Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

COPPE UFRJ

HYPER-HEURISTICS FOR THE TIME-DEPENDENT ATSP VARIANTS APPLIED TO AIR TRAVEL

Matheus Cunha Simões

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Laura Silvia Bahiense da Silva Leite
Celina Miraglia Herrera de Figueiredo

Rio de Janeiro
Julho de 2023

HYPER-HEURISTICS FOR THE TIME-DEPENDENT ATSP VARIANTS APPLIED TO AIR TRAVEL

Matheus Cunha Simões

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2023

Simões, Matheus Cunha

Hyper-heuristics for the time-dependent ATSP variants applied to air travel/Matheus Cunha Simões. – Rio de Janeiro: UFRJ/COPPE, 2023.

XII, 40 p.: il.; 29, 7cm.

Orientadores: Laura Silvia Bahiense da Silva Leite
Celina Miraglia Herrera de Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2023.

Referências Bibliográficas: p. 37 – 40.

1. Hyper-heuristics. 2. Path Relinking. 3. ATSP. I. Leite, Laura Silvia Bahiense da Silva *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais por apoiarem
minha educação e formação.

Agradecimentos

Aos meu pais, André e Vanessa, e minha irmã, obrigado pelo apoio contínuo durante minha formação acadêmica.

Às minhas orientadoras, Laura Bahiense e Celina Figueiredo, obrigado pela parceria. A dedicação e acompanhamento de vocês foi excencial para o desenvolvimento desta dissertação.

Ao Glaydston Ribeiro, Priscila Machado e Pedro Henrique pela participação e comentários no exame de qualificação.

Aos professores Glaydston Ribeiro e Daniel Ratton por aceitarem fazer parte da banca de avaliação desta dissertação.

A todos os professores que tive o prazer de conhecer durante a pós-graduação, agradeço pela dedicação e ensinamentos em todas as aulas.

Por fim, agradeço à CAPES e ao CNPq pelo apoio financeiro que foi essencial para realização deste trabalho.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

HIPER-HEURÍSTICAS PARA VARIANTES DO PCVA DEPENDENTE DO TEMPO APLICADAS A VIAGENS AÉREAS

Matheus Cunha Simões

Julho/2023

Orientadores: Laura Silvia Bahiense da Silva Leite
Celina Miraglia Herrera de Figueiredo

Programa: Engenharia de Sistemas e Computação

Hiper-heurística é um método de busca, que pode incluir um mecanismo de aprendizado, para selecionar ou gerar heurísticas de baixo nível para resolver problemas computacionais difíceis. Uma particularidade deste método é que ele opera em um espaço de busca de heurísticas ao invés de diretamente no espaço de busca de soluções. É parcialmente motivado pela ideia de desenvolver uma metodologia de busca mais geral que não exija muito conhecimento prévio sobre o problema específico e suas instâncias.

Este trabalho explora o uso da Hiper-heurística perturbativa de seleção em duas variantes do problema do Caixeiro-Viajante Assimétrico Dependente do Tempo no contexto de viagens aéreas. Na primeira, heurísticas utilizadas em algoritmos de busca local apresentados na literatura são incorporadas à hiper-heurística para mostrar sua eficiência. Na segunda, as particularidades e a dificuldade do problema impactaram o desempenho da Hiper-heurística simples e sua hibridização com o Simulated Annealing, levando a propostas de modificação do método. Um critério de aceitação adaptado do "melhor ou igual" e a adição de Path Relinking mostraram-se capazes de gerar soluções de boa qualidade.

Duas soluções iniciais foram geradas para cada instância a partir de diferentes heurísticas construtivas com comportamento complementar e várias estratégias de seleção de heurísticas de baixo nível foram implementadas, como Simple Random, Random Descendent, Random Permutation e Reinforcement Learning. Melhores custos foram encontrados na maioria das instâncias para ambos os problemas em pelo menos uma das estratégias de seleção da hiper-heurística quando comparado a outros métodos da literatura.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

HYPER-HEURISTICS FOR THE TIME-DEPENDENT ATSP VARIANTS APPLIED TO AIR TRAVEL

Matheus Cunha Simões

July/2023

Advisors: Laura Silvia Bahiense da Silva Leite
Celina Miraglia Herrera de Figueiredo

Department: Systems Engineering and Computer Science

Hyper-heuristic is a search method, that may include a learning mechanism, for selecting or generating low level heuristics to solve computational hard problems. A particularity of this method is that it operates on a search space of heuristics instead of directly on the search space of solutions. It is partially motivated by the idea of developing a more generally applicable search methodology that does not require much prior knowledge about the specific problem and its instances.

This work explore the use of the selection perturbative Hyper-heuristic framework in two Time-dependent Asymmetric Traveling Salesman problem variants in the context of air travel. In the first, heuristics used in local search algorithms presented in the literature are embedded into the hyper-heuristic framework to show its efficiency. In the second, particularities and the difficulty of the problem impacted the performance of the simple Hyper-heuristic and its hybridization with Simulated Annealing, leading to proposals for modifying the method. An adapted acceptance criterion of the "improve or equal" and the addition of Path Relinking proved capable of generating good quality solutions.

Two initial solutions were generated for each instance from different constructive heuristics with complementary behavior and several low level heuristic selection strategies were implemented, such as Simple Random, Random Descendent, Random Permutation and Reinforcement Learning. Better solution costs were found in most of the instances for both problems in at least one of the hyper-heuristic selection strategies when compared to other methods in the literature.

Contents

List of Figures

List of Tables

List of Abbreviations

ACO	Ant Colony Optimization, p. 1
ATSP	Asymmetric Traveling Salesman Problem, p. 3
BN	Bellmore and Nemhauser heuristic, p. 11
CI	Cheapest Insertion heuristic, p. 11
GTD-ATSP-TW	Generalised Time-dependent Asymmetric Traveling Salesman Problem with Time Windows, p. 2
GTSP	Generalised TSP, p. 5
LLH	Low level heuristics, p. 17
NN	Nearest-Neighbor heuristic, p. 1
PSO	Particle Swarm Optimization, p. 1
RD	Random Descent, p. 14
RL	Reinforcement Learning, p. 2
RPD	Random Permutation Descent, p. 15
RP	Random Permutation, p. 15
SA	Simulated Annealing, p. 1
SR	Simple Random, p. 14
TD-ATSP-TWPC	Time-dependent ATSP with Time Windows and Precedence Constraints, p. 4
TSC	Travelling Salesman Challenge 2.0, p. 2
TSP-PC	TSP with Precedence Constraints, p. 5
TSP-TW	TSP with Time Windows, p. 5
TSP	Traveling Salesman Problem, p. 1

Chapter 1

Introduction

The Traveling Salesman Problem (TSP) is a well-known optimization $\mathcal{NP}$-hard problem with many real-world applications. Given its complexity, it is quite common for exact branch-and-bound-based algorithms to face difficulties in solving large and complex instances. So, heuristics and metaheuristics emerge as an alternative to find solutions close to optimality in reasonable computational time [3]. In fact, there has been much research in the literature [4–6] exploring the development of heuristic methods to produce good solutions for the TSP and its variants within low computational costs. These heuristics are commonly introduced in metaheuristic algorithms to improve the final result.

