediate neighbors to identify improvements. However, they may become trapped in local optima. To mitigate this, TS employs a tabu list, a memory structure that prevents revisiting previously explored solutions, thereby reducing cycling and enhancing search efficiency.

In summary, TS starts with an initial solution x_0, which may be infeasible. It then iteratively moves from the current solution x to a neighboring solution x' in search of an improved solution. The neighborhood $N(x)$ is defined as all solutions that can be reached from x after applying a specific move. The next solution x' is chosen as the best solution in $N(x)$ that does not exist in the tabu list. An exception can be made based on aspiration criteria, allowing a tabu move if it improves the current best solution x^*. The function f is used to evaluate each solution.

To prevent cycling, a tabu list (T_{list}) is maintained, containing attributes of recently visited solutions or moves that cannot be revisited for a specific number of iterations.

Various strategies can be applied to search the neighborhood solutions, and the moves and strategies used to explore the solution space are crucial factors affecting the quality of solutions in TS. Over time, various advancements have expanded the capabilities of TS. These include Adaptive TS (ATS) [26], probabilistic TS [27], Reactive Tabu Search (RTS) [28], and multiple neighborhood structures. The latter share similarities with the Variable Neighborhood Search (VNS) [116], a powerful method to systematically change neighborhood structures in order to explore the solution space more effectively. Unlike TS, which uses memory structures to avoid revisiting previously explored areas, VNS dynamically switches between different neighborhood structures to escape local optima and improve the search process.

The key hyperparameters for TS include the size of the tabu list, the stopping criteria, the size of various neighborhoods, and the selection of different neighborhood structures.

5.3.2 The hyperparameter optimization problem

We now formulate the HPO problem for TS, the target algorithm, as a BBO problem. For that, we need to define ϕ, the objective function in (5.1), and $\mathbb{P}$, the set of hyperparameters of TS, such as the size of the tabu list or the neighborhood size n. The blackbox process that provides the value of ϕ executes TS on a series of runs in the set S, with fixed values $p \in \mathbb{P}$ for the hyperparameters, and we note $f_s^{TS}(p)$ the value of each of these TS runs, with $s \in S$. We propose the two following variants for $\phi(p)$:

- The maximum of f_s^{TS}, for searching the overall best solution:

$$\phi_{\max}(p) = \max_{s \in S} \quad f_s^{TS}(p) \tag{5.2}$$

- The average of f_s^{TS}, for searching the most robust set of hyperparameters:

$$\phi_{\text{avg}}(p) = \text{avg}_{s \in S} \quad f_s^{TS}(p) \tag{5.3}$$

A BBO solver can then be applied to optimize (5.2) or (5.3) over $\mathbb{P}$, the set of hyperparameters, that is specific to each case study, as shown in the next section.

5.3.3 Case studies

In this section, we present the key characteristics of two case studies, including a brief description of the problem, the proposed TS algorithm, and the associated TS hyperparameters defining the set $\mathbb{P}$.

Multi-Resource Generalized Assignment Problem

The assignment problem involves assigning a set of tasks to a set of resources or agents in a way that minimizes the total cost or maximizes the total efficiency of the assignments. The assignment problem has numerous applications in various fields, including logistics, transportation, workforce management, project scheduling, and resource allocation. Each task needs to be assigned to exactly one resource, and each resource can handle only one task. In the generalized case, each resource can handle multiple tasks, subject to a capacity constraint (or multiple ones in the multi-resource case).

Given T tasks and R resources, and a cost matrix C where c_{ij} represents the cost of assigning task $i \in T$ to resource $j \in R$, the objective is to find an assignment that minimizes the total assignment cost.

The authors in [25] present a novel algorithm for assigning patients to nurses in home care services, aimed at balancing nurse workloads and minimizing travel distances. The fluctuating demand for homecare services can create significant workload imbalances among nurses. The authors introduced a mixed integer programming model, that considers three main components of a nurse's workload: visit load (number and complexity of visits), case load (number of patients, categorized into five groups based on care complexity), and travel load (distance traveled for patient visits). However, due to the high computational complexity of exact methods for large combinatorial problems, and given that the objective function and a set of constraints are nonlinear, a TS algorithm is designed to solve the optimization problem. It uses the following five moves:

Flip: changing the assignment of patient m from nurse i to nurse i';

2-swap: exchanging patients m and m' between nurses i and i';

3-swap: exchanging patients m, m' and m'' initially assigned to i, i' and i'' respectively, such that m is reassigned to i', m' to i'', and m'' to i;

2-mswap: exchanging patient m assigned to i with a set of patients m' assigned to i'. Workload of m and m' is comparable;

3-mswap: exchanging patient m, patients in set m' and m'' initially assigned to i, i' and i'' respectively, such that m is reassigned to i', set of patient m' to i'', and m'' to i.

A tabu list prevents cycling by temporarily forbidding the reassignment of a patient (and comparable ones, i.e. similar category and workload) back to their previous nurse unless it improves the overall solution.

In addition to using distinct move types, intensification strategies focus the search on promising areas by prioritizing patients with the highest impact on workload imbalance, while diversification strategies explore under-explored regions of the solution space when progress stagnates. A total of nine strategies are used to increase or reduce the size of the neighborhood.

The key hyperparameters that need to be optimized in this setting include: the size of the tabu list, the sequence of moves and neighborhoods, and the effort (in terms of number of iterations) spent in each neighborhood.

Physician Scheduling

The second case study is the physician scheduling problem [1]. Its objective is to generate a weekly cyclic schedule for physicians in a radiotherapy department, while also assigning arriving patients to the most suitable available specialist based on their cancer type. Number and type of patients arriving each day is unknown.

The physician scheduling problem in radiotherapy centers is generally task-based [109, 110]. Each day is divided into one or multiple periods, and each period is dedicated to a specific task. All tasks have to be scheduled weekly, and some may be repeated. The objective is to minimize the pre-treatment phase duration for patients, which is defined as the time from the patient's arrival at the center until the final task is completed and treatment begins. Typically, this phase lasts one week for curative patients and at most three days for palliative patients. Physician preferences may also be considered during scheduling.

For a set of physicians I, set of patients M, set of tasks T and periods D, the physician scheduling problem can be mathematically formulated by defining binary variables x_{ti}^{d} to represent whether physician $i \in I$ performs task $t \in T$ on period $d \in D$, and y_{im} to denote if patient $m \in M$ is assigned to physician i. Another variable, z_{imt}^{d}, is introduced to determine if patient m undergoes task t on period d when assigned to physician i.

This study aims to achieve two objectives: maximizing physician preferences and minimizing the pre-treatment duration. A mathematical model for this problem was proposed in [1], along with a tabu search metaheuristic.

The TS algorithm uses three types of moves related to the decision variables presented above. These moves are:

Swap: for each physician i, exchanging tasks t and t' scheduled on periods d and d';

Change: for each physician i, replace one of the repeated tasks. Indeed, in general, there are more periods than tasks to be performed;

Assignment: for each patient $m \in M$, change the assignment from the physician initially assigned to another one.

The key hyperparameters that need to be optimized is similar to previously: the size of the tabu list, the sequence of moves, and the effort (in terms of number of iterations) spent in each neighborhood.

In the following, we explain the characteristics of the three methods for optimizing the TS hyperparameters.

5.4 Methodology

In this section, we present the three considered methods for HPO: MADS, IRACE, and two variants of Grid Search. MADS and IRACE are employed to our context and the Grid Search variants are considered as simple comparative baselines.

5.4.1 Mesh Adaptive Direct Search (MADS)

The Mesh Adaptive Direct Search (MADS) algorithm was introduced by Audet and Dennis in 2006 [113]. MADS is a BBO method that follows the framework of Figure 5.1, and, more specifically, it is a direct search method that generates sets of points at each iteration, using dense sets of directions around the current iterate. These points lie on a discretization of the space called the mesh, and are sent to the blackbox to be evaluated. The algorithm iteratively refines its search until a predefined stopping criterion is met, such as reaching a maximum number of evaluations or a minimal value for the mesh size.

We treat the TS evaluation function as the blackbox and use the trial points in $\mathbb{P}$, generated by MADS to optimize the TS hyperparameters, such as the size of the tabu list or the number of iterations to search within a neighborhood.

5.4.2 IRACE

The IRACE package [24] is designed for automatic algorithm configuration, particularly in complex optimization scenarios where multiple parameters influence the performance of algorithms. The package implements an iterated racing algorithm, which evaluates different configurations of algorithm parameters by progressively testing them on a set of problem instances. This iterative process continues until a predefined stopping criterion is met, such as exhausting the computational budget. Poor-performing configurations are eliminated using statistical tests, making IRACE an efficient tool for parameter tuning.

One of IRACE's key features is its "racing procedure", where configurations are tested and progressively discarded if they perform significantly worse than others based on statistical evaluations across instances. This racing mechanism ensures that only the most promising configurations proceed to the next rounds of testing. To further improve performance and prevent premature convergence on suboptimal configurations, IRACE incorporates a "soft-restart mechanism", which reinitializes parts of the search space if the search converges too early on similar configurations. Another key feature is "elitist racing", where the best configurations (those evaluated on the largest number of instances) are given preference in subsequent evaluations, ensuring that they are tested extensively across multiple problem instances to avoid losing potentially good configurations due to variance in early rounds.

The IRACE package is highly flexible, capable of handling categorical, ordinal, and numerical parameters, and supports conditional parameters, which depend on the values of other parameters. The package also updates its sampling distributions iteratively, using truncated normal distributions for numerical parameters and discrete distributions for categorical ones, which allows for fine-tuned exploration of the parameter space. Furthermore, IRACE supports parallel execution across multiple cores or computers, making it scalable for large computational tasks. These features make IRACE particularly suitable for automatic tuning in scenarios involving complex algorithms with numerous parameters, ensuring robust performance across a wide variety of problem instances.

5.4.3 Grid search

In addition to MADS and IRACE, we consider two simple alternatives based on Grid Search, called "Random Grid Search" (Grid-Rand), and "Heuristic Grid Search" (Grid-Heuristic). The principle of Grid Search for HPO is to discretize the space of hyperparameters. This "grid" defines a predefined set of hyperparameter values that are systematically evaluated, and the best design is then returned. Grid search is commonly used for HPO, as it evaluates a predefined set of parameter values to identify the best combination [1, 25]. In this study, we employ a heuristic-based Grid Search, which systematically explores the search space while incorporating heuristic principles to improve efficiency. It will serve as a comparative baseline for MADS and IRACE. The Grid-Rand variant simply performs these evaluations in a random order, to avoid any bias. The Grid-Heuristic variant is a bit more elaborated: It iterates through subsets of the S set evaluating each hyperparameter combination. For each combination $p \in \mathbb{P}$, the performance of the TS target algorithm is measured on the current subset of S, and the best-performing configuration is updated accordingly. If no improvement is found after a given number of iterations, the search terminates, preventing unnecessary evaluations. Unlike Grid-Rand, which exhaustively evaluates all the hyperparameter combinations, Grid-Heuristic introduces a termination condition based on performance stagnation, which helps avoid redundant evaluations.

5.5 Computational experiments

In this section, we evaluate how the different approaches impact the performance of TS. All experiments are performed on the two case studies presented earlier.

5.5.1 Experimental Setup

For each of the case studies, we compare the performance of the HPO methods for TS and the results obtained on the target algorithms previously published in [1, 25]. We use the hyperparameters determined in [1,25] to run the target algorithms. We refer to the solutions obtained as baseline, and the case studies as Assignment^{TS} and Scheduling^{TS} in the remainder of this paper. All methods have the same stopping criteria when running the blackbox.

In the experiments for Assignment^{TS}, we consider 5 instances of 496 patients and 19 nurses where the clustering of patients and staffing differ, with different sets of weights for the three criteria of the objective function: ({1,1,1}, {1,0,1}, {1,100,1}, {100,0,1} and {1,0,100}) in our experiments.

In Scheduling^{TS}, we use 21 generated pseudo-real instances, where the number of new patients arriving each week, their profile and the number of available physicians varies. Number of physicians varies from 6 to 10 and number of patients from 7 to 60. Each instance is labeled $pr-$(# of physicians, # of patients). We refer the reader to [1] for more details on instances.

In both Assignment^{TS} and Scheduling^{TS}, a set of 10 runs was generated ($|S| = 10$). In Assignment^{TS}, each run uses a different random seed, notably used to generate the initial solution for the target algorithm, while in Scheduling^{TS}, each run uses different patients arrival and profiles.

Table 5.1 shows the list of hyperparameters $\mathbb{P}$ for both Assignment^{TS} and Scheduling^{TS}, along with the corresponding boundaries considered across the three HPO methods: MADS, IRACE, and Grid Search. The values for the size of tabu list (p_θ), sequence of neighborhoods (p_n where n represents number of neighborhoods), and the sequence of movements (p_{move} where $move$ represents number of movements) are parameterized with ranges or formulas, where specific variables (e.g., p_θ, p_n and p_{move}) are optimized with given limits.

For example, in Assignment^{TS}, the target algorithm uses 9 neighborhoods, resulting in a total of 9! possible sequences. Considering that some of the neighborhoods are special cases of others, it is possible to reduce the number of sequences to (9 - 3)!. This improves the efficiency of the search process and avoids computationally expensive calculations. Additionally, the target algorithm in Assignment^{TS} uses 5 possible moves, leading to 5! total movement sequences. Similarly, the target algorithm in Scheduling^{TS} considers 3 moves, resulting in 3! possible movement sequences. For the remaining parameters, we consider sufficiently large values to ensure all methods explore a reasonable search space.

In both MADS and IRACE methods, the number of blackbox evaluations and the stopping criterion for the target algorithm (number of iterations) need to be specified. These are

Table 5.1 Hyperparameters and their boundaries for Assignment^{TS} and Scheduling^{TS} Problems

Problem	Size of tabu list	Sequence of neighborhoods	Sequence of movements	Size of neighborhood 1	Size of neighborhood 2
Assignment^{TS}	$p_\theta = [\rho \times \sqrt{\#\text{ of neighbors}}]$ with $\rho \in [1,5]$	$p_n \in [1,(n-3)!]$ with $n = 9$	$p_{move} \in [1, move!]$ with $move = 5$	N/A	N/A
Scheduling^{TS}	$p_\theta \in [\#\text{ of physicians}, 40]$	N/A	$p_{move} \in [1, move!]$ with $move = 3$	$p_{n_1} \in [\#\text{ of physicians}, 240]$	$p_{n_2} \in [1,100]$

problem specific and will be discussed for each case study. The stopping criteria in particular should reflect a compromise between time and quality of results. Indeed, it may differ completely from what was used in the previous publications [1,25], making the results incomparable. To illustrate, the stopping criteria (inside the blackbox) was set to more than 10,000 iterations, which is time consuming. In the context of this study, the number of iterations should be set based on reasonable criteria that balance model performance, computational efficiency, and generalization capability.

The NOMAD software package [117] is used as the MADS implementation.

5.5.2 Experimental Results

We report and analyze results based on the performance of the algorithm. The performance is evaluated by comparing the cost value defined in (5.2) and (5.3) for both Assignment^{TS} and Scheduling^{TS}.