Among the applications of the TSP in Air Travel, there is the air flight connection problem, in which given a set of airports in different cities, aim to obtain the route that visits each city with the lowest possible cost, usually the sum of costs to travel from every adjacent vertex in the tour. Marques et al. [7] developed a complete application based on different heuristics and metaheuristic optimization algorithms. They compared the implementation of Simulated Annealing (SA), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) with two variations of the Nearest Neighbor heuristic (NN), addressing the total price of the trip, total duration and a balanced cost corresponding to a weighted sum between the price and duration. Finally, they compared the performance of the method with a commercial flight search application denoted as Nomad [7] launched by Kiwi.com and made publicly available for the resolution of the Flying Tourist Problem, a formulation related to the Time-dependent TSP.

In 2017, the online travel agency Kiwi.com released the Traveling Salesman Challenge [8] - given a set of cities to visit and a list of flights with the respective origin, destination, price and day, the challenge consisted of finding the combination of flights with the lowest price that visits each city once and returns to the first city. In this case, instances had from 15 to 100 vertices and execution time limited in 30 seconds. This problem can be modeled as a Time-dependent Asymmetric Traveling

Salesman Problem (TD-ATSP). Duque et al. [9] developed a hybrid metaheuristic combining ACO and SA to solve the challenge instances that outperformed the solutions provided by an greedy heuristic and these algorithms separately.

In 2018, Kiwi.com proposed the Travelling Salesman Challenge 2.0 (TSC) [2], using the same structure of the first competition, but considering a more complex problem. The contestants had to determine the cheapest possible connection between specific areas, with an area being a set of cities. Besides, the trip had to start from a given city, it was necessary to visit exactly one city in each area, it was obligatory to move between areas every day, and the trip had to continue from the same city. Lastly, the entire trip had to end in a city located within the starting area. This problem was modeled as a Generalised Time-dependent Asymmetric Traveling Salesman Problem with Time Windows (GTD-ATSP-TW) [10]. The challenge included small and large instances that are more specific and complex, with up tp 300 areas and vertices, and execution time limit of 15 seconds. In particular, the large instances are difficult to solve via exact methods considering the time limit set by the challenge. Ahmad et al. [11] developed a hybrid heuristic combining Tabu Search and Simulated Annealing to solve the problem, showing that their approach could improve the initial solutions quite a bit and final results were better than the Great Deluge algorithm [12].

Pylyavskyy et al. [10] proposed a Reinforcement Learning (RL) hyper-heuristic algorithm to solve the instances introduced in TSC [2]. They tested six selection hyper-heuristics that controlled a set of four low level heuristics to improve the initial solutions. Their work was the evolution of Alrasheed et al. [13], where some of the same authors used a local search algorithm to improve the initial solutions in the same problem. A similar approach was used by Saradatta and Pongchairerks [1] to solve a different variation of TSP known as the Time-dependent Asymmetric Traveling Salesman Problem with Time Windows and Precedence Constraints. They generated random initial solutions that were improved by two local search algorithms and compared with a modified Nearest Neighbor algorithm [4] to solve a real-world application of air transport much more complicated than the classical asymmetric traveling salesman problem due to the properties of the airfare prices, the time windows constraints and the precedence constraints.

Regarding the Time-dependent TSP with time windows and precedence constraints, although it is common to find papers addressing the separate variations of this problem (Time-dependent TSP, TSP with time windows and TSP with precedence constraint), few papers explore all these variants in the same problem. Specially, there are no article exploring the use of hyper-heuristics for this problem, until the paper submitted from this work [14], i.e., to the best of our knowledge, this would be the first article to present a hyper-heuristic for it.

1.1 Contributions

In this work is used simple generation and selection low-level heuristics well known in the literature to develop a new hyper-heuristic framework capable of solving, in a better way, the problems solved by [1] and [10]. In addition, the computational results were improved compared to various methods presented in the literature for these problems [1, 10, 11, 13].

For the Generalized time-dependent ATSP with time windows, we used different constructive heuristics to quickly generate good initial solutions and feed our hyper-heuristic with the modified improve or equal criterion to speed up the search and improve the cost of the final result when compared to the methods used in [11], [13], and [10]. This use of different constructive heuristics combined with the characteristics of our acceptance criterion was crucial to find better solutions in the end, since these heuristics presented distinct and complementary behaviors in relation to the tested instances. By using this strategy for the GTD-ATSP-TW, in the vast majority of cases, we were able to improve the best results known in the literature for instances of the problem from the Kiwi competition.

It is common to find the use of Path Relinking methods within a metaheuristic algorithm or as a post-optimization technique operating on the set of solutions. However, although we commonly find in the literature the use of hyper-heuristics with metaheuristics, there are not many examples of using Path Relinking within a hyper-heuristic framework. For instance, Jiang et al. [15] introduced the combination of hyper-heuristics with GRASP and Path Relinking to solve the nurse rostering problem, comparing their computational results with a more traditional procedure based on hyper-heuristic with simulated annealing. So, it is proposed a new path relinking within a hyper-heuristic framework to solve the GTD-ATSP-TW from TSC by Kiwi.com. The Path Relinking connect the local optimal solutions found during the search. As a result, new good solutions are built and more choices may be returned to the final user without much additional time. Finally, the outcome showed that the adequate use of a hyper-heuristic framework with Path Relinking can be very efficient in improving the final results and combining the outcome of every method, new better solutions were set up for the majority of the instances.

1.2 Structure

The following chapters are organized as follows:

- Chapter 2 presents the literature review on Asymmetric Traveling Salesman Problem (ATSP) variants, highlighting the two problems considered in this work: the Time-dependent ATSP with Time Windows and Precedence Constraints (TD-ATSP-TWPC) and the Generalized Time-dependent ATSP with Time Windows (GTD-ATSP-TW).