In all tables, the first columns refer to the instance, the second set of columns presents the value of the objective function obtained in the "Baseline" (with the original hyperparameters of Assignment^{TS} or Scheduling^{TS}), and the HPO problem with "MADS", "IRACE", "Grid-Heuristic" and "Grid-Rand" for the two Grid Search variants. The last set of columns provides the gaps when comparing each method to baseline. A negative value in the gap columns signifies that the corresponding method has produced a better solution than the baseline, while a positive value indicates a worse performance. Finally, we summarize the findings in the last two lines, which provide the overall average for each column as well as the improvement ratio; that is, the percentage of problems (out of all problems considered) in which we achieved an improved solution compared to the target algorithms.

Assignment^{TS}

The performance of the optimization methods is assessed on the assignment problem based on Equations (5.2) and (5.3) where $|\mathbb{P}| = 3$. In both MADS and IRACE methods, we considered a limit of maximum 1,000 blackbox evaluations and the stopping criterion for the target algorithm is set to 2,000 iterations for all approaches.

Table 5.2 presents the results obtained for Assignment^{TS} based on Equation (5.2). On average, compared to the baseline, MADS achieved a 0.91% improvement, IRACE achieved 0.35%, and Grid-Heuristic achieved 0.37%. In contrast, Grid-Rand deteriorated the solution by 3.30%. MADS is consistently superior for all instances, while IRACE and the Grid-heuristics yield mixed reesutls. Additionally, Grid-Heursitics appears to perform better than IRACE. improvement over the results obtained with Assignment^{TS}.

Table 5.2 Comparing the performance of different methods in the "Minimization" problem - Assignment^{TS}.

	OBJ					GAP (%)			
ω_1 - ω_2 - ω_3	**Baseline**	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**
1 - 1 - 1	**5,419**	**5,403**	5,421	5,416	5,513	**-0.29**	0.04	-0.05	1.74
1 - 0 - 1	4,530	**4,492**	4,589	4,551	5,113	**-0.85**	1.30	0.46	12.86
1 - 100 - 1	8,721	**8,710**	8,721	8,709	8,873	**-0.13**	0.0	-0.14	1.74
100 - 0 - 1	**8,144**	**8,144**	8,144	8,144	8,144	**0.0**	**0.0**	**0.0**	**0.0**
1 - 0 - 100	96,916	**93,733**	93,914	94,879	97,078	**-3.28**	-3.10	-2.10	0.17
Average	24,746.1	**24,096.4**	24,157.8	24,339.9	24,944.2	**-0.91%**	-0.35%	-0.37%	3.30%
Improvement Ratio						100%	60%	60%	20%

Table 5.3 highlights the robustness of the results achieved by various optimization methods. The results show that, once again, MADS outperforms the Assignment^{TS} method, achieving an average improvement of 5.73%. Notably, most evaluated instances (80%, or 4 out of 5) showed improvements, highlighting the consistency and effectiveness of the MADS approach in enhancing solution quality across all tested scenarios. IRACE shows similar performance, followed by Grid-Heuristic. However, this performance is less consistent (40 to 60% of the instances could be improved).

In summary, MADS consistently produces better solutions across all instances and, on average, outperforms the baseline as well as the other optimization methods. IRACE and Grid-Heuristic exhibit similar performance.

Table 5.3 Comparing the performance of different methods in the “Average” problem - Assignment^{TS}.

	OBJ					GAP (%)			
ω_1 - ω_2 - ω_3	**Baseline**	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**
1 - 1 - 1	**5,534**	5,551	5,567	5,545	5,600	**0.31**	0.60	0.20	1.19
1 - 0 - 1	4,742	**4,707**	4,771	4,726	5,212	**-0.74**	0.61	-0.34	9.91
1 - 100 - 1	8,934	**8,776**	8,947	8,868	8,985	**-1.77**	0.14	-0.74	0.57
100 - 0 - 1	**8,144**	**8,144**	**8,144**	**8,144**	**8,144**	**0.0**	**0.0**	**0.0**	**0.0**
1 - 0 - 100	131,754	**96,916**	93,914	110,773	128,460	-26.44	**-28.72**	-15.92	-2.50
Average	31,821.7	24,818.8	**24,268.6**	27,611.2	31,280.2	**-5.73%**	-5.47%	-3.36%	1.83%
Improvement Ratio						**80%**	40%	**80%**	40%

Scheduling^{TS}

The performance of Scheduling^{TS} is evaluated based on Equations (5.2) and (5.3) with $|\mathbb{P}| = 4$. In both MADS and IRACE methods, we considered a limit of maximum 1,000 blackbox evaluations and the stopping criterion for the target algorithm (inside the blackbox) is set to 3,000 iterations for all approaches.

From Table 5.4, it is evident that MADS consistently achieved the best results among all methods for nearly all problems. Specifically, MADS outperformed the baseline in 95.24% of the cases (20 out of 21 instances), resulting in an average improvement of 5.25%. This highlights the superior ability of MADS to optimize the problem. The improvements were 2.01%, 3.93% and 9.82% on average for small, medium and large set of instances respectively.

Additionally, IRACE, Grid-Heuristic, and Grid-Rand achieved comparable results with an average improvement of 1%. However, this improvement was less consistent, as indicated by an improvement ratio of 61.9% to 80% across all instances. It is worth to note that the Grid-Heuristic method has demonstrated strong performance, delivering an average improvement of 1.94% over the baseline. It improved the results in 80.95% of the instances (18 out of 21), making it a reliable alternative for optimization in most scenarios.

Finally, we observe that the improvement increases as the size of instances increase, from 2% improvement to more than 9%.

Table 5.5 presents the results when using Equation (5.3). It is again evident that MADS outperforms the baseline, achieving an average improvement of 2.46%. Notably, this method exhibits significant enhancements across 71.43% of the tested instances, specifically improving 15 out of 21 cases, highlighting the robustness and reliability of MADS in diverse scenarios.

Additionally, IRACE and the Grid-Heuristic method demonstrate commendable performance as well, with an average improvement of 0.5%, with a success rate of 47 to 62% of the

Table 5.4 Comparing the performance of different methods in the "Maximization" problem - Scheduling^{TS}.

		OBJ					GAP (%)			
		Baseline	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**
Small	Pr-(6,7)	409	**420**	413	**420**	416	**-2.69**	-0.98	**-2.69**	-1.71
	Pr-(6,9)	395	**403**	396	401	394	**-2.03**	-0.25	-1.52	0.25
	Pr-(6,11)	**392**	385	378	385	383	1.79	3.57	1.79	2.30
	Pr-(6,12)	367	**390**	372	389	376	**-6.27**	-1.36	-5.99	-2.45
	Pr-(8,7)	567	**574**	558	**574**	570	**-1.23**	1.59	**-1.23**	-0.53
	Pr-(8,9)	552	**566**	557	565	557	**-2.54**	-0.91	-2.36	-0.91
	Pr-(10,7)	726	**734**	724	**734**	726	**-1.10**	0.28	**-1.10**	0.00
	Average:	486.86	496.00	485.43	495.43	489.86	**-2.01**	0.28	-1.10	0.00
Medium	Pr-(6,20)	315	**343**	318	336	327	**-8.89**	-0.95	-6.67	-3.81
	Pr-(8,11)	544	**553**	536	550	547	**-1.65**	1.47	-1.10	-0.55
	Pr-(8,12)	526	**543**	**543**	542	537	**-3.23**	**-3.23**	-3.04	-2.09
	Pr-(8,20)	464	**503**	476	497	495	**-8.41**	-2.59	-7.11	-6.68
	Pr-(10,9)	709	**721**	715	**721**	709	**-1.69**	-0.85	**-1.69**	0.00
	Pr-(10,11)	705	**710**	695	709	706	**-0.71**	1.42	-0.57	-0.14
	Pr-(10,12)	687	**707**	693	**707**	689	**-2.91**	-0.87	**-2.91**	-0.29
	Average:	564.29	582.86	568.00	580.29	571.00	**-3.93**	-0.80	-3.30	-1.94
Large	Pr-(6,40)	219	249	**256**	233	231	-13.7	**-16.89**	-6.39	-5.48
	Pr-(6,60)	121	**139**	109	113	101	**-14.88**	9.92	6.61	16.53
	Pr-(8,40)	398	**411**	388	376	386	**-3.27**	2.51	5.53	3.02
	Pr-(8,60)	250	**309**	265	263	276	**-23.60**	-6.00	-5.20	-10.40
	Pr-(10,20)	642	665	**671**	665	661	-3.58	**-4.52**	-3.58	-2.96
	Pr-(10,40)	546	568	**575**	555	542	-4.03	**-5.31**	-1.65	0.73
	Pr-(10,60)	438	**463**	425	437	433	**-5.71**	2.97	0.23	1.14
	Average:	373.43	400.57	384.14	377.43	375.71	**-9.82**	-2.48	-0.64	0.37
	Average:	474.86	**493.81**	482.52	484.38	478.86	**-5.25%**	-1.00%	-1.94%	-0.67%
	Improvement Ratio:						**95.24%**	61.90%	80.95%	61.90%

instances. While IRACE requires to use a package, the Grid-heuristic stands out as an alternative optimization technique.

Table 5.5 Comparing the performance of different methods in the “Average” problem - Scheduling^{TS}.

		OBJ					GAP (%)			
		Baseline	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**	**MADS**	**IRACE**	**Grid-Heuristic**	**Grid-Rand**
Small	Pr-(6,7)	404.45	406.6	**407.36**	403.9	379.6	-0.53	**-0.72**	0.14	6.14
	Pr-(6,9)	389.27	**392.39**	388	391.7	361.2	**-0.80**	0.33	-0.62	7.21
	Pr-(6,11)	**386.82**	371.19	375.18	370.2	338.5	4.04	**3.01**	4.30	12.49
	Pr-(6,12)	357.45	**367.69**	360.75	365.6	336	**-2.86**	-0.92	-2.28	6.00
	Pr-(8,7)	562.55	564	**564.2**	563.7	522.3	-0.26	**-0.29**	-0.21	7.15
	Pr-(8,9)	545.45	**552**	547.63	551.9	512.6	**-1.20**	-0.40	-1.18	6.02
	Pr-(10,7)	**723.18**	723	720.5	722.5	672.2	0.03	0.37	**-0.09**	7.05
	Average:	481.31	**482.41**	480.52	481.36	446.06	**-0.23**	0.20	0.03	7.44
Medium	Pr-(6,20)	297.91	**310.6**	302.22	307	274.9	**-4.26**	-1.45	-3.05	7.72
	Pr-(8,11)	**537.64**	537	536	535.4	495.6	**0.12**	0.30	0.42	7.82
	Pr-(8,12)	515.18	**531**	509.45	528.9	485.9	**-3.07**	1.11	-2.66	5.68
	Pr-(8,20)	447.73	**478**	454	476.6	419.3	**-6.76**	-1.40	-6.45	6.35
	Pr-(10,9)	701.00	**709**	699	707.2	655.2	**-1.14**	0.29	-0.88	6.53
	Pr-(10,11)	**698.73**	698	694.44	696.4	644.5	**0.10**	0.61	0.33	7.76
	Pr-(10,12)	681.55	**691**	674.7	690	635.4	**-1.39**	1.00	-1.24	6.77
	Average:	554.25	**564.94**	552.83	563.07	515.83	**-2.34**	0.07	-1.93	6.95
Large	Pr-(6,40)	185.27	196.5	**197.64**	190.1	147.9	-6.06	**-6.68**	-2.61	20.17
	Pr-(6,60)	77.36	**82**	79.44	63	10.1	**-5.99**	-2.68	18.57	86.94
	Pr-(8,40)	**368.73**	366	365.6	356.4	297.3	**0.74**	0.85	3.34	19.37
	Pr-(8,60)	215.82	**238**	218.2	234	139.6	**-10.28**	-1.10	-8.42	35.32
	Pr-(10,20)	624.45	640	616.18	**640.5**	570.5	**-2.49**	1.33	-2.57	8.64
	Pr-(10,40)	**525.18**	516	521.54	517.2	447.9	1.75	**0.69**	1.52	14.72
	Pr-(10,60)	374.36	**417**	392.7	402.4	307.8	**-11.39**	-4.90	-7.49	17.78
	Average:	338.74	**350.79**	341.61	343.37	274.44	**-4.82**	-1.79	0.33	28.99
	Average	458.10	**466.05**	458.32	462.6	412.11	**-2.46%**	-0.51%	-0.52%	14.46%
	Improvement Ratio						**71.43%**	47.62%	61.90%	0.00%

Overall, the results highlight the strengths of both MADS and Grid-Heuristic methods in optimizing costs compared to the baseline, indicating their potential for practical applications in various optimization problems.

5.5.3 Sensitivity Analysis

In this section, we present more supplementary and sensitivity analysis on the performance of HPO problem using three methods: MADS, IRACE, and Grid Search, as applied to the Scheduling^{TS} with respect to criterion (5.2), where the objective to find the best overall solution. This problem exhibited the greatest variation in performance among the optimization methods.

Additionally, since MADS has shown the best performance, we conduct a sensitivity analysis, focusing on its response to variations in starting points and S.

Hyperparameter sensitivity in tabu search

We start by comparing the hyperparameter configurations obtained from different methods to evaluate their impact on the performance of the target algorithm. By analyzing variations in key hyperparameters, we aimed to understand how different optimization approaches influence solution quality.

Table 5.6 Comparison of TS Hyperparameter Configurations with different methods in Scheduling^{TS} - Maximization

Experiment	Method	Size of tabu list (p_θ)	Sequence of movements (p_{move})	Size of neighborhood 1 (p_{n_1})	Size of neighborhood 2 (p_{n_2})	OBJ
Pr-(6,12)	Baseline	19	1	40	1	367
	MADS	7	2	6	1	**390**
	IRACE	11	1	22	12	372
	Grid-Heuristic	6	2	6	1	389
	Grid-Rand	6	1	6	1	376
Pr-(8,20)	Baseline	19	1	58	1	464
	MADS	9	2	18	1	**503**
	IRACE	17	2	20	16	476
	Grid-Heuristic	8	1	8	20	497
	Grid-Rand	8	3	8	20	495
Pr-(10,20)	Baseline	19	1	60	1	642
	MADS	10	1	10	1	665
	IRACE	15	2	30	43	**671**
	Grid-Heuristic	10	2	10	1	665
	Grid-Rand	14	2	42	20	661

Table 5.6 presents a comparison of the TS hyperparameter values for Scheduling^{TS} - Maximization across three instances: Pr-(6,12), Pr-(8,20), and Pr-(10,20). These instances were selected as representative cases for small, medium, and large problem sizes, respectively. The table highlights the impact of key hyperparameters, tabu list size (p_θ), sequence of movements (p_{move}), and neighborhood sizes (p_{n_1}, p_{n_2}). The tabu list size (p_θ) controls how long solutions remain forbidden, influencing search diversification. The baseline uses a fixed large value (19), while HPO methods favor smaller values (e.g., 7, 9, 10), suggesting that a more dynamic tabu list improves exploration. The sequence of movements (p_{move}) determines the order in which movements are used. The baseline is fixed at 1, whereas HPO methods vary this parameter (e.g., 2 or 3), indicating that flexibility in search order enhances performance. The size of neighborhood (p_{n_1}, p_{n_2}) defines the number of candidate solutions examined at each step. The baseline uses a large value for p_{n_1} and a minimal value for p_{n_2} (e.g., 40 and 1), while HPO methods explore more balanced or diverse sizes (e.g., 6 & 1, 30 & 43), showing that optimizing these values is crucial for better results. The results suggest that smaller

tabu lists, flexible movement sequences, and well-balanced neighborhood sizes lead to better search efficiency and higher objective values.