- In Chapter 3 is introduced the generation and selection low level heuristics used, followed by the hyper-heuristic framework developed and its hybridizations with Simulated Annealing and Path Relinking.

- In Chapter 4 is present all computational experiments in chronological and result improving order. The results on the TD-ATSP-TWPC and on the GTD-ATSP-TW are also compared to the best solutions available in the literature.

- Chapter 5 shows the conclusions and commentaries on another methodology being developed and placed as future work.

Chapter 2

Related work

The TSP optimization is $\mathcal{NP}$-hard problem with various applications in real-world scenarios. There are also several variants of the TSP in the literature, depending on each application that is modeled by this very important problem. Two cases based on multiple of these variants are studied in this work with instances and results found in the literature. Each of theses cases include some of the following characteristics:

- Asymmetric TSP (ATSP): The cost of traveling from city i to city j may not be the same as traveling from j to i;

- Time-dependent TSP (TD-TSP): The travel cost depends on the distance and the day of travel;

- Generalised TSP (GTSP): The cities are divided into clusters and the salesman visits exactly one city of each cluster;

- Open tour TSP: The salesman does not have to end the tour from the point it started.

- TSP with Time Windows (TSP-TW): The salesman must visit a city within a specified time window; and

- TSP with Precedence Constraints (TSP-PC): The salesman must visit city i immediately after visiting city j, for some pairs of cities (i, j).

There is an extensive literature on heuristics for the TSP [4, 5] that can be adapted for variants and used within more complex methods such as metaheuristics and hyper-heuristics. Most of these heuristics are fast and easy to implement, which are good characteristics for low level heuristics. A presentation by Kheiri and Keedwell [16] show the Selection Hyper-heuristics and some examples of how to apply low level heuristics in an iteration of the search for the optimal solution.

The next sections introduce the two problems consider in this work, based on the above-mentioned TSP variants.

2.1 Time-dependent ATSP with Time Windows and Precedence Constraints

The Time-dependent ATSP with Time Windows and Precedence Constraints (TD-ATSP-TWPC) consists of a salesman that has to visit N countries within N weeks, exactly one country each week, and the last visited country must be the same as the starting one, making a cycle, known as Hamiltonian cycle. Let c_{ijk} be the lowest flight price offered by all available airlines to travel from country i to country j in week k, where $i \neq j$. In addition, c_{ijk} is possibly different than c_{jik}, producing the asymmetry in the TSP and c_{ija} may be different from c_{ijb} because the price of a flight between two countries may vary on the week, making this problem a Time-dependent ATSP. It is also possible that there are no available flights to travel from country i to country j in week k and, in this case, a high artificial cost $\tilde{c}_{ijk}$ is associated.

Moreover, as defined in [1], this problem has time windows constraints (a country with a time window constraint must be visited in an exact pre-assigned week, and not the usual time window of an interval potentially wider than 1 day) and precedence constraints (if there is a precedence constraint concerning countries i and j, country j must by visited immediately after country i). The objective of TD-ATSP-TWPC is to minimize the total cost of traveling around all N countries, defined as the sum of the prices of all flights belonging to the selected trips.

The specific characteristics of this problem compared to the classic TSP make the optimization even more difficult and the use of well-defined exact methods almost impossible, considering the acceptable time limit for real-world applications. Saradatta and Pongchairerks [1] proposed a method which generates random initial solutions that are improved by two local search algorithms and compared them with a modified nearest neighbor algorithm to solve the TD-ATSP-TWPC.

Figure 2.1 illustrates an example of a tour based on a real instance, and containing two precedence constraints in which it is necessary to visit China immediately before the USA, and South Korea immediately before Malaysia. As can be seen in Figure 2.1, these precedence constraints make the solutions have very long arcs, with high costs. This specificity opposes the traditional TSP idea of minimizing costs and using short arcs to visit nearby vertices before distant vertices.

There are several studies in the literature about the separate characteristics of this TSP variant including exact and search optimization approaches, such as [17, 18] for the TD-ATSP, [19] for the TSP-TW and [20] for the TSP-PC. However, few papers explore these characteristics all together in the same problem. In particular, no articles was found exploring the use of hyper-heuristics for this problem until the submission of [14]. Regarding this problem, the algorithms used by Saradatta and

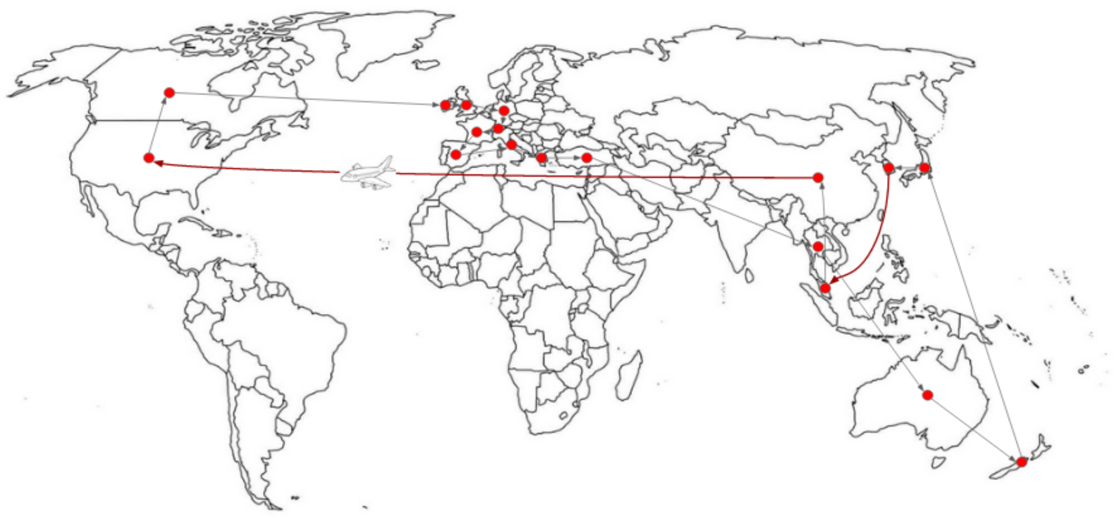


Figure 2.1: Illustration of a tour based on a real instance and containing precedence constraints.

Pongchairerks [1] were implemented and embedded into a hyper-heuristic framework, to show that the combined use of heuristics can produce better solutions than their use separately.