Generalization of MADS hyperparameters to different problem scales

To evaluate the generalization of hyperparameter optimization, we applied the set of hyperparameters obtained by MADS for the representative instances Pr-(6,12), Pr-(8,20) and Pr-(10,20) to solve other instances within the categories of small, medium, and large instances, respectively. This analysis aims to determine whether a hyperparameter configuration optimized for a specific problem size can be effectively transferred to similar-sized problems while maintaining competitive performance. By assessing the impact of this approach, we examine its effectiveness in reducing the computational cost of hyperparameter optimization while ensuring consistent solution quality across different instances within the same problem scale. We call this approach static MADS and Table 5.7 compares the results obtained through the MADS and static MADS strategies.

The results show that while static MADS consistently improves performance compared to baseline, it does not match the effectiveness of MADS in all problem sizes. In particular, the performance gap between MADS and static MADS widens as the problem size increases, suggesting that static MADS may struggle with larger-scale problems compared to the fully adaptive MADS approach. In particular, MADS achieves the highest overall improvement, with an average increase of 5.25%, while static MADS improves by 3.14%. Furthermore, the improvement ratio shows that MADS outperforms the baseline in 95.24% of cases, whereas static MADS does so in 80.95% of cases. These findings highlight the advantage of MADS in effectively optimizing hyperparameters for complex problems.

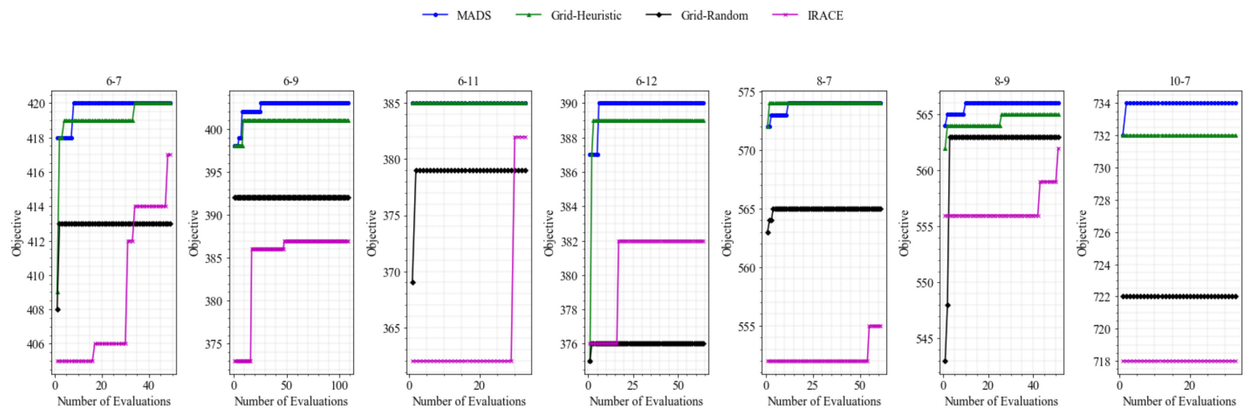
Efficiency of the HPO approaches: A convergence and performance comparison

To assess the efficiency of the HPO approaches, we use the Scheduling^{TS} problem using Equation (5.2).

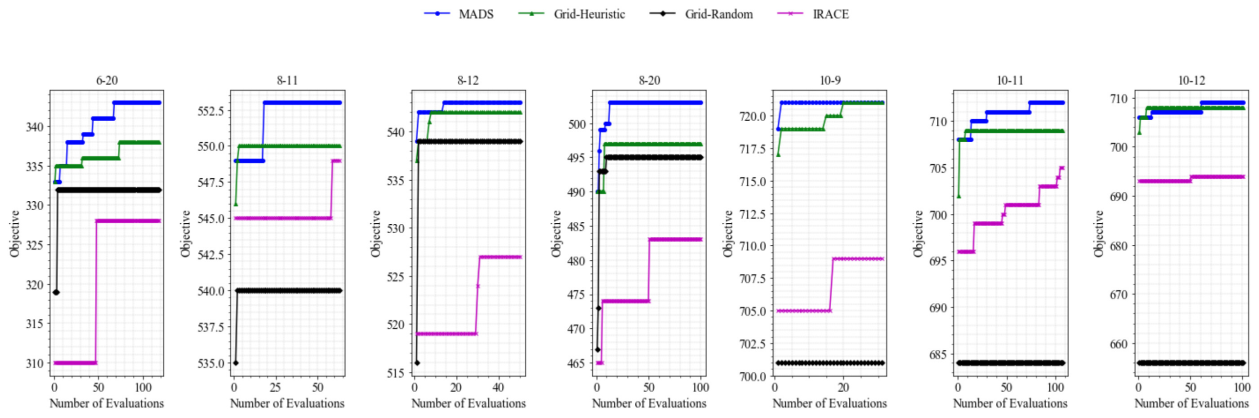
Figure 5.2 compares the convergence speed between all approaches. In all instances, we can see that MADS (which is represented with blue line) converges faster than other methods and requires fewer (less than 20) evaluations to converge to the best solution. In most cases, the Grid-Heuristic algorithm (green line) achieved results comparable to MADS but required a greater number of evaluations. On the other hand, IRACE demonstrated lower performance for this problem, with slower convergence compared to the other methods. This is consistent regardless of the size of the problem.

Table 5.7 Comparing the performance of MADS vs. Static MADS in the "Maximization" problem - Scheduling^{TS}

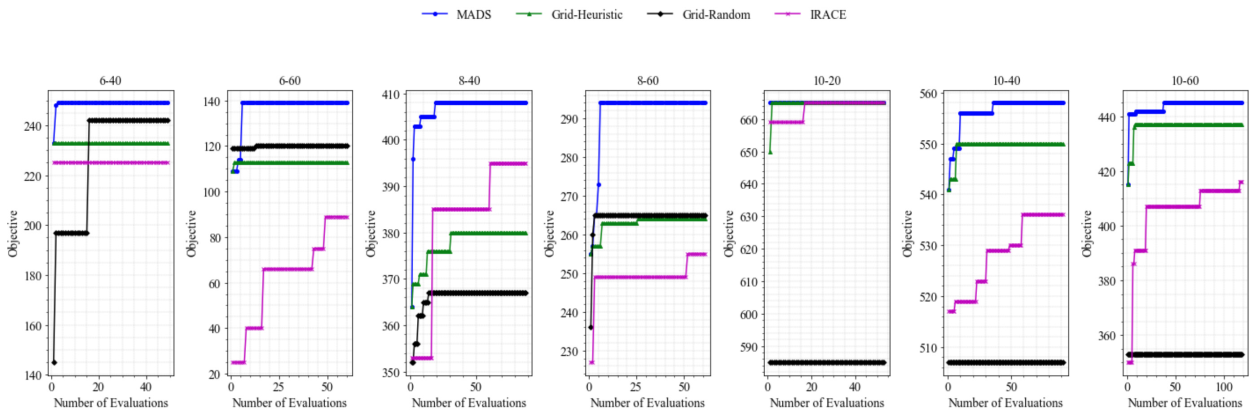
		OBJ			GAP (%)	
		Baseline	**MADS**	**Static MADS**	**MADS**	**Static MADS**
Small	Pr-(6,7)	409	**420**	416	**-2.69**	-1.71
	Pr-(6,9)	395	**403**	398	**-2.03**	-0.76
	Pr-(6,11)	**392**	385	384	**1.79**	2.04
	Pr-(6,12)	367	**390**	**390**	**-6.27**	**-6.27**
	Pr-(8,7)	567	**574**	573	**-1.23**	-1.06
	Pr-(8,9)	552	**566**	559	**-2.54**	-1.27
	Pr-(10,7)	726	**734**	733	**-1.10**	-0.96
	Average:	486.86	496.00	493.29	**-2.01**	-1.43
Medium	Pr-(6,20)	315	**343**	339	**-8.89**	-7.62
	Pr-(8,11)	544	**553**	548	**-1.65**	-0.74
	Pr-(8,12)	526	**543**	540	**-3.23**	-2.66
	Pr-(8,20)	464	**503**	**503**	**-8.41**	**-8.41**
	Pr-(10,9)	709	**721**	719	**-1.69**	-1.41
	Pr-(10,11)	705	**710**	704	**-0.71**	0.14
	Pr-(10,12)	687	**707**	705	**-2.91**	-2.62
	Average:	564.29	582.86	579.71	**-3.93**	-3.33
Large	Pr-(6,40)	219	249	248	**-13.7**	-13.24
	Pr-(6,60)	121	**139**	136	**-14.88**	-12.40
	Pr-(8,40)	398	**411**	401	**-3.27**	-0.75
	Pr-(8,60)	250	**309**	272	**-23.60**	-8.80
	Pr-(10,20)	642	**665**	**665**	**-3.58**	**-3.58**
	Pr-(10,40)	546	**568**	541	**-4.03**	0.92
	Pr-(10,60)	438	**463**	415	**-5.71**	5.25
	Average:	373.43	400.57	382.57	**-9.82**	-4.66
	Average:	474.86	**493.81**	485.19	**-5.25%**	-3.14%
	Improvement Ratio:				**95.24%**	80.95%



(a) Small



(b) Medium



(c) Large

Figure 5.2 Convergence speed among different approaches for the "Maximization" problem - Scheduling^{TS}

Figure 5.3 presents the data profiles. Data profiles are graphical tools used to compare the performance of optimization algorithms over a range of test problems. In these plots, the x-axis shows a normalized computational budget (i.e., function evaluations), while the y-axis indicates the fraction of problems solved to a specified tolerance. Introduced in [118], this approach is especially valuable for benchmarking derivative-free optimization methods, as it visually highlights both efficiency and robustness across various problems. In Figure 5.3, each curve corresponds to a different method. Curve trends reflect the efficiency and convergence behavior of each approach. This figure demonstrates that both MADS and Grid-Heuristic achieved superior results. We can also observe that Grid-Random has extremely poor performance, resulting in an almost flat or non-existent data profile curve.

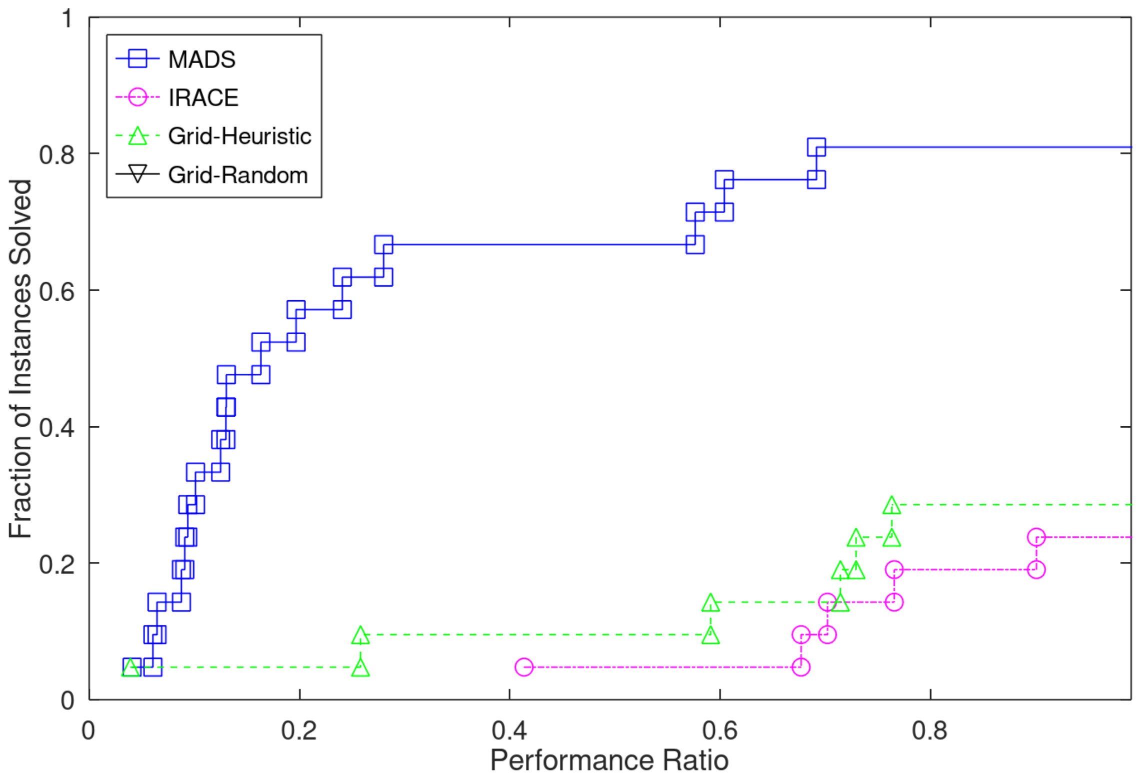


Figure 5.3 Performance comparison of HPO approaches: data profiles for the "Maximization" problem - Scheduling^{TS}

We also compare the relative strengths and weaknesses of each algorithm under varying resource constraints and budgets. Figure 5.4 presents a comparison between the MADS and IRACE packages under different computational budgets for blackbox. The box plots illustrate the distributions of GAP values relative to the baseline when allocating budgets of 3,000 and 10,000 iterations to blackbox. The range of the box plots indicate an impact of increasing the number of iterations from 3,000 to 10,000.

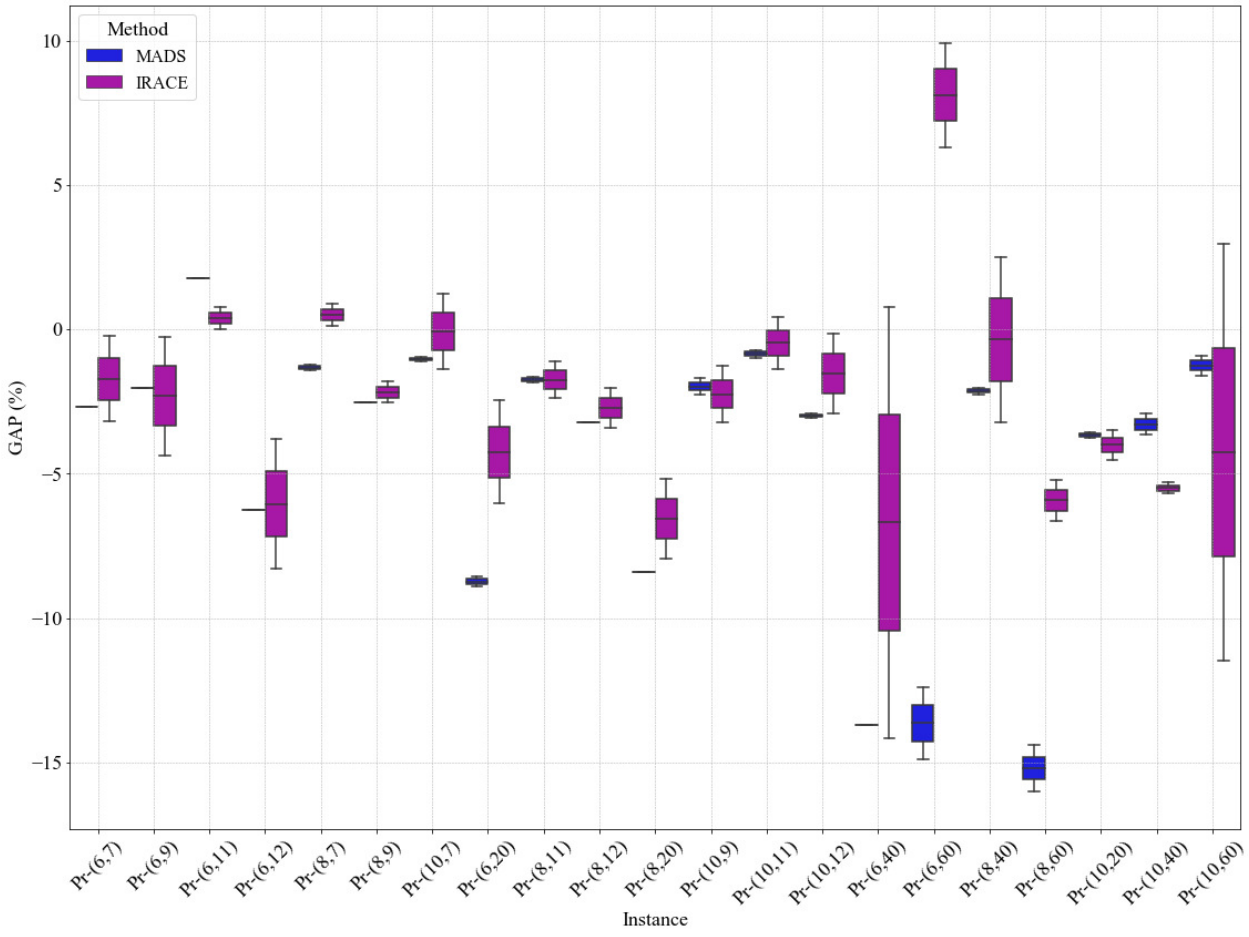


Figure 5.4 Comparing the impact of blackbox stopping criterion in MADS and IRACE methods for the “Maximization” problem - Scheduling^{TS}

Figure 5.4 shows that MADS converges faster and finds better, more stable solutions than IRACE. MADS consistently produces low GAP values (very small range to no range in the boxplots). IRACE exhibits great variability; its performance improves with a higher budget, showing that it does require more effort to reach best solutions.

In addition, as the size of problem increases, the GAP values for IRACE show more variation, whereas MADS maintains steady performance. For small problems, both methods perform similarly, but for medium and large instances, MADS clearly delivers more robust and reliable results.

Starting points and size of S - MADS

Table 5.8 focuses on the sensitivity of MADS to different starting points, again in reference to Equation (5.2). The columns labeled "SP_i" with $i \in \{1, 2, \dots, 10\}$, report the results obtained from running MADS with 10 distinct starting points. This table emphasizes the significant impact that the choice of initial conditions has on the performance of the algorithm. Varying the starting points leads to diverse outcomes, illustrating MADS's sensitivity to initialization, which can result in the algorithm either converging to local optima or discovering near-global solutions.

Table 5.8 Comparing the performance of MADS with different starting points - Scheduling^{TS}

		SP_1	SP_2	SP_3	SP_4	SP_5	SP_6	SP_7	SP_8	SP_9	SP_{10}
Small	Pr-(6,7)	419	**420**	**420**	419	395	419	419	420	419	419
	Pr-(6,9)	**403**	**403**	402	384	**403**	**403**	**403**	402	401	**403**
	Pr-(6,11)	**391**	386	389	389	361	389	387	389	389	390
	Pr-(6,12)	390	388	390	**391**	390	390	370	389	389	388
	Pr-(8,7)	**575**	574	574	574	537	573	574	574	573	574
	Pr-(8,9)	**566**	564	**566**	**566**	523	565	**566**	**566**	564	**566**
	Pr-(10,7)	**734**	733	733	733	677	**734**	**734**	**734**	**734**	731
Medium	Pr-(6,20)	**343**	342	340	339	341	339	337	340	342	341
	Pr-(8,11)	551	552	552	**554**	510	551	550	553	552	551
	Pr-(8,12)	543	542	**544**	**544**	542	543	543	543	541	542
	Pr-(8,20)	**503**	502	500	498	496	498	496	499	498	498
	Pr-(10,9)	721	720	**722**	**722**	659	**722**	719	**722**	720	720
	Pr-(10,11)	710	711	712	713	711	**714**	710	713	713	709
	Pr-(10,12)	**708**	707	**708**	691	**708**	**708**	**708**	**708**	706	707
Large	Pr-(6,40)	**249**	**249**	**249**	248	208	**249**	218	**249**	**249**	247
	Pr-(6,60)	**136**	131	132	**136**	68	132	99	**136**	**136**	**136**
	Pr-(8,40)	406	401	407	404	359	406	405	**408**	405	**408**
	Pr-(8,60)	286	287	288	274	273	283	270	**309**	306	292
	Pr-(10,20)	**665**	661	658	660	664	664	656	662	662	663
	Pr-(10,40)	562	555	555	558	558	559	555	563	**568**	560
	Pr-(10,60)	442	446	**453**	422	450	432	442	435	441	447

We also analyze the impact of the size of S on the performance of MADS by varying the size of S. Figure 5.5 illustrates the performance of MADS for the Scheduling^{TS} problem with $|S| = \{10, 30, 50\}$, where the variation in the objective value for $|S| = 10$, $|S| = 30$, and $|S| = 50$ is illustrated.

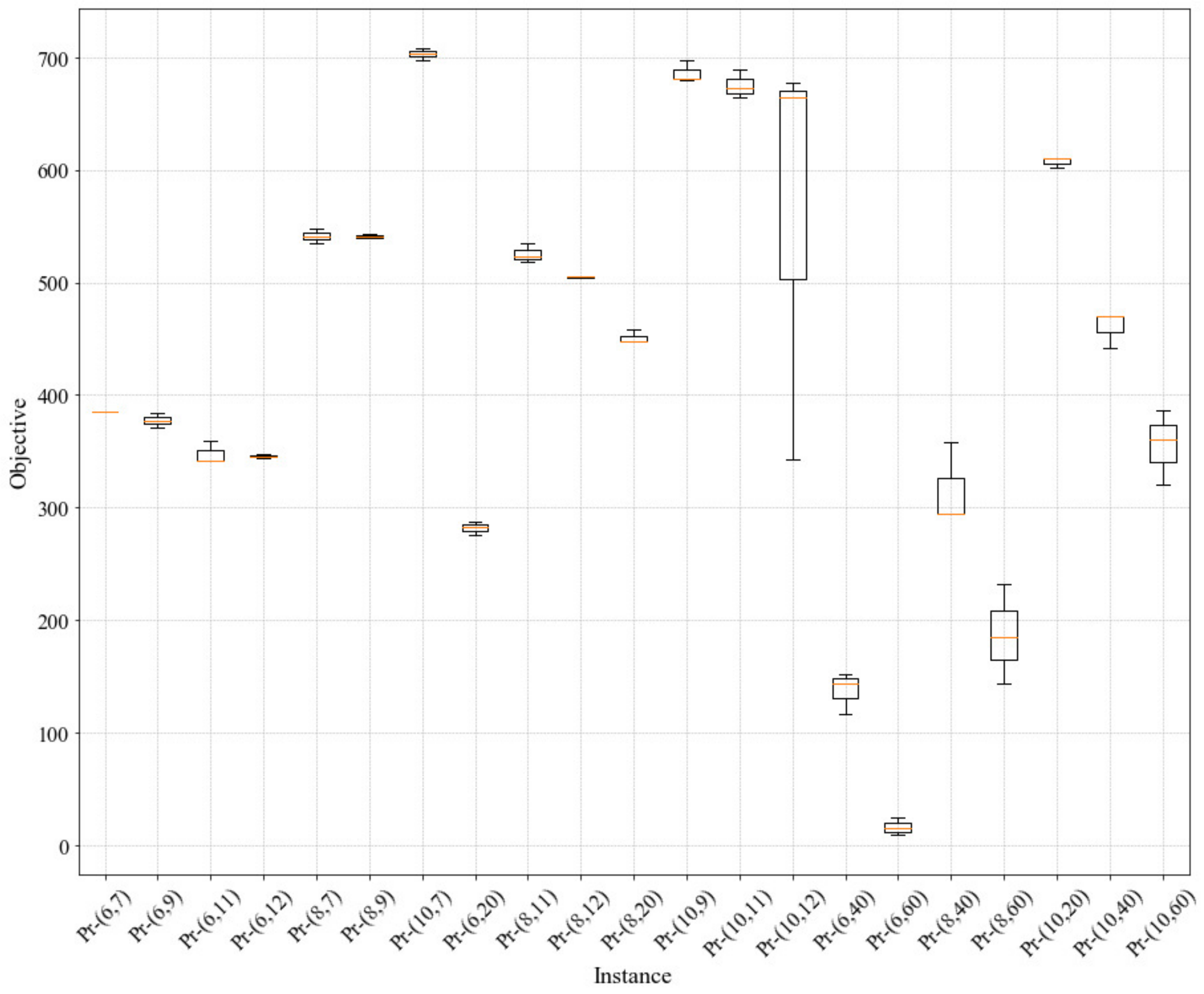


Figure 5.5 Comparing the impact of different sizes for S for the "Maximization" problem - Scheduling^{TS}

As shown in Figure 5.5 the size of the S has a minimal effect on the algorithm's performance, except for one instance, Pr-(10,12). This indicates that even with $|S| = 10$, the algorithm was able to gather sufficient information about the search space.

5.6 Discussion

The analysis presented in Section 5.5.2 and Section 5.5.3 highlights several key insights.

Parameter Tuning in Tabu Search The results demonstrate that optimizing the hyperparameters of the TS algorithm has a direct and positive impact on its performance across various problem types.

Static optimization The sensitivity analysis between MADS and static MADS highlights the impact of S and the method used to obtain hyperparameters on algorithm performance. The results indicate that the choice of evaluation runs plays an important role in determining the optimized hyperparameters. This suggests that selecting hyperparameters based on evaluations within the same problem size category can lead to more consistent and reliable performance improvements.

Sensitivity to Starting Points The sensitivity analysis of MADS highlights an important challenge in optimization algorithms, the reliance on the choice of initial conditions. Our findings suggest that while MADS is a powerful algorithm, its performance can vary widely based on the starting point.

Sensitivity to the Size of S The sensitivity analysis of MADS illustrates that the size of S had a minimal effect on algorithm performance. This suggests that even with a reduced size S, the algorithm effectively captured the necessary information about the search space. These findings highlight the robustness of the MADS algorithm and its ability to maintain efficiency with limited evaluation data.

5.7 Conclusion

This study demonstrated the benefits of integrating HPO methods with the TS algorithm. By integrating a HPO approach with the guided exploration and exploitation capabilities of TS, we designed an approach capable of effectively balancing intensification and diversification in the search process.

The experimental results demonstrate that our proposed methods consistently outperform standalone applications of TS. When evaluated two case studies and real-world optimization scenarios, the proposed methods exhibited faster convergence rates, higher solution quality, and greater robustness in navigating complex search spaces.

For Scheduling^{TS}, MADS improved TS results by 5.25% and 2.46% relative to finding the overall best solution and the most robust set of hyperparameters, respectively. Similarly, significant improvements were observed in Assignment^{TS}, with enhancements of 0.91% and 5.73%. Although the Grid-Heuristic approach performed well and can serve as an alternative to MADS, MADS remains more efficient and scalable than Grid Search, especially for high-dimensional, mixed-variable, and computationally expensive scenarios like TS hyperparameter optimization. While Grid Search is simple and effective for low-dimensional problems, MADS is better suited for complex tasks that require adaptive refinement.

Additionally, the analysis revealed that MADS is more sensitive to the starting point while being less affected by the size of S. Furthermore, the sensitivity analysis between MADS and static MADS highlights the impact of S and the strategy used to obtain hyperparameters on algorithm performance.

In conclusion, the proposed methods allow fine-tuning of hyperparameters, such as the size of tabu tenure, and neighborhood structures, providing users with a highly customizable algorithm for a wide range of problem types.

In addition, the proposed algorithms demonstrated success in handling constrained and unconstrained problems alike; this makes them a valuable contribution to the growing body of optimization literature. Future work will focus on expanding the applicability and efficiency of this framework.

CHAPTER 6 GENERAL DISCUSSION

In this chapter, we discuss the key contributions and findings of the two presented papers in Chapter 4 and Chapter 5, focusing on how they improve the performance of the TS algorithm. The integration of ML techniques and HPO methods represents an advancement in the field of optimization, which leads to the improvement of performance and adaptability of TS in various problem domains.

The first paper, presented in Chapter 4, introduces a novel approach to improve the performance of TS, particularly in scheduling applications. However, this approach can be easily adapted to other optimization problems. One of its key contributions is the identification of critical features for training ML models. By analyzing the search process, we determined how ML techniques can help reduce the search space and identified features that assist ML models in learning the behavior of the search space. In addition, we integrate logistic regression and decision trees to refine the search process. These methods allow TS to focus on promising neighborhoods, reducing computational effort while improving solution quality for stochastic environment, and obtaining comparable solutions in a deterministic environment. The study also explores adaptive learning mechanisms, demonstrating how both online and offline learning strategies can improve algorithm performance through real-time adaptation. The proposed

The second paper, presented in Chapter 5, highlights the importance of hyperparameter optimization to improve TS performance. This study evaluates different HPO methods, including MADS, IRACE, and Grid Search, showing their effectiveness in solving multi-objective problems. By optimizing hyperparameters, the study demonstrates improvements in convergence rates and overall solution quality, reinforcing the idea that well-tuned configurations lead to better solutions. Furthermore, extensive testing across different problem domains, such as the Physician Scheduling Problem and the Multi-Resource Generalized Assignment Problem, demonstrates the versatility of these HPO approaches. The results suggest that these methods can be applied effectively to real-world challenges in various fields.

In general, the findings of these studies show how the TS algorithm can be improved for better performance. The integration of ML techniques not only enhances the algorithm's ability to explore and exploit the search space but also establishes a framework for adaptive learning that responds to changing conditions. Additionally, the emphasis on hyperparameter optimization highlights the importance of aligning algorithm hyperparameters with specific problem characteristics to achieve more effective results.

CHAPTER 7 CONCLUSION

This thesis successfully improves the performance of TS through the integration of ML techniques and HPO methods. It contributes to the field of optimization by improving computational efficiency and solution quality in different optimization problems. In the following, we first present the contributions achieved by this dissertation. Then we highlight the limitations of the proposed methods and possible future research directions.

7.1 Summary of works

A primary contribution of this work is the enhancement of the exploration and exploitation capabilities in TS. The algorithm can identify and focus on promising neighborhoods by using ML techniques such as logistic regression and decision trees. This approach reduces computational effort while maintaining, and/or improving, solution quality. Both online and offline learning techniques provided valuable insights.

This research also addressed the challenge of hyperparameter optimization for multi-objective problems. We optimized key TS hyperparameters by evaluating methods such as MADS, IRACE, and Grid Search, leading to better performance. These improvements help TS to adapt more effectively to different optimization problems. Extensive testing on different optimization problems shows the flexibility of these methods.