2.2 Generalized Time-dependent ATSP with Time Windows

The flight network growth and the complexity of travel planning considering the different paths and costs to fly from one location to another [21] made several travel agencies to develop their own search engines. Moreover, it motivated some agencies as Kiwi.com to launch online challenges for researchers and developers in order to improve their own software, named Nomad. The Travelling Salesman Challenge was released in 2017 [8] based on a more traditional Time-dependent TSP with instances up to 100 vertices. In 2018, Kiwi.com proposed the Travelling Salesman Challenge 2.0 (TSC) [2].

The TSC is a more complex variant of the TSP with instances up to 300 areas and vertices, each instance having distinct rates of vertices per areas and average adjacency matrix density. The problem is defined by a list of N areas, a list of M cities/airports per area, the traveling costs between the cities/airports per day, and the starting city. The objective of the real-world application is to find the cheapest trip that visits exactly one city of each given area and ends at the starting area. During the trip, it is not possible to arrive to an airport and then continue the trip by departing from another airport of the same area. The final destination of the trip is the starting area but not necessarily the starting airport, so the returned trip does not necessarily define a Hamiltonian cycle. Figure 2.2 shows a fictional example of instance for the problem including a world tour where each area correspond to a major region of the globe.

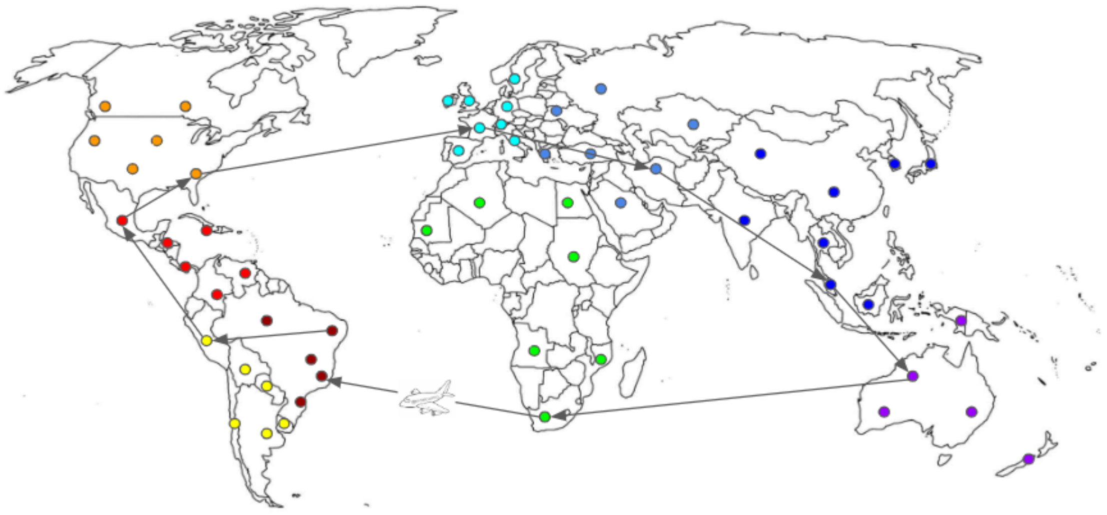


Figure 2.2: Example of an instance of the GTD-ATSP-TW showing a feasible solution that starts and ends in the same area over Brazil.

Pylyavskyy et al. [10] defined this problem as a Generalized Time-dependent ATSP with Time Windows combining the time-dependent with time-windows, asymmetric and generalised variants of the TSP. Different from the problem defined in Subsection 2.1, there are no precedence constraints, and the time windows constraints may have another interpretation. However, there exists the same time dependence based on the cost of traveling between two cities in different days and the same asymmetry on the cost of going from i to j or from j to i. Combined with the characteristics of the Generalised TSP and the increase in the instances size, this problem can be considered harder than the version from 2017 and the one defined in the previous subsection.

A hybrid metaheuristic based on Tabu Search and Simulated Annealing was proposed by Ahmad et al. [11] to solve the TSC [2]. They showed that this approach could improve the initial solutions quite a bit and better the final results compared to the Great Deluge algorithm [12]. Alrasheed et at. [13] used a Local Search algorithm with four perturbative heuristics and were able to find solution in only 12 of the 14 instances from the challenge. Finally, a hyper-heuristic algorithm was proposed by Pylyavskyy et al. [10], in colaboration with some of the same author from Alrasheed et at. [13] work, which improved the results to the best known in the literature. They used a random initial solutions improved by a local search procedure to somehow ensure its feasibilty and input into a Hyper-heuristic with Reinforcement Learning selection strategy.

In this work, the low level heuristics of Pylyavskyy et al. [10] were embedded into a Hyper-heuristic with modified improve or equal acceptance criteria and a new hyper-heuristic framework with Path Relinking [22], that produce better initial solutions based on constructive heuristics to find better final solutions. The results were compared with the three works mentioned in the previous paragraph and we show the impact of the initial solution on the performance of the method through constructive heuristics with complementary characteristics.

Chapter 3

Heuristics

This chapter introduces the heuristics implemented within a Hyper-heuristic to solve the GTD-ATSP-TW and the TD-ATSP-TWPC. The first section presents the generation and selection low level heuristics, commonly used in the literature. The second section show the Hyper-heuristic framework and its modifications developed in this work.

3.1 Low level heuristics

There are two main categories of heuristics related to methods for solving the TSP: (i) constructive or generation heuristics and (ii) local search or selection heuristics. Both of them can be used to compose the set of low level heuristics used within a Hyper-heuristic. Constructive heuristics are used to generate a solution from scratch, which usually works as starting point for a search algorithm. Diversely, local search heuristics are used within iterative algorithms that explore the neighborhood of a current solution, trying to find a better one, i.e., a solution that improves the objective function. We adopt the terms generation and selection, as they are the most used in the context of Hyper-heuristics.

3.1.1 Generation heuristics

As described in Section 2.2, Pylyavskyy et al. [10] proposed a Hyper-heuristic method to solve the GTD-ATSP-TW problem proposed by Kiwi.com in 2018 [2]. They used a random initial solution improved by a local search method to be a starting point of the search.

By reproducing their methodology using only the random initial solution, it was possible to notice the presence of many infeasible points that were difficult to fix during the execution of the hyper-heuristic. These points are generated when it is necessary to use the artificial costs associated with vertices without available flights between them, as mentioned in Section 2.1. Therefore, to amend this situation, the following constructive heuristics were implemented aiming to produce better initial solutions than the ones produced by [10] and remove the need of a pre-optimization to bring solutions closer to feasibility:

- Nearest Neighbor heuristic (NN) [5]: one of the most common and simplest heuristics used for finding initial solutions for the TSP. It starts in some vertex (the origin, if it exists), and at each step, it includes a non-visited vertex with the lowest cost from the latest vertex inserted in the current solution. This process ends when all vertices in the graph have been visited. If there is a connection between the last vertex and the first vertex, then a Hamiltonian cycle is built; otherwise a non-feasible solution is provided.