7.2 Limitations and future research

Despite the improvements presented in this thesis, several limitations must be acknowledged. First, while the integration of ML techniques has shown promise in improving the exploration and exploitation capabilities of TS, the effectiveness of these methods may vary depending on the specific characteristics of the optimization problems addressed. More research is needed to assess the generalizability of the proposed approaches across a wider range of problems.

Furthermore, optimizing the hyperparameters of TS has demonstrated a direct and positive impact on performance across different types of problem. The sensitivity analysis between MADS and static MADS highlights the influence of both the hyperparameter selection method and the choice of evaluation instances. The findings suggest that selecting hyperparameters within the same problem size category can lead to more consistent and reliable improvements.

Another key limitation is the sensitivity of the HPO methods to the starting conditions. The MADS analysis showed that the algorithm's performance can vary significantly based on the initial conditions, emphasizing the need for robust initialization strategies. Furthermore, MADS sensitivity analysis with respect to the size of S revealed that reducing S had a minimal impact on performance, indicating that the algorithm effectively captures necessary search space information even with fewer evaluations.

Building on these contributions and limitations, several directions for future research can be explored. One potential direction is the integration of more advanced ML techniques, such as ensemble learning, to further refine the identification of promising search regions. These approaches could improve search efficiency and provide deeper insight.

Moreover, a more extensive benchmarking effort involving various optimization problems could provide stronger empirical validation of the proposed methods. Investigating the scalability of the enhanced TS approaches in real-world and large-scale applications, such as logistics or energy management, would further demonstrate their practical impact.

In conclusion, while this thesis represents significant progress in integrating ML techniques and HPO methods into TS, addressing its limitations and exploring future research directions will contribute to further refining these methods. Advancing these efforts will help develop more robust and efficient optimization algorithms capable of solving complex problems.

REFERENCES

[1] N. Niroumandrad and N. Lahrichi, “A stochastic tabu search algorithm to align physician schedule with patient flow,” *Health care management science*, vol. 21, no. 2, pp. 244–258, 2018.

[2] F. Glover, “Tabu search: A tutorial,” *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[3] E.-G. Talbi, “A taxonomy of hybrid metaheuristics,” *Journal of heuristics*, vol. 8, pp. 541–564, 2002.

[4] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, “Hybrid metaheuristics in combinatorial optimization: A survey,” *Applied Soft Computing*, vol. 11, no. 6, pp. 4135–4151, 2011.

[5] Y. Bengio, A. Lodi, and A. Prouvost, “Machine learning for combinatorial optimization: a methodological tour d’horizon,” *European Journal of Operational Research*, 2020.

[6] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Automated configuration of mixed integer programming solvers,” in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings.* Springer, 2010, pp. 186–202.

[7] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown, and T. Stützle, “Aclib: A benchmark library for algorithm configuration,” in *Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers 8.* Springer, 2014, pp. 36–40.

[8] M. López-Ibáñez and T. Stützle, “Automatically improving the anytime behaviour of optimisation algorithms,” *European Journal of Operational Research*, vol. 235, no. 3, pp. 569–582, 2014.

[9] C. Thornton, F. Hutter, H. H. Hoos, and K. 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*, 2013, pp. 847–855.

[10] B. Adenso-Diaz and M. Laguna, “Fine-tuning of algorithms using fractional experimental designs and local search,” *Operations research*, vol. 54, no. 1, pp. 99–114, 2006.

[11] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil, “Using experimental design to find effective parameter settings for heuristics,” *Journal of Heuristics*, vol. 7, no. 1, pp. 77–97, 2001.

[12] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, “A racing algorithm for configuring metaheuristics,” in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2002, pp. 11–18.

[13] C. Ansótegui, M. Sellmann, and K. Tierney, “A gender-based genetic algorithm for the automatic configuration of algorithms,” in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2009, pp. 142–157.

[14] P. Balaprakash, M. Birattari, and T. Stützle, “Improvement strategies for the f-race algorithm: Sampling design and iterative refinement,” in *Hybrid Metaheuristics: 4th International Workshop, HM 2007, Dortmund, Germany, October 8-9, 2007. Proceedings 4*. Springer, 2007, pp. 108–122.

[15] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, “Paramils: an automatic algorithm configuration framework,” *Journal of artificial intelligence research*, vol. 36, pp. 267–306, 2009.

[16] V. Nannen and A. E. Eiben, “A method for parameter calibration and relevance estimation in evolutionary algorithms,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 183–190.

[17] M.-C. Riff and E. Montero, “A new algorithm for reducing metaheuristic design effort,” in *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 3283–3290.

[18] T. Bartz-Beielstein, “Experimental research in evolutionary computation: The new experimentalism (natural computing series),” *Springer*, vol. 1, pp. 10–73.

[19] F. Hutter, H. H. Hoos, and K. 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*. Springer, 2011, pp. 507–523.

[20] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Cham, Switzerland: Springer, 2017.

[21] R. Battiti, M. Brunato, and F. Mascia, *Reactive search and intelligent optimization.* Springer Science & Business Media, 2008, vol. 45.

[22] G. Karafotias, M. Hoogendoorn, and Á. E. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 167–187, 2014.

[23] C. Audet and J. E. Dennis Jr, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on optimization*, vol. 17, no. 1, pp. 188–217, 2006.

[24] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[25] A. Hertz and N. Lahrichi, "A patient assignment algorithm for home care services," *Journal of the Operational Research Society*, vol. 60, no. 4, pp. 481–495, 2009.

[26] S. Sujitjorn, T. Kulworawanichpong, P. Dcacha, and A. Kongpan, "Adaptive tabu search and applications in engineering design," *Integrated Intelligent Systems for Engineering Design*, vol. 149, p. 233, 2006.

[27] Y. A. Kochetov and E. N. Goncharov, "Probabilistic tabu search algorithm for the multi-stage uncapacitated facility location problem," in *Operations research proceedings.* Springer, 2001, pp. 65–70.

[28] R. Battiti and G. Tecchiolli, "The reactive tabu search," *ORSA journal on computing*, vol. 6, no. 2, pp. 126–140, 1994.

[29] S. Suwannarongsri and D. Puangdownreong, "Adaptive tabu search for traveling salesman problems," *International Journal of Mathematics and Computers in Simulation*, vol. 6, no. 2, pp. 274–281, 2012.

[30] D. Puangdownreong, T. Kulworawanichpong, and S. Sujitjorn, "Finite convergence and performance evaluation of adaptive tabu search," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems.* Springer, 2004, pp. 710–717.

[31] M. Gendreau and J.-Y. Potvin, "Tabu search," in *Search methodologies.* Springer, 2005, pp. 165–186.

[32] Y. Chen, J.-K. Hao, and F. Glover, "An evolutionary path relinking approach for the quadratic multiple knapsack problem," *Knowledge-Based Systems*, vol. 92, pp. 23–34, 2016.

[33] Y. Jin and J.-K. Hao, "Hybrid evolutionary search for the minimum sum coloring problem of graphs," *Information Sciences*, vol. 352, pp. 15–34, 2016.

[34] X. Lai and J.-K. Hao, "A tabu search based memetic algorithm for the max-mean dispersion problem," *Computers & Operations Research*, vol. 72, pp. 118–127, 2016.

[35] J. Xie, X. Li, L. Gao, and L. Gui, "A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems," *Journal of Manufacturing Systems*, vol. 71, pp. 82–94, 2023.

[36] H. Alazzam, E. Alhenawi, and R. M. H. Al-Sayyed, "A hybrid job scheduling algorithm based on tabu and harmony search algorithms," *The Journal of Supercomputing*, vol. 75, pp. 7994 – 8011, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195246542

[37] M. Alinaghian, E. B. Tirkolaee, Z. K. Dezaki, S. R. Hejazi, and W. Ding, "An augmented tabu search algorithm for the green inventory-routing problem with time windows," *Swarm Evol. Comput.*, vol. 60, p. 100802, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:228914658

[38] K.-C. Tseng, W.-C. Lai, I.-C. Chen, Y.-H. Hsiao, J.-Y. Chiue, and W.-C. Huang, "Amplitude-ensemble quantum-inspired tabu search algorithm for solving 0/1 knapsack problems," in *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2024, pp. 718–725.

[39] J. B. Holliday, E. Osaba, and K. Luu, "An advanced hybrid quantum tabu search approach to vehicle routing problems," *arXiv preprint arXiv:2501.12652*, 2025.

[40] C. Moussa, H. Wang, H. Calandra, T. Bäck, and V. Dunjko, "Tabu-driven quantum neighborhood samplers," in *Evolutionary Computation in Combinatorial Optimization: 21st European Conference, EvoCOP 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 21*. Springer, 2021, pp. 100–119.

[41] D. M. Jaeggi, G. T. Parks, T. Kipouros, and P. J. Clarkson, "The development of a multi-objective tabu search algorithm for continuous optimisation problems," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1192–1212, 2008.

[42] S. Ning, Z. Yuan, Z. Han, and Y. Yang, "An advanced and adaptive tabu search algorithm for dynamic shared parking reservation and allocation," *Stud. Inform. Control*, vol. 31, no. 3, pp. 97–106, 2022.

[43] F. Glover and M. Laguna, *Tabu search.* John Wiley & Sons, Inc., 1993.

[44] Y. Wang, Q. Wu, and F. Glover, "Effective metaheuristic algorithms for the minimum differential dispersion problem," *European Journal of Operational Research*, vol. 258, no. 3, pp. 829–843, 2017.

[45] Q. Wu, Y. Wang, and F. Glover, "Advanced algorithms for bipartite boolean quadratic programs guided by tabu search, strategic oscillation and path relinking," 2017.

[46] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision sciences*, vol. 8, no. 1, pp. 156–166, 1977.

[47] F. Glover and J.-K. Hao, "Diversification-based learning in computing and optimization," *Journal of Heuristics*, vol. 25, no. 4-5, pp. 521–537, 2019.

[48] J. Wang, Y. Zhou, Y. Cai, and J. Yin, "Learnable tabu search guided by estimation of distribution for maximum diversity problems," *Soft Computing*, vol. 16, pp. 711–728, 2012.

[49] R. C. L. Riley and C. Rego, "Intensification, diversification, and learning via relaxation adaptive memory programming: a case study on resource constrained project scheduling," *Journal of Heuristics*, vol. 25, no. 4-5, pp. 793–807, 2019.

[50] D. Schindl and N. Zufferey, "A learning tabu search for a truck allocation problem with linear and nonlinear cost components," *Naval Research Logistics (NRL)*, vol. 62, no. 1, pp. 32–45, 2015.

[51] V. R. Máximo and M. C. Nascimento, "Intensification, learning and diversification in a hybrid metaheuristic: an efficient unification," *Journal of Heuristics*, vol. 25, no. 4-5, pp. 539–564, 2019.

[52] F. Glover, "Tabu search and adaptive memory programming—advances, applications and challenges," *Interfaces in computer science and operations research: Advances in metaheuristics, optimization, and stochastic modeling technologies*, pp. 1–75, 1997.

[53] M. I. Ciarleglio, "Modular abstract self-learning tabu search (masts): Metaheuristic search theory and practice," 2008.

[54] M. A. Tahir, A. Bouridane, and F. Kurugollu, “Simultaneous feature selection and feature weighting using hybrid tabu search/k-nearest neighbor classifier,” *Pattern Recognition Letters*, vol. 28, no. 4, pp. 438–446, 2007.

[55] L. Mousin, L. Jourdan, M.-E. K. Marmion, and C. Dhaenens, “Feature selection using tabu search with learning memory: learning tabu search,” in *International Conference on Learning and Intelligent Optimization.* Springer, 2016, pp. 141–156.

[56] A. C. G. C. Rajan, “Neural based tabu search method for solving unit commitment problem with cooling-banking constraints,” *Serbian Journal of electrical engineering*, vol. 6, no. 1, pp. 57–74, 2009.

[57] K.-M. Lan, U.-P. Wen, H.-S. Shih, and E. S. Lee, “A hybrid neural network approach to bilevel programming problems,” *Applied Mathematics Letters*, vol. 20, no. 8, pp. 880–884, 2007.

[58] N. T. Nguyen and K. Lee, “Deep learning-aided tabu search detection for large mimo systems,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 6, pp. 4262–4275, 2020.

[59] H.-Y. Lu, S. P. Azizi, and S.-C. Cheng, “Deepeigen-tabu: Deep eigen network assisted probabilistic tabu search for massive mimo detection,” *IEEE Transactions on Vehicular Technology*, vol. 73, pp. 13 292–13 308, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:269343144

[60] X. Gu, S. Zhao, and Y. Wang, “Reinforcement learning enhanced multi-neighborhood tabu search for the max-mean dispersion problem,” *Discret. Optim.*, vol. 44, p. 100625, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233926774

[61] Z. Sun, U. Benlic, M. Li, and Q. Wu, “Reinforcement learning based tabu search for the minimum load coloring problem,” *Computers & Operations Research*, vol. 143, p. 105745, 2022.

[62] S. Minton, “Automatically configuring constraint satisfaction programs: A case study,” *Constraints*, vol. 1, no. 1-2, pp. 7–43, 1996.

[63] T. Bartz-Beielstein and M. Preuss, “Experimental research in evolutionary computation,” in *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation.* ACM, 2007, pp. 3001–3020.

[64] M. Oltean, "Evolving evolutionary algorithms using linear genetic programming," *Evolutionary Computation*, vol. 13, no. 3, pp. 387–410, 2005.

[65] M. Birattari, "The problem of tuning metaheuristics as seen from a machine learning perspective," 2004.

[66] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NIPS) 25*, 2012, pp. 2960–2968.

[67] K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, and M. Reimann, *Metaheuristics: Progress in complex systems optimization.* Springer Science & Business Media, 2007, vol. 39.

[68] C. Fonlupt, D. Robilliard, P. Preux, and E.-G. Talbi, "Fitness landscapes and performance of meta-heuristics," in *Meta-Heuristics.* Springer, 1999, pp. 257–268.

[69] P. Merz, "Memetic algorithms for combinatorial optimization problems: fitness landscapes and effective search strategies," 2006.

[70] H. Hoos and T. Stutzle, "Stochastic local search: Foundations and applications morgan kaufmann," *San Francisco*, 2005.

[71] D. Michie, J. Fleming, and J. Oldfield, "A comparison of heuristic, interactive and unaided methods of solving a shortest-route problem," *Machine intelligence*, vol. 3, pp. 245–255, 1968.

[72] P. Krolak, W. Felts, and G. Marble, "A man-machine approach toward solving the traveling salesman problem," *Communications of the ACM*, vol. 14, no. 5, pp. 327–334, 1971.

[73] G. W. Klau, N. Lesh, J. Marks, and M. Mitzenmacher, "Human-guided tabu search," in *AAAI/IAAI*, 2002, pp. 41–47.

[74] M. Kadluczka and P. C. Nelson, "N-to-2-space mapping for visualization of search algorithm performance," in *16th IEEE International Conference on Tools with Artificial Intelligence.* IEEE, 2004, pp. 508–513.

[75] D. Monett-Diaz, "Agent-based configuration of metaheuristic algorithms," Ph.D. dissertation, PhD Thesis. Humboldt University of Berlin, 2004.

[76] C. Audet, W. Hare, C. Audet, and W. Hare, *Introduction: tools and challenges in derivative-free and blackbox optimization.* Springer, 2017.

[77] N. Ploskas, "Parameter tuning of linear programming solvers," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM).* IEEE, 2022, pp. 1–6.

[78] Y. Eryoldaş and A. Durmuşoglu, "A literature survey on offline automatic algorithm configuration," *Applied Sciences*, vol. 12, no. 13, p. 6316, 2022.

[79] F. Mascia, P. Pellegrini, M. Birattari, and T. Stützle, "An analysis of parameter adaptation in reactive tabu search," *International Transactions in Operational Research*, vol. 21, no. 1, pp. 127–152, 2014.

[80] A. Bagis, "Tabu search algorithm based pid controller tuning for desired system specifications," *Journal of the Franklin Institute*, vol. 348, no. 10, pp. 2795–2812, 2011.

[81] C. Lagos, B. Crawford, R. Soto, E. Cabrera, J. Vega, F. Johnson, and F. Paredes, "Improving tabu search performance by means of automatic parameter tuning," *Canadian Journal of Electrical and Computer Engineering*, vol. 39, no. 1, pp. 51–58, 2016.

[82] J. Xu, S. Y. Chiu, and F. Glover, "Fine-tuning a tabu search algorithm with statistical tests," *International Transactions in Operational Research*, vol. 5, no. 3, pp. 233–244, 1998.

[83] C. Audet, C.-K. Dang, and D. Orban, "Optimization of algorithms with OPAL," *Mathematical Programming Computation*, vol. 6, no. 3, pp. 233–254, 2014.

[84] C. Audet, "Tuning Runge-Kutta parameters on a family of ordinary differential equations," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 8, no. 3, pp. 277–286, 2018.

[85] R. Montemanni and D. H. Smith, "Heuristic manipulation, tabu search and frequency assignment," *Computers & operations research*, vol. 37, no. 3, pp. 543–551, 2010.

[86] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem," *INFORMS Journal on Computing*, vol. 24, no. 3, pp. 343–355, 2012.

[87] C. Archetti, G. Guastaroba, and M. G. Speranza, "An ilp-refined tabu search for the directed profitable rural postman problem," *Discrete Applied Mathematics*, vol. 163, pp. 3–16, 2014.

[88] F. Arnold, Í. Santana, K. Sörensen, and T. Vidal, "Pils: Exploring high-order neighborhoods by pattern mining and injection," *Pattern Recognition*, p. 107957, 2021.

[89] R. Battiti and M. Brunato, "The lion way," *Machine Learning plus Intelligent Optimization. LIONlab, University of Trento, Italy*, vol. 94, 2014.

[90] F. Hafiz and A. Abdennour, "Particle swarm algorithm variants for the quadratic assignment problems-a probabilistic learning approach," *Expert Systems with Applications*, vol. 44, pp. 413–431, 2016.

[91] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[92] J. Ceberio, A. Mendiburu, and J. A. Lozano, "The plackett-luce ranking model on permutation-based optimization problems," in *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 494–501.

[93] J. Boyan and A. W. Moore, "Learning evaluation functions to improve optimization by local search," *Journal of Machine Learning Research*, vol. 1, no. Nov, pp. 77–112, 2000.

[94] D. C. Porumbel, J.-K. Hao, and P. Kuntz, "A search space "cartography" for guiding graph coloring heuristics," *Computers & Operations Research*, vol. 37, no. 4, pp. 769–778, 2010.

[95] J.-P. Hamiez and J.-K. Hao, "An analysis of solution properties of the graph coloring problem," in *Metaheuristics: computer decision-making*. Springer, 2003, pp. 325–345.

[96] S. Baluja, A. Barto, K. Boese, J. Boyan, W. Buntine, T. Carson, R. Caruana, D. Cook, S. Davies, T. Dean *et al.*, "Statistical machine learning for large-scale optimization," 2000.

[97] J. Boyan and A. W. Moore, "Learning evaluation functions for global optimization and boolean satisfiability," in *AAAI/IAAI*, 1998, pp. 3–10.

[98] C. Bongiovanni, M. Kaspi, J.-F. Cordeau, and N. Geroliminis, "A predictive large neighborhood search for the dynamic electric autonomous dial-a-ride problem," Tech. Rep., 2020.

[99] S. Thevenin and N. Zufferey, "Learning variable neighborhood search for a scheduling problem with time windows and rejections," *Discrete Applied Mathematics*, vol. 261, pp. 344–353, 2019.

[100] Y. Sun, X. Li, and A. Ernst, “Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems,” *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[101] J. Lauri and S. Dutta, “Fine-grained search space classification for hard enumeration variants of subset problems,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 2314–2321.

[102] B. Abbasi, T. Babaei, Z. Hosseinifard, K. Smith-Miles, and M. Dehghani, “Predicting solutions of large-scale optimization problems via machine learning: A case study in blood supply chain management,” *Computers & Operations Research*, vol. 119, p. 104941, 2020.

[103] M. Fischetti and M. Fraccaro, “Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks,” *Computers & Operations Research*, vol. 106, pp. 289–297, 2019.

[104] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, “Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art,” *European Journal of Operational Research*, 2021.

[105] E.-G. Talbi, “Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics,” 2020.

[106] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[107] Y.-J. Hu, T.-H. Ku, R.-H. Jan, K. Wang, Y.-C. Tseng, and S.-F. Yang, “Decision tree-based learning to predict patient controlled analgesia consumption and readjustment,” *BMC medical informatics and decision making*, vol. 12, no. 1, pp. 1–15, 2012.

[108] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology,” *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[109] I. A. Bikker, N. Kortbeek, R. M. van Os, and R. J. Boucherie, “Reducing access times for radiation treatment by aligning the doctor’s schemes,” *Operations research for health care*, vol. 7, pp. 111–121, 2015.

[110] B. Vieira, E. W. Hans, C. van Vliet-Vroegindeweij, J. van de Kamer, and W. van Harten, "Operations research for resource planning and-use in radiotherapy: a literature review," *BMC medical informatics and decision making*, vol. 16, no. 1, p. 149, 2016.

[111] T. Berthold, "Measuring the impact of primal heuristics," *Operations Research Letters*, vol. 41, no. 6, pp. 611–614, 2013.

[112] J. Kallestad, R. Hasibi, A. Hemmati, and K. Sörensen, "A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems," *European Journal of Operational Research*, vol. 309, no. 1, pp. 446–468, 2023.

[113] C. Audet and J. Dennis, Jr., "Mesh Adaptive Direct Search Algorithms for Constrained Optimization," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 188–217, 2006.

[114] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and Efficient Hyperparameter Optimization at Scale," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1437–1446.

[115] J. Hooker, "Testing heuristics: We have it all wrong," *Journal of heuristics*, vol. 1, pp. 33–42, 1995.

[116] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997. [Online]. Available: https://dx.doi.org/10.1016/S0305-0548(97)00031-2

[117] C. Audet, S. Le Digabel, V. Rochon Montplaisir, and C. Tribes, "Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm," *ACM Transactions on Mathematical Software*, vol. 48, no. 3, pp. 35:1–35:22, 2022.

[118] J. Moré and S. Wild, "Benchmarking Derivative-Free Optimization Algorithms," *SIAM Journal on Optimization*, vol. 20, no. 1, pp. 172–191, 2009.

[119] A. Chmiela, E. Khalil, A. Gleixner, A. Lodi, and S. Pokutta, "Learning to schedule heuristics in branch and bound," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 235–24 246, 2021.

APPENDIX A CONFERENCE ARTICLE. A LEARNING METAHEURISTIC ALGORITHM FOR A SCHEDULING APPLICATION

Nazgol Niroumandrad
Department of Mathematics and Industrial Engineering, Polytechnique Montréal
Nadia Lahrichi
Department of Mathematics and Industrial Engineering, Polytechnique Montréal
Andrea Lodi
Department of Mathematics and Industrial Engineering, Polytechnique Montréal

Abstract Tabu Search is among one of the metaheuristic algorithms that are widely recognized as efficient approaches to solve many combinatorial problems. Studies to improve the performance of metaheuristics have increasingly relied on the use of various methods, either combining different metaheuristics or originating outside of the metaheuristic field.
This paper presents a learning algorithm to improve the performance of tabu search by reducing its search space and the evaluation effort. We study its performance using classification methods in an attempt to select moves through the search space more intelligently. The experimental results demonstrate the benefit of using a learning mechanism under deterministic environment and with uncertainty conditions.

Keywords Learning tabu search; combinatorial problems; logistic regression

History This conference article is published in Metaheuristics. MIC 2022. Lecture Notes in Computer Science, vol 13838. Springer, Cham.

A.1 Introduction

Metaheuristics are popular and demanded optimization methods for many hard optimization problems because they are flexible and can be adapted to complex applications. These algorithms were first introduced by [46] and refer to approximate algorithms that obtain near optimal solutions for combinatorial or integer programming problems.

Metaheuristics generate a lot of dynamic data during the iterative search process. However, they do not use explicit knowledge discovered during the search. The main idea of this paper revolves around using the data generated during the search process to show that metaheuristics, and more specifically tabu search, can behave more intelligently and efficiently. Using advanced machine learning (ML) models can be helpful to extract valuable knowledge that will guide and enhance the search performance to move smarter through the search space. Having a better exploration of the search space and exploring solutions that might not be not reachable without these learning algorithms leads to optimizing computation time in complex problems.

Since tabu search (TS) has been successful in solving many hard optimization problems, we chose to study the effects of learning methods on the performance of a tabu search algorithm. We are proposing a novel learning tabu search algorithm using logistic regression in the context of a physician scheduling problem. To the best of our knowledge, there is no comprehensive study on integrating ML techniques into TS to explore the search space. Most studies on learning metaheuristics evolve in the context of clustering or intensification and diversification strategies.

Although there are a number of studies on improving the performance of TS, the literature lacks a comprehensive study on how ML techniques can be integrated into TS to enhance its search procedure. This paper provides an intensive study on the use of ML techniques in the design of TS, which does not rely on a simple diversification/intensification strategy to improve the search procedure. We believe this paper is beneficial for both academic and industry experts engaged in solving hard combinatorial problems. Our proposed method is used to study a scheduling problem, and our results are compared with those of [1]. Since the original tabu search presented in [1] proved to be very efficient and optimized under deterministic conditions, this paper focuses on improving, by learning, the performance of the TS with uncertainty conditions.

The rest of the paper is organized as follows. In Appendix A.2, we review the related studies. In Appendix A.3, we explain the problem along with the proposed method in detail. We describe the instance generation and present the results in Appendix A.4. A discussion on the results and different testing strategies is presented in Appendix A.5 and Appendix A.6 provides concluding remarks.

A.2 Related literature

The study of learning within metaheuristics has not been given the attention it deserves. However, benefiting from machine learning (ML) techniques to solve combinatorial problems received a lot of attention recently. More precisely, [89] and [90] demonstrated that employing ML during the search process can improve the performance of heuristic algorithms. [93] described algorithms that improve the search performance by learning an evaluation function that predicts the outcome of a local search algorithm from features of states visited during the search. Other studies on learning evaluation functions are presented in [96], [97] and [98]. Along the same subject, [99] is another example of a recent study on learning metaheuristics. This study proposed a learning variable neighborhood search (LVNS) that identifies quality features simultaneously. This information is then used to guide the search towards promising areas of the solution space. The LVNS learning mechanism relies on a set of trails, where the algorithm measures the quality of the solutions.

ML was also employed to prune the search space of large-scale optimization problems by developing pre-processing techniques [100], [101]. Some other studies used ML-based methods to directly predict a high-quality solution [102, 103]. Moreover, [104] and [105] provided a review of studies where metaheuristic algorithms benefited from ML and the potential future work.

Building upon these previous studies, we propose a learning tabu search algorithm enhanced with a logistic regression method to guide the search through the solution space of hard combinatorial problems. We observe that the emphasis on adaptive memory within tabu search represents the nature of its learning mechanism. However, most previous works focused on clustering [46] or intensification/diversification [47] strategies. In metaheuristics, intensification and diversification strategies play important roles in the quality of solutions. In a recent study, [49] presented a relaxation-adaptive memory programming algorithm on a resource-constrained scheduling problem. In that approach, primal-dual relationships help to effectively explore the interplay between intensification, diversification, and learning (IDL). The algorithm is designed to integrate the current most effective Lagrangian-based heuristic with a simple tabu search. The authors aimed to present a study on the IDL relationship when dual information is added to the search. In [50], the authors presented a learning tabu search algorithm for a truck-allocation problem in which they considered a trail system (in the ant colony optimization, a trail system is inspired from the pheromone trails of ants to mark a path) for the combination of important characteristics. In this study, a diversification mechanism is introduced to help visit new solution space regions. The authors proposed diversifying the search by performing “good” moves that were not often performed in the previous cycles.

There are also numerous studies found in the literature on improving the performance of tabu search. In particular, [43] emphasized selecting particular attributes of solutions and determining conditions that help to find the prohibited moves, in order to produce high-quality solutions. Following that work, [44] and [45] employed a balance among more commonly used attributes. The presented computational experiments showed that considering these attributes can significantly outperform all other methods. These outcomes underpin researchers’ ongoing strategy of identifying attributes that lead to more effective methods. However, to the best of our knowledge, our study in favor of learning the characteristics of the search space during the tabu search algorithm’s search procedure is a novel contribution to the literature. We aim to use a classification model to fill this gap. Our contribution focuses on how ML helps to learn the best neighborhoods to build or change a solution during the search process.

A.3 Problem statement and proposed learning algorithm

In a nutshell, tabu search [2] is an iterative procedure that starts from an initial solution x_0 (possibly infeasible). From each current solution x ($x = x_0$ at the beginning of the procedure), it moves to a neighbor solution x'. The neighborhood is defined by all solutions that can be reached from $x \in X$, where X is the solution space, after applying a specific move. Let us denote this neighborhood by $N(x)$. The next solution x' is the best non-tabu solution in the neighborhood $N(x)$ (an exception is made if a move is tabu but it improves upon the current best solution x^*, i.e., through the so-called aspiration criteria). The function $f(x)$ is defined to evaluate each solution. To prevent cycling, a tabu list (T_{list}) containing attributes of recently visited solutions (or attributes of moves) is maintained. The associated solutions cannot be revisited for a specific number of iterations. Various strategies can be applied to search the neighborhood solutions. The moves and strategies to search the solution space are the main ingredients of tabu search methods. Hence, the adequate combination and sequence have a great impact on the quality of the results. Tabu search is used in multiple applications and is adapted to handle uncertainty. This is commonly done by considering several scenarios, typically generated using historical data or a probability distribution. Instead of moving to the best solution in the neighborhood (deterministic environment), the best solution in average (with uncertainty conditions) is preferred.

A naïve approach to choose a solution x' is to evaluate and analyze all possibilities, which would be computationally expensive. Instead, we can characterize the neighborhood using experiments. In particular, the guiding principle of our investigation is that using a learning method can help us find good solutions faster. ML techniques allow us to extract knowledge from good solutions and use it to generate even better solutions. This knowledge can be in the form of a set of rules or patterns [88]. Table A.1 shows how applying a learning method during the search process to reduce the space in a metaheuristic algorithm has computational impact. This table presents an example where we have $|P|$ number of physicians, $|M|$ number of patients and $|K|$ number of time blocks in a physician scheduling problem. However, this can be generalized to other problems.