- Bellmore and Nemhauser (BN) heuristic [23]: similar to the NN heuristic, it also starts in some vertex (the origin, if it exists) but, at each step, it considers the two extremes of the current solution, the head and the tail, and examines the two non-visited vertices with the lowest cost to connect with these two extremes. The vertex included in the solution is the one with the lowest cost. This process ends when all vertices in the graph have been visited. If there is a connection between the last vertex and the first vertex, then a Hamiltonian cycle is built; otherwise a non-feasible solution is provided.

- Cheapest Insertion (CI) heuristic [5]: it starts with a partial tour consisting of two vertices (the origin, if it exists, and another vertex randomly chosen). At each iteration, it finds vertices k, i and j (i and j being the extremes of an edge belonging to the partial tour, and k not belonging to that tour) for which $c_{ik} + c_{kj} - c_{ij}$ is minimized, and it inserts vertex k between vertices i and j, removing the connection $i - j$ from the solution. This process ends when all vertices in the graph have been visited. If there is a connection between the last vertex and the first vertex, then a Hamiltonian cycle is built; otherwise a non-feasible solution is provided.

In the TD-ATSP-TWPC, the only particularity when applying these heuristics is that when a vertex is selected, there must be a validation on its presence in a precedence constraint or a time window in order to avoid generating solutions that do not satisfy all restrictions. Therefore, if a vertex that precedes another is selected, both are included in the solutions and the cost of including a vertex outside its time window or next to a vertex other than its precedence is as high as possible. In the GTD-ATSP-TW, the vertices being considered for selection in each iteration are airports and when any vertex is included in the solution, all vertices from the same area are removed from the set to be selected. This way, the initial solution satisfy the problem description of visiting only one airport of each area.

All three heuristics are simple and efficient enough to assist the search given the strict time limit imposed by the challenge. While NN and BN heuristics are $O(N^2)$ algorithms, the time complexity of the CI heuristic is $O(N^3)$. However, it won't have a big impact on the optimization performance because the largest instance size is when N is equal to 300 and there is enough time for the search. Finally, none of these heuristics guarantee the feasibility of the solution and the amount of artificial edges presented in the solution may have a big impact in the search method performance.

3.1.2 Selection heuristics

The selection heuristics are typically used in each iteration of a search method to create a little change in the current solution in a direction of improvement of the objective function. We use the following simple selection heuristics, commonly found in the literature (Figure 3.1 illustrates how each of these heuristics works. The first sequence is the original and the vertices highlighted in each line are the ones selected in the heuristic):

- SWAP: it randomly selects two vertices in the solution and exchanges their positions in the solution.
- INSERT: it randomly selects a vertex and a position in the solution. Then the vertex is removed from the current position and inserted into the selected position. Consequently, all vertices between the old position and the new one are moved; and
- REVERSE: it randomly selects two vertices in the solution and reverses all vertices between these two, including them.

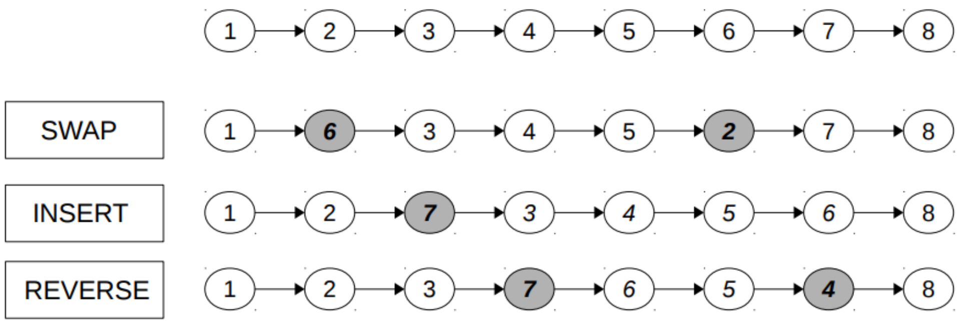


Figure 3.1: Illustration of how the selection heuristics work.

The first experiment with Hyper-heuristic uses only the SWAP and INSERT selection heuristics as low level heuristics, since the goal is to show that the combined use of these simple heuristics can produce better solutions than their use separately in distinct Local Search algorithms, as done by Saradatta and Pongchairerks [1] to solve the TD-ATSP-TWPC presented in Section 2.1. Moreover, it is important to notice that both selection heuristics were adapted to consider the time windows and the precedence constraints. This way, a vertex in the right time window do not change its position and adjacent vertices proper precedence do not move alone, avoiding violating the constraint. Lastly, in this case, the initial solution is randomly generated, as done in [1].

In the second problem, the Hyper-heuristic uses SWAP, INSERT and REVERSE selection heuristics along with a CHANGE AIRPORT heuristic that selects an area and changes, with a 50% probability, the visited airport for this area, as done in Pylyavskyy et al. [10]. However, differently from [10], that used a random initial solution improved by a local search procedure, we use the generation heuristics BN and CI to produce better initial solutions.

3.2 Hyper-heuristics

Heuristic and metaheuristic algorithms have been used to solve large instances of many hard combinatorial problems in acceptable execution time. However, these algorithms, like Ant Colony Optimization, Simulated Annealing and Tabu Search, need some specific knowledge of the problem and its peculiarities in order to adjust their parameters when there are changes in the problem description or in the scope of instances. Moreover, the use of simple heuristics alone may not be sufficient for obtaining results close enough to optimality. As a result, many researchers have devoted themselves to develop more sophisticated heuristic methods, known as Hyper-heuristics, that operates on a search space of heuristics instead of directly on the search space of solutions. These methods do not require much prior knowledge about the problem and are not greatly affected by instances with very different characteristics.

According to Burke et al. [24] and Drake et al. [25], there are two main categories of hyper-heuristics: generation Hyper-heuristics, which generate new heuristics from components of available heuristics, and selection Hyper-heuristics, which select one heuristic from a set of low level available heuristics at each iteration. This categorization is usually accompanied by subcategories. For example, selection constructive hyper-heuristics use constructive methods to gradually build solutions from scratch by choosing a heuristic at each moment of the construction. Conversely, selection pertubative hyper-heuristics operate on complete solutions, using the available low level heuristics to slightly change that solutions.