Table A.1 Examples of computational impact of reduction of search space

Size of the problem	Complexity of the problem **without** learning	Predicting element	Complexity of the problem **with** learning
$\|P\| \times \|M\| \times \|K\|$	$O(n^3)$	$p \in P$	$O(n^2(n-1))$
		$p \in P$ & $m \in M$	$O(n(n-1)^2)$
		$p \in P$ & $m \in M$ & $k \in K$	$O((n-1)^3)$

We studied the impact of deploying a learning procedure to answer this situation. We chose logistic regression for this purpose, as it is one of the most important models that can be applied to analyze categorical data. The learning procedure is employed in two phases of training and application.

- Training phase : this phase is divided in stages T_1 and T_2. In T_1, the original TS collects data related to the structure of the search space. Once enough data is collected, training begins with the logistic regression model for a specific number of iterations (T_2). Training starts at iteration I_{st} and ends at I_{end};
- Application phase : in this phase, an action will be taken based on the prediction of the trained model and after validating the elements of a tabu list (T_{list}).

The learning methods are very sensitive to the input information. Thus, extracting the features that have the greatest impact on the solution space is the first and most important step. These features represent important characteristics of the solutions space and moves. Table A.2 presents these features in different categories.

Let x be the current solution, x^* the best known solution, and x' the next solution. Each solution is represented as a matrix of integers. In the category of cost improvement, we are first considering the improvement in cost, which is presented as $\Delta_x = f(x') - f(x)$. The first

Table A.2 Input/output features for the training model

	Input		Output
Category	Features	Format	Label
Cost improvement	Δ_x	$\mathbb{R}$	Observed Output
	$\Delta_x > 0$	$\mathbb{B}$	
Tabu	$x_i \in T_{list}$	$\mathbb{B}$	
	$\lvert t_i \rvert = T_{list}(x)$	$\mathbb{Z}$	
	$Freq_{-I^D}$	$\mathbb{Z}$	
Solution	x	$M_{\mathbb{Z}}$	
	D^*	$M_{\mathbb{B}}$	
	D'	$M_{\mathbb{B}}$	
Move	Attractiveness	$\mathbb{Q}$	
	Trail of the moves	$M_{\mathbb{Q}}$	

feature is the value of Δ_x and the second reflects if the solution is improved ($\Delta_x > 0$ in the context of maximizing). In the tabu category, we have a binary variable that determines if the accepted move belongs to the T_{list} and whether it has already been visited along with the tabu value for the move, meaning when it will be free and can be considered again. The frequency of the move is also considered, which indicates how many times the move has been visited so far. The next category represents the characteristics of solution. First, the solution x itself. A binary matrix (D^*) denotes the difference between the obtained solution and the last best known solution. A 1 indicates if the corresponding entry is equal, 0 otherwise. Also, a binary matrix (D') denotes the difference between the obtained solution and the previous solution, with the same meaning for 1's and 0's. The final category is related to the move characteristics, i.e., the attractiveness of the move and a trail matrix of the moves. These features were inspired by the recent work of [99]. Here, a trail system influences the decisions made by the ants in the Ant Algorithms, and the notion of move attractiveness shows that moves with high attractiveness values have a higher chance to be performed.

As tabu search can be used in situations with uncertainty conditions, the set of input features under these conditions is modified by considering the average of $f(x)$ over all scenarios at each iteration. In other words, we need to find the move that is the best in average over all the scenarios. Thus, our objective is to find the solution $x' \in N_v(x)$ that minimizes the average solution over all scenarios $\left(\frac{\sum_w f_w(x'_w)}{|W|}\right)$.

The learning tabu search (L-TS) model differs from the original TS mostly in the search space. We seek to reduce the number of evaluations in the search process. Thus, we predict the subset of promising moves and evaluate only those neighbors instead of evaluating every possible neighbor of $N(x)$.

The L-TS algorithm starts with the original TS to collect the necessary data for I_{st}^{D} iterations. We train the learning method for a specific number of iterations (stage T_2), evaluate the result of the learning algorithm, and update the set of input features to encourage the method to search more promising regions. In the application phase, we use the last trained model at iteration $I_{end}^{D} - 1$ to identify (by prediction) promising moves to build $N'(x) \subset N(x)$ (note that superscript D stands for deterministic and S will be used for the case with uncertainty conditions). The total procedure ends when the stopping criterion ($Stop_{max}$) is reached. Our criterion is 1h of CPU time. Details of the parameter initialization step are documented in Appendix A.4.

A.4 Experiments

In this section, we show the advantage of using a learning algorithm during the TS procedure for both deterministic environment and with uncertainty conditions.

We compare the results with a benchmark previously published in [1] for both deterministic environment and an environment with uncertainty in which the performance was compared with CPLEX. We refer to this previous work as "original TS" in the remainder of the paper. In [1], the authors studied a tabu search algorithm in a physician scheduling problem. The goal is to find a weekly cyclic schedule for physicians in a radiotherapy department and to assign the arriving patients to the best possible available specialist for their cancer type. In most radiotherapy centers, the physician schedule is task based. Each day is divided into one or multiple periods, and each period is dedicated to a single task. The goal is to minimize the duration of the pre-treatment phase for patients. This is defined as the time from the patient's arrival day to the day the final task is finished before the treatment starts. Taking inspiration from this work, we use 21 generated pseudo-real instances where we vary the number of new patients arriving each week and the number of available physicians. The number of physicians varies from 6 to 10 and the number of patients from 7 to 60, ranging from small instances to real-world applications. Each instance is labeled $pr-$(# of physicians, # of patients). In the deterministic case, we select one scenario to obtain a typical schedule, and in the situation with uncertainty, we consider a subset of W for different scenarios. We refer the readers to [1] for more details on instance generation.

We report and analyze results in deterministic and uncertain environments. All results were compared and evaluated with respect to the original TS method and a random approach (in which $N'(x) \subset N(x)$ was randomly chosen) to test the performance of the L-TS method. Comparing these three approaches helps us to see the performance of each and confirm the advantage of choosing TS over the random approach and of choosing L-TS over TS. It shows that we are learning during the process, and the results are not achieved by chance.

First, we wish to validate our L-TS algorithm and determine the value of the parameters, i.e., the size of the T_{list}, the number of iterations in each neighborhood, and the iteration to start the training phase and application phase. The values tested are all related to the size of the instances (i.e., number of patients, number of physicians, and number of time blocks). For the deterministic case, we use $I_1^D = 1$, $I_2^D = 2|J| + |I| + |5n|$, $I_3^D = 1$, $I_{st}^D = 3\sqrt{|J| \times |I| \times |D|}$, $I_{end}^D = 2|I_{st}^D|$, $Stop_{max} = 1h$, and we set $\theta^D = 2|J| + |I| + |D|$.

To evaluate the solution obtained from the learning tabu search algorithm under uncertainty conditions, we proceed as follows:

- Generate a set A of scenarios (up to 50 different scenarios);
- For each instance (i.e., pr-(6,7) to pr-(10,60)), run the algorithm using 10 or 50 scenarios from set A (using one scenario is equivalent to the deterministic case).

To evaluate the performance of the learning algorithm with uncertainty conditions, the values of the parameters I_1^S, I_3^S and θ^S are the same, except that I_2^S is now equal to $|J|$, the number of patients, $I_{st}^S = \sqrt{|J| \times |I| \times |D|}$, and $Stop_{max} = 5h$.

Table A.3 compares the cost (same definition as in [1]) values of different methods and the gap columns show improvements with respect to the original TS. In this table, "GAP - Best compares the best solution obtained from ten different runs of each method and represents the improvements from the best solution obtained from the original TS. Conversely, "GAP - Avg" represents the average values from ten different runs. A negative value in the GAP columns indicates that the learning tabu search has improved the solution on average. It can be observed that logistic regression succeeded in slightly improving the cost, by 0.03% on average. We see more improvement in large instances where the algorithm has more flexibility.

It can be observed that the learning method improved tabu search in uncertainty case globally. We can see the most improvement in cases using 10 scenarios, with -1.67% on average for L-TS.

The value of the cost function alone cannot represent the advantage of using each approach. Hence, we measured the primal integral value for all methods to compare the progress of the

Table A.3 Comparing the performance of different methods

	Tests	Deterministic case				Uncertainty case							
		GAP - Best (%)		GAP - Avg (%)		GAP - Best (%)				GAP - Avg (%)			
						10 Scenarios		50 Scenarios		10 Scenarios		50 Scenarios	
		Random	L-TS	Random	L-TS	Random	L-TS	Random	L-TS	Random	L-TS	Random	L-TS
Small	pr-(6,7)	0.5	**-1.0**	2.3	**-0.7**	1.3	**-0.3**	1.0	**-0.7**	1.1	0.2	0.9	**-1.1**
	pr-(6,9)	1.0	0.8	1.7	**-1.4**	0.8	0.0	1.5	0.3	0.9	**-1.2**	0.5	**-2.2**
	pr-(6,11)	2.4	0.3	2.1	**-0.2**	2.2	**-0.3**	1.1	**-1.3**	1.0	**-1.1**	2.6	**-2.6**
	pr-(6,12)	2.9	0.5	3.1	0.3	0.8	**-1.4**	0.8	**-1.6**	0.3	**-1.3**	1.6	1.3
	pr-(8,7)	2.7	**-0.7**	2.4	**-0.1**	1.3	**-0.5**	**-0.4**	**-1.1**	0.4	**-1.0**	2.1	**-1.3**
	pr-(8,9)	2.2	**-0.4**	2.5	**-0.2**	1.7	**-0.4**	1.4	**-0.4**	1.6	**-0.6**	0.7	**-1.4**
	pr-(10,7)	1.5	0.1	2.3	0.1	1.0	**-0.7**	1.0	0.1	1.3	**-0.4**	0.7	**-0.9**
Medium	pr-(6,20)	1.2	0.9	3.6	1.3	0.7	**-1.3**	2.7	0.6	1.6	**-1.4**	0.5	0.6
	pr-(8,11)	3.0	0.6	2.9	0.4	1.0	**-0.6**	0.9	**-0.6**	1.3	**-1.3**	0.3	**-1.6**
	pr-(8,12)	1.5	0.0	2.3	0.0	1.2	**-0.8**	0.0	**-1.5**	0.6	**-0.8**	0.0	**-2.0**
	pr-(8,20)	3.3	0.8	3.7	**-0.1**	2.4	**-0.7**	1.8	**-0.4**	**-1.3**	**-1.4**	1.7	**-2.1**
	pr-(10,9)	2.3	0.3	2.1	0.1	1.5	**-0.1**	1.0	**-0.4**	1.5	**-0.4**	0.3	**-1.6**
	pr-(10,11)	1.7	**-0.3**	2.2	0.2	0.4	**-1.0**	0.9	0.3	1.5	**-1.0**	**-0.4**	**-1.9**
	pr-(10,12)	1.5	0.1	3.0	0.4	1.3	**-0.3**	0.4	**-0.4**	1.2	**-0.3**	0.1	**-2.2**
Large	pr-(6,40)	2.6	**-4.0**	4.5	**-0.4**	5.2	4.6	**-1.3**	**-3.5**	2.2	1.9	4.3	**-0.8**
	pr-(6,60)	9.4	**-0.7**	5.8	**-3.6**	**-10.7**	**-25.0**	3.8	**-1.5**	16.3	**-44.7**	0.1	**-3.2**
	pr-(8,40)	2.3	**-0.5**	4.4	1.7	1.3	**-0.6**	0.8	0.0	2.8	**-0.4**	0.5	**-2.4**
	pr-(8,60)	8.2	1.3	8.5	3.3	0.0	**-3.2**	1.7	**-0.3**	3.3	**-4.8**	0.1	**-3.5**
	pr-(10,20)	3.9	0.2	4.1	0.4	1.1	**-0.7**	1.5	0.0	1.2	**-1.0**	0.2	**-4.6**
	pr-(10,40)	3.4	0.5	4.4	0.1	0.2	**-2.1**	0.9	**-0.4**	0.4	**-1.0**	0.1	**-1.5**
	pr-(10,60)	1.5	0.4	6.2	2.2	2.0	0.3	0.9	0.4	1.5	4.3	2.8	**-1.9**
	Average:	2.81	**-0.03**	3.53	0.18	0.79	**-1.67**	1.07	**-0.59**	1.93	**-2.75**	0.94	**-1.75**

primal bound's convergence towards the best-known solution over the entire solving time. Figure A.1 illustrates this measure for all instances.

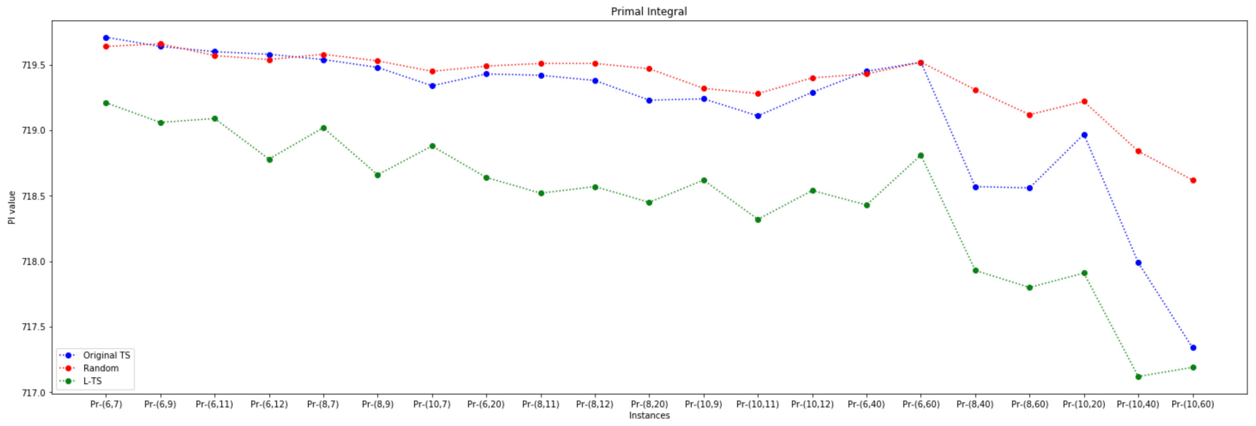


Figure A.1 Comparing the convergence speed for different methods in the deterministic case

The idea of the primal integral [111] is that the smaller the primal integral value is, the better the expected quality of the solution will be if we stop the solver at an arbitrary point in time. It can be observed from Figure A.1 that the logistic regression method has better primal integral values than the other methods for all instances. Again, this figure shows that the learning method, presented by the green line, performs better than the original TS and Random algorithms. Figure A.2 compares the number of evaluations at each iteration, the total number of evaluations, and the total number of iterations until we reach the stopping criterion for different methods. It clearly shows a decrease in the number of calculations in the L-TS. We can also observe that the random method performs same number of evaluations but requires more iterations (compared with L-TS) to find the solution, due to its poor performance. This behavior was expected from the random approach since it is evaluating $N'(x)$, a subset of $N(x)$, randomly.