A common strategy for generation Hyper-heuristics is based on genetic programming (GP) [26]. The GP is similar to a genetic algorithm, but with individual represented using expression trees instead of permutations or integer numbers. This representation is used to encode heuristics that suffer mutations and crossover operations to generate new heuristics. Ðurasević and Ðumić [27] show the use of GP for generating relocation rules in the container relocation problem and Duflo et al. [28] use this kind of Hyper-heuristic focused on a heuristic generation method for the TSP.

This work focus on the use of selection pertubative Hyper-heuristics with an adapted Improve or Equal deterministic acceptance criterion to solve the variants of the TSP introduced in Sections 2.1 and 2.2. At each iteration, these methods choose a heuristic from a set of low level heuristics and apply it in the current solution trying to improve its cost in a direction of enhancement of the objective function. This approach is interesting because it takes advantage of the fact that some heuristics perform better at some stages of the search while other ones work better at other times. Therefore, the sequence in which the low level heuristics are applied is important and different sequences may generate interesting solutions [29]. Moreover, it is possible to introduce a feedback mechanism into the Hyper-heuristic framework to make it capable of learning the best sequence of application of the low level heuristics during the search process.

In short, a selection Hyper-heuristic is a general-purpose problem-independent heuristic search framework which operates a set of low level heuristics to solve computationally hard problems, using only limited problem information such as the objective function and the direction of the optimization. A Hyper-heuristic usually has a learning mechanism that collects and uses some feedback from the search process, such as the amount of times that each heuristic was used and their respective changes in the solution, generating scores for each heuristic based on its performance. These scores are used to select the heuristic to be applied at each step. Kheiri and Keedwell [29], for example, proposed a Hidden Markov Model to determine the transition matrix between the low level heuristics during the search.

The selection hyper-heuristic usually has two sequential steps: (i) heuristic selection and (ii) move acceptance. The first step selects the low level heuristic from the available set and applies it to the current solution, while the second one decides if it will accept or reject the new solution. This process is illustrated in Figure 3.2. The heuristic selection strategies used are the following:

- Simple Random (SR): it uses a uniform probability distribution to randomly select a low level heuristic at each step;

- Random Descent (RD): it selects a low level heuristic randomly and applies it repeatedly as long as an improvement is found;

- Random Permutation (RP): it generates a random ordering of the low-level heuristics and, at each step, successively applies a low-level heuristic in the provided order;
- Random Permutation Descent (RPD): it generates a random ordering of the low-level heuristics and applies each of them repeatedly as long as an improvement is found, respecting the provided order; and
- Reinforcement Learning (RL): As described by Pylyavskyy et al. [10] and detailed in Algorithm 1, it assigns an initial score to each low level heuristic in the beginning of the algorithm and adjusts the scores while learning through the iterations. When a low level heuristic improves a solution, its score is updated positively, while a worsening move decreases the score of a low level heuristic.

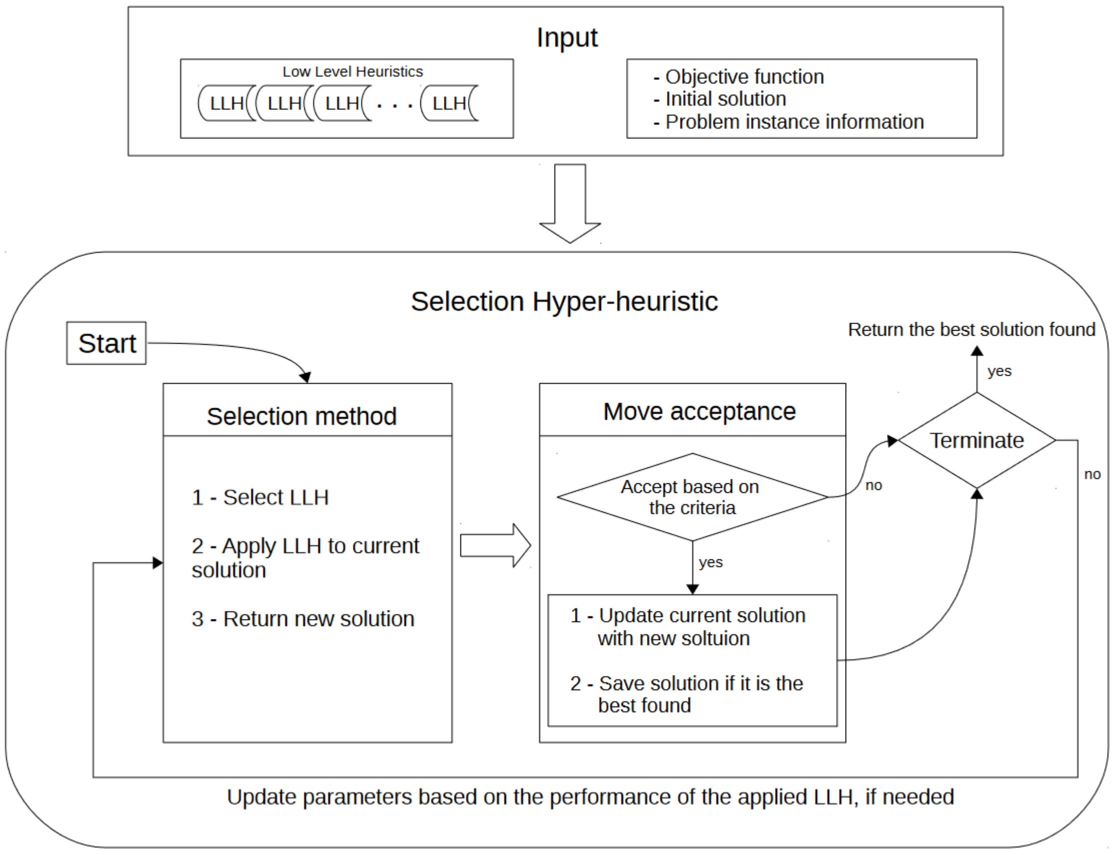

Input
Low Level Heuristics
LLH
LLH
LLH
. . .
LLH
- Objective function
- Initial solution
- Problem instance information
Selection Hyper-heuristic
Start
Selection method
1 - Select LLH
2 - Apply LLH to current solution
3 - Return new solution
Move acceptance
Accept based on the criteria
yes
1 - Update current solution with new soltuion
2 - Save solution if it is the best found
no
Terminate
yes
no
Return the best solution found
Update parameters based on the performance of the applied LLH, if needed

Figure 3.2: Hyper-heuristic framework diagram.

Algorithm 1: Hyper-heuristic introduced by Pylyavskyy et al. [10]

```
Data: set of $LLHs$, initial solution $s$, objective function $f$, artificial edges
      counter $a$, tolerance of non-improvement $\alpha$, factor for rewarding
      $reward$, factor for penalising worsening or non-feasible solutions
      $peanalty_w$ and $penalty_f$
Result: best solution $Best$
1  $Best \leftarrow s$;
2  $LLHS \leftarrow [0.5, 0.5, 0.5, 0.5]$;
3  $iter \leftarrow 1$;
4  $count \leftarrow 0$;
5  while $Execution\ time \leq Time\ limit$ do
6      $LLH_i \leftarrow$ SelectLLH($LLHs$);
7      $new\_s \leftarrow$ ApplyLLH($LLH_i$);
8      if $a(new\_s) == 0$ then
9          if $f(new\_s) < f(s)$ then
10             $s \leftarrow new\_s$;
11             $count \leftarrow 0$;
12             $LLHS_i \leftarrow LLHS_i + iter \times reward$;
13             if $f(new\_s) < f(Best)$ then
14                 $Best \leftarrow new\_s$;
15         else if $count > \alpha$ then
16             $s \leftarrow new\_s$;
17             $count \leftarrow 0$;
18             $LLHS \leftarrow [0.5, 0.5, 0.5, 0.5]$;
19         else
20             $LLHS_i \leftarrow LLHS_i - iter \times penalty_w$;
21             $count \leftarrow count + 1$;
22     else
23         $LLHS_i \leftarrow LLHS_i - iter \times penalty_f$;
24         $count \leftarrow count + 10$;
25     $iter \leftarrow iter + 1$;
```

Lastly, the acceptance strategy is an important component of any selection heuristic and it can be deterministic or non-deterministic. Deterministic strategies always make the same decision for acceptance regardless of the input, while a non-deterministic approach might generate a different decision for the same input. In the final Hyper-heuristic framework, is set an adapted Improve or Equal deterministic acceptance criterion which only accepts two kinds of solutions: (i) the ones with better or equal cost; or (ii) the ones having the same number of artificial edges, even if they have a slightly worse cost (as all the artificial edges have the same high cost, and the exchange is only allowed when the number of artificial edges is equal, the sum of artificial costs dominates the final solution cost, and the possible slightly worse in cost does not deteriorate the value of the solution much.)

3.2.1 Simulated Annealing

There are many cases in the literature where the acceptance strategy used in the Hyper-heuristic is based on metaheuristics such as Simulated Annealing [30, 31] and Tabu Search [32, 33]. The merge with Simulated Annealing brings a non-determinism to the Hyper-heuristic that can be used to better explore the capacity of each LLH.

The Simulated Annealing metaheuristic relies on two loops, the outer and the inner one. At each iteration, the inner loop is responsible for generating a new candidate solution, according to an appropriate neighborhood function and a validation step. Then, the outer loop updates the temperature based on a predefined cooling schedule. The inner loop usually ends when reaching a set number of iterations, while the outer stops when the temperature gets low enough.

In Algorithm 2 is presented a standard Hyper-heuristic combined with Simulated Annealing, the "SelectLLH" method abbreviate the implementation of the heuristic selection strategy. The acceptance criteria allow a move if it improves the current solution or if it is within the probability given by $e^{\frac{-d}{T}}$, where T is the current temperature and d is the cost difference of the two solutions. When the algorithm starts T is in a high temperature and the probability of accepting worsening movements is high to avoid reaching local minima. As the temperature is cooled, this probability decreases abruptly and the search behaves closer to Local Search algorithms.

Even though it is one of the most used combinations with Hyper-heuristics, this strategy contradicts some of the objectives proposed by Hyper-heuristics. The need for various parameter adjustments makes the method more dependent on the problem and instances it is dealing with. The values used for the Simulated Annealing parameters have a significant impact on the quality of the result. Different problems may perform better with faster or slower cooling, while the same initial and final temperature values have distinct effects for instances of the same problem. This way, the Hyper-heuristic with Simulated Annealing comes more to complement the Simulated Annealing method, by providing the possibility of accessing LLHs with several neighborhood structures, than the opposite.

In Section 4.2 a few more comments are included on these difficulties based on experiments made for the GTD-ATSP-TW. Moreover, as the main problem with this approach is presented in the contradiction of generalization from the Hyper-heuristic and the need for parameter tuning from Simulated Annealing, in Section 5.1 is included a idea of future work to make a Hyper-heuristic with Simulated Annealing that perform multiple searches and make parameter adjustment based on previous solutions found.

Algorithm 2: Hyper-heuristic with Simulated Annealing

```
Data: set of $LLHs$, initial solution $s$, initial temperature $T_0$, final
      temperature $T_f$, iterations at each temperature $K$, cooling rate $\alpha$,
      objective function $f$
Result: best solution $Best$
1  $T \leftarrow T_0$;
2  $Best \leftarrow s$;
3  while $T > T_f$ do
4      $count \leftarrow 0$;
5      while $count < K$ do
6          $LLH \leftarrow$ SelectLLH($LLHs$);
7          $new\_s \leftarrow$ ApplyLLH($LLH$);
8          $d \leftarrow f(new\_s) - f(s)$;
9          if $d \leq 0$ and $f(new\_s) < f(Best)$ then
10             $Best \leftarrow new\_s$;
11             $s \leftarrow new\_s$;
12         else if $exp(\frac{-d}{T}) > random(0,1)$ then
13             $s \leftarrow new\_s$;
14         $count \leftarrow count + 1$;
15     $T \leftarrow \alpha \cdot T$;
```

3.2.2 Path Relinking

Usually the Path Relinking is introduced in a Hyper-heuristic framework when it is already combined with other metaheuristics. Jiang et al. [15] combined the hyper-heuristics with GRASP and Path Relinking to solve the nurse rostering problem. Their method generated multiple initial solutions through a GRASP approach and used the result from the search on each of these to compose the Path Relinkg pool. Therefore, in this section is proposed a Path Relinking from a single search using local minima solution found during the process.

The motivation for using a Path Relinking post-optimization method lies in the fact that better solutions are likely to be found when the path between two high quality solutions found during the Hyper-heuristic search is explored. These high quality solutions are stored in a pool during the search to be used in the Path

Relinking procedure after the search. Algorithm 3 shows the inclusion of local minima solutions in a pool. In the "PathRelinking" method, all pairs of solutions present in the pool are selected to be explored.