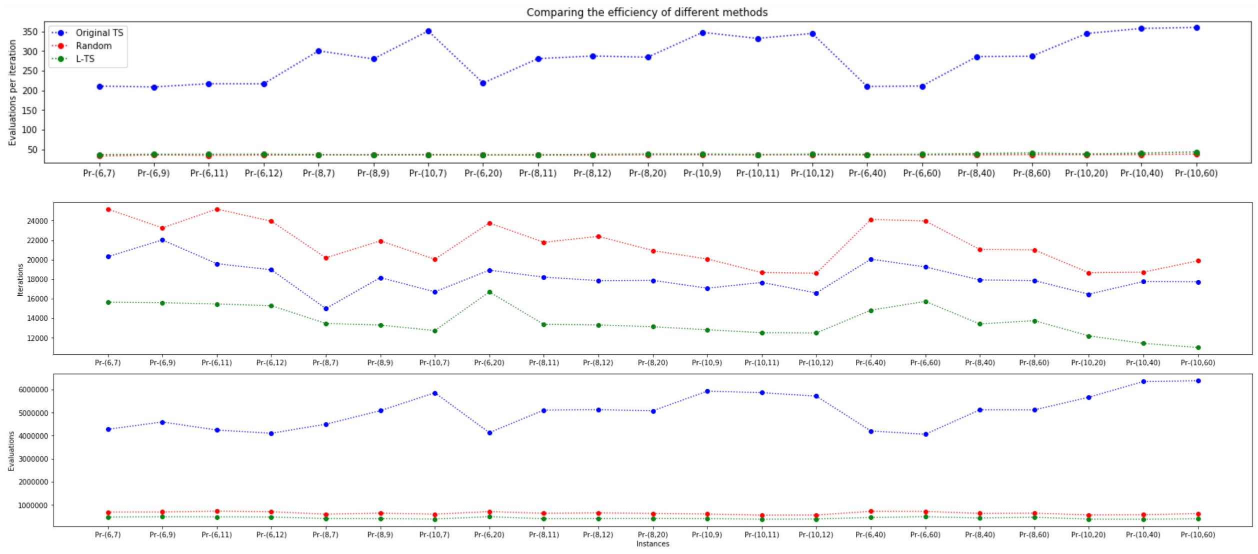
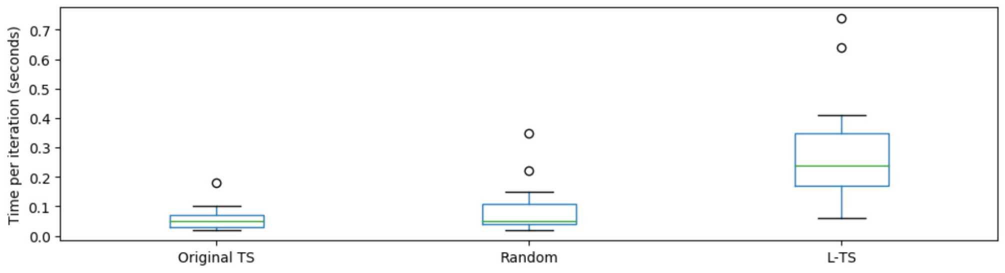


Figure A.2 Comparing computing performance for all methods in the deterministic case

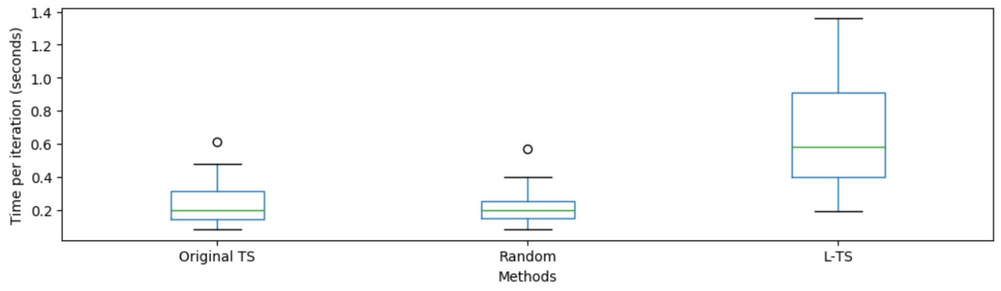
All of these illustrations, including the improvement in the objective value presented in Table A.3, show the advantage of employing the learning TS idea. More precisely, given the similar computing time and number of iterations to reach the best solution, as well as the improvement in the cost value, the logistic regression method demonstrates superior performance.

The performance of the learning algorithm in an uncertain environment was also evaluated based on its computing time. Figure A.3 compares the time per iteration by method for 10 and 50 scenarios. It is clear that increasing the number of scenarios increases the computing time.

We observe that with 10 scenarios, the average gap improved by 1.67% with the logistic regression model. Also, with 50 scenarios, we have 0.59% improvements in average gap for L-TS model.



(a) 10 Scenarios



(b) 50 Scenarios

Figure A.3 Comparing computing time per iteration for all methods with uncertainty

A.5 Discussion

The proposed L-TS algorithm show improvements in both deterministic and uncertain environments. The performance of this algorithm was evaluated with extensive number of experiments and this paper presents part of these experiments to demonstrate the advantage of employing a learning mechanism within TS.

This study was initially started by applying a neural network to predict the promising neighbors. However, the method was time consuming and might not be applicable to large combinatorial and real case problems. Sometimes, a simpler method is able to achieve same results with less computational efforts. Hence, we focused on classification methods, decision trees and logistic regression. The performance of these methods also evaluated through different configurations; in most cases, the logistic regression model outperforms the other ones.

Additionally, in the case with uncertainty conditions, we performed tests where we varied the number of scenarios from 10 to 50. We observed no significant impact with 30 scenarios. We also performed a sensitivity analysis to set the parameters, e.g. $Stop_{max}$ = 2.5, 5, 7.5 and 10 hours.

A.6 Conclusion

In this paper, we proposed a learning tabu search method and studied its performance on a physician scheduling problem. The performance of the proposed algorithm was evaluated using the benchmark instances.

We evaluated the new method in both deterministic and uncertain environments. We showed that our method is very efficient compared with the original tabu search and a random method, especially in the case with uncertainty conditions where the algorithm has more flexibility. Over 21 instances, the average gap improved by 1.67% with logistic regression in the case with 10 scenarios. The learning method obtained best solutions faster than the original TS and random methods over the computing time. Although we studied the application of this method in a scheduling problem, tabu search has already been used to solve pretty much all optimization problems. Thus, the learning tabu search algorithm can be adapted to other applications. Future work could employ learning tabu search to solve other optimization problems by generalizing several ingredients for which we gave special attention to our specific application.

APPENDIX B SUPPORTING INFORMATION FOR CHAPTER 4

B.1 List of symbols

ML	machine learning
TS	tabu search
The TS	the tabu search method presented in [1]
X	the solution space
x_0	the initial solution
x^*	the best solution
x'	best next available solution
$N(x)$	neighborhood of solution x
$f(x)$	the objective value of solution x
Δ_x	the improvement in cost at solution x
T_{list}	the tabu list
L-TS	the learning tabu search
DT	decision trees
LR	logistic regression
$T_{collect}$	phase for collecting data
T_{train}	phase for training the model
T_{apply}	phase for applying the prediction
I_{train}	the iteration that training phase starts
I_{apply}	the iteration that application phase starts
M^*	a binary matrix denotes the difference between the obtained solution and the last best known solution.
M'	a binary matrix denotes the difference between the obtained solution and the previous solution
$Stop_{max}$	the stopping criterion
I	set of physicians
J	set of patients
D	set of days
δ	number of time blocks
R	set of scenarios in the stochastic case
det	deterministic
$stoc$	stochastic

B.2 Primal integral

The primal integral can briefly described as the area underneath the primal gap function p up to time T [119]. In formal context, when x represents a feasible solution and x^* stands as the optimal or best-known solution for the mixed-integer programming problem (P_{MIP}), we define the primal gap of x as follows:

$$\gamma(x) = \begin{cases} 0 & \text{if} \quad |c^T x| = |c^T x^*|, \\ 1 & \text{if} \quad c^T x \cdot c^T x^* < 0, \\ \frac{|c^T x - c^T x^*|}{\max\{|c^T x|, |c^T x^*|\}} & \quad \text{otherwise} \end{cases}$$

Using x^t to represent the current incumbent at time t, we define the primal gap function, denoted as $p : R \geq 0 \rightarrow [0, 1]$ as follows:

$$p(t) = \begin{cases} 1 & \text{if no incumbent is found until time } t \\ \gamma(x^t) & \text{otherwise} \end{cases}$$

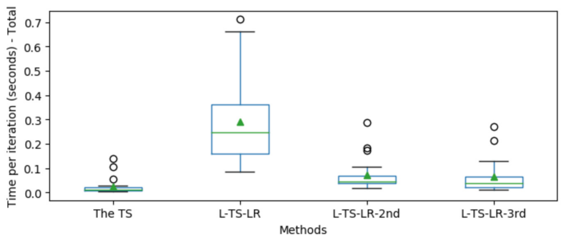
Given a time limit $T \in \mathbb{R} \geq 0$, the primal integral $P(T)$ can be determined by calculating the area under the primal gap function p up to time T,

$$P(T) := \sum_{i=1}^{K} p(t_{i-1})(t_i - t_{i-1}) \tag{B.1}$$

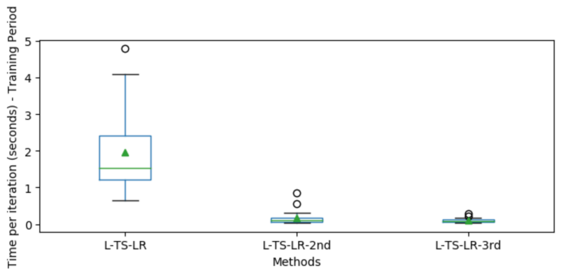
B.3 Computation time

Computation time can vary significantly based on several factors, including hardware specifications and the chosen implementation approach [112]. To highlight this variability, we employed three distinct implementations of the L-TS-LR method, aiming to demonstrate its impact on computation time. Figure B.1 highlights the impact of the implementation approach on computation time. In this figure, we have three variations: L-TS-LR, which represents our primary approach utilizing the C++ package of the OpenCV/3.4.3 library; L-TS-LR-2nd, a complete C++ implementation; and L-TS-LR-3rd, which utilizes the Eigen library for our L-TS-LR method.

In Figure B.1a, a distinct reduction in the time required per iteration to achieve the best solution is evident, particularly with the second implementation. While L-TS-LR-2nd does exhibit slightly longer execution times compared to the TS method, this disparity is minimal, measuring less than 0.05 seconds. This marginal difference can be associated to the data



(a) Time (seconds) per iterations to reach the best solution - Total



(b) Time (seconds) per iterations to reach the best solution - Training Period

Figure B.1 Comparing computing time per iteration for different implementation approach in the deterministic case with online learning

preparation and feature calculation processes related to the learning phase. It's worth noting that the primary contrast in computation time between the three learning implementation approaches becomes more apparent during the training phase, as illustrated in Figure B.1b. The average time per iteration to reach the best solution decreases from 0.29 to 0.14 seconds when using L-TS-LR-2nd, and from 0.29 to 0.065 seconds when using L-TS-LR-3rd. Note that each of these machine learning libraries and packages employ different sets of parameters. This makes the parameter tuning difficult to achieve and need to be done carefully for each method.

APPENDIX C ADDITIONAL NOTES ON CHAPTER 4

This appendix provides additional clarification regarding the prediction and evaluation process used in Chapter 4, particularly concerning the predictive model and its integration within the TS framework.

C.1 Integration of Learning Algorithms with Tabu Search

The overall process was divided into three distinct phases: $I_{collect}$, I_{train} and I_{apply}.

During the I_{train} phase, the predictive model was trained to learn the mapping between problem features and the most suitable physician. This involved generating labeled training data using full evaluations, where the performance of all physicians was assessed for each problem instance. The resulting model aimed to generalize this mapping and provide predictions on unseen instances.

In the I_{apply} phase, the trained model was used to predict the most appropriate physician. Rather than evaluating all possible physicians, which would be computationally expensive, the system evaluated only a limited set of candidates: selected neighbors using the predicted physician only.

This selective evaluation significantly reduced computational cost while still maintaining solution quality. The rationale was that even if the model's prediction was not exactly correct, it would often be close to a good solution, and high-performing candidates could still be found in its immediate neighborhood. The TS then proceeded from this localized starting point, continuing the search within this reduced search space.

This integration of predictive modeling and TS allowed for efficient and effective exploration, keeping a balance between performance and resource usage.

C.2 Model Accuracy

In the paper, we are not reporting the output of the learning methods or analyzing the accuracy of its result. We used the prediction results inside of the TS procedure.

The performance or the overfitting issues of the learning method are not a concern as the main purpose of the method is to extract a knowledge about the search-space behavior rather than to maximize classifier precision. In general, a local search algorithm explores different neighborhoods and the learning algorithm could perform poorly (with low accuracy rate). In this work, we define accuracy rate as the proportion of moves within the search for which the model's top prediction coincides with the physician that ultimately yields the best solution for that move. Therefore, a low (high) accuracy rate indicates that the model's predictions rarely (often) match the best solution. In a local search, what ultimately matters is the quality of the final solution, not the cost of the immediate neighborhood. Even when the model directs the search to a suboptimal neighborhood, it still provides useful information about where high-quality solutions may lie, and by narrowing the search focus, it can reduce the total number of evaluations required. Thus, guiding the TS with a learning-based prediction; even an imperfect one; can accelerate convergence and improve our understanding of how the search space is structured.

In what follows, we present our analysis on the accuracy of L-TS-LR method for some instances. We compared the results obtained from the learning method with the results selected from local search at each neighborhood.

Tests	Accuracy
pr-(10,60)	70%
pr-(8,20)	65%
pr-(6,40)	61%
pr-(4,12)	52%

It shows that for large instances the classification method performs better, and this is due to more available data for learning.

C.3 Evaluation of Alternative Learning Methods

To assess the robustness and flexibility of the proposed framework, alternative machine learning methods were also evaluated during the I_{train} phase. Specifically, two additional models were implemented and tested: $k-$Nearest Neighbors ($k-$NN) and Neural Networks.

The purpose of this comparison was to determine whether different learning algorithms could provide meaningful improvements when combined with the TS phases.

The $k-$NN algorithm was tested as a simple non-parametric method. While $k-$NN benefited from interpretability and required minimal training, its performance was sensitive to the choice of distance metric and the value of k. In practice, it tended to be less consistent than logistic regression and decision tree models in cases involving high-dimensional or noisy features.

Neural Networks were also tested as a more expressive and data-driven approach. Their capacity to capture non-linear relationships between features and outcomes allowed them to perform competitively on more complex instances. However, they also introduced higher complexity and CPU time for training and required careful tuning of hyperparameters, such as network depth, learning rate, and regularization.

Despite some variation in raw prediction accuracy across the models, their impact on overall solution quality during the I_{apply} phase was more complex. Since the TS procedure explores the neighborhood of the predicted physician, small differences in prediction precision did not necessarily lead to significant differences in the final outcome. In many cases, even when the top prediction varied between models, the neighborhood still contained high-performing candidates.

These observations highlight the flexibility of the framework with respect to the choice of predictive model. While logistic regression and decision trees were used in the main experiments for their favorable balance of accuracy, interpretability, and efficiency, alternative models such as $k-$NN and Neural Networks were also evaluated. Neural Networks, while capable of capturing complex relationships, proved to be computationally intensive and less practical in scenarios with limited time or resources.

Overall, the results suggest that the framework can accommodate a range of learning models, particularly when combined with a robust local search component that mitigates small inaccuracies in prediction.