For two solutions in the pool, one is set as source, while the other is used as a guide. To create the path between these two solutions, small changes are made in the first of one that approximate it to the second. For example, as in the GTD-ATSP-TW, a solution is represented by a sequence of vertices, swap two vertices to make at least one of them stay in the same position as in the guide solution. This way, during the construction of this path between the source and the guide, new solutions are generated and new sequences not explored before may lead to better costs.

The objective with the new solutions generated from the Path Relinking is not only in the possibility of new improvements in the best cost overall, but also in the fact that new good solutions are built and may be introduced in a new pool of high quality solutions to be returned to the final user without much additional time. In real-world applications, it is common to find best solutions that demonstrate a much greater implementation difficulty than other sub-optimal solutions at just as good a cost. By returning a pool of good solutions, a final user have more options to select a suitable answer to the real problem.

Our selection hyper-heuristic has three sequential steps: (i) heuristic selection, (ii) move acceptance, and (iii) pool insertion. The first selects the low level heuristic from the available set and apply it to the current solution, while the second decides if it will accept or reject the new solution produced in the previous step based on our adapted Improve or Equal. Lastly, in the third it is decided whether the new solution should enter the pool of high quality solutions for post-optimization with Path Relinking. This process is demonstrated in Algorithm 3.

As the GTD-ATSP-TW proposed by Kiwi.com is limited by execution time, the Path Relinking was made only as post-optimization to avoid process undesirable bad local minima solutions that would be removed from a full pool further in the search. In this case, the Hyper-heuristic is stopped $10^{-4} \cdot N \cdot PS$ seconds before the time limit, where N is the number of areas and PS is the pool size, to execute the Path Relinking between solutions in the pool. During the Hyper-heuristic search, when a new local optimal solution is found, it becomes a candidate to participate in the Path Relinking post-optimization procedure. Local minima are determined when the current solution does not changed for more than some predetermined number of iterations (in Algorithm 3 this value is set to N). This solution enter the pool in three situations:

1. if the pool is not full;

2. if the pool is full, and the solution has the best cost overall;

3. if the pool is full, the solution does not have the worst cost, and it is sufficiently different from the other solutions in the pool. In the last two cases, the solution with the worst cost is removed from the pool.

The third criteria to accept a solution in the pool is important to maintain diversity between solutions. It is common to find different good solutions sharing the same sub-sequence of vertices since it may be the main cause for the reduction in the cost. However, when two solutions are too similar, the Path Relinking is not that efficient because it does not generate many intermediate solutions in the path linking them. For the implementation in the GTD-ATSP-TW, the distance between two solutions is the number of positions with distinct vertices and they are considered sufficiently different when this number is greater than $\sqrt{N}$.

This Hyper-heuristics with Path Relinking was used in additon to the modified improve or equal acceptance criteria to solve the GTD-ATSP-TW described described in Section 2.2. As done in all experiments, the initial solutions were generated by the heuristics described in Section 3.1.1 and the selection heuristics, introduced in Subsection 3.1.2, composed the set of low level heuristics that were applied to the solutions to improve the objective function value.

Algorithm 3: Hyper-heuristic with Path Relinking

```
Data: set of $LLHs$, initial solution $s$, objective function $f$, artificial edges
      counter $a$, maximum pool size $PS$, instance size $N$, time limit $TL$
Result: best solution $Best$
1  $Best \leftarrow s$;
2  $count \leftarrow 0$;
3  while $Execution\ time \leq TL - (10^{-4} \cdot N \cdot PS)$ do
4      $LLH \leftarrow$ SelectLLH($LLHs$);
5      $new\_s \leftarrow$ ApplyLLH($LLH$);
6      $d \leftarrow f(new\_s) - f(s)$;
7      if $d \leq 0$ or $a(new\_s) = a(s)$ then
8          $s \leftarrow new\_s$;
9          $count \leftarrow 0$;
10         if $d \leq 0$ then
11             $Best \leftarrow new\_s$;
12     else
13         $count \leftarrow count + 1$;
14         if $count = N$ then
15             AddToPool($s$);
16 AddToPool($s$);
17 PathRelinking();
```

Chapter 4

Experiments and results

In this chapter is describe the experiments and results from our selection pertubative Hyper-heuristic framework applied to the Time-dependent ATSP with Time Windows and Precedence Constraints (TD-ATSP-TWPC) and the Generalized Time-dependent ATSP with Time Windows (GTD-ATSP-TW), both described in Chapter 2. All methods were coded in C++ language and all experiments were executed on a personal computer with an i7-7500U 2.7 GHz Intel processor and 8 GB of RAM memory. Moreover, each method was executed 10 times with different seeds to get a wider view of its performance. Various heuristic selection strategies were tested within the hyper-heuristic framework and the best of them is presented for each instance of the problems TD-ATSP-TWPC and GTD-ATSP-TW.

4.1 Results for the TD-ATSP-TWPC

The experiments presented in Saradatta and Pongchairerks [1], as commented in Chapter 1, were based on the comparison between the use of the modified nearest neighbor algorithm with two different local search heuristics - the SWAP and INSERT heuristics. They tested six small instances classified into two classes, each one having the same number of countries, time windows constraints and precedence constraints. Class 1 includes Instances 1 to 3, with 15 countries, in which two countries have pre-assigned visited week (one of them being the starting country), and one pair of countries has Precedence Constraints. Class 2 includes Instances 4 to 6, with 20 countries, in which three countries have a pre-assigned visit week, and two pairs of countries have precedence constraints. These instances are presented in Table 4.1, in which column "Dens." indicates the density of the adjacency matrix of each instance. It is worth to highlight that all standard deviations are null in these instances, but this might not be the case in more general instances, since, for each week, the adjacency matrix can change.

Our Hyper-heuristic combines the low level heuristics used in the Local Search

Table 4.1: Instances of the TD-ATSP-TWPC [1].

Instance	# of countries	Time Windows Constr.	Precedence Constr.	Dens.(%)
1	15	Thailand in week 1; Singapore in week 2.	France after Italy.	79.0
2	15	Thailand in week 1; South Korea in week 6.	Malaysia after Japan.	79.0
3	15	Thailand in week 1; Japan in week 9.	China after Germany.	79.0
4	20	Thailand in week 1; United Kingdom in week 6; Malaysia in week 9.	Japan before Australia; France before Hong Kong.	71.0
5	20	Thailand in week 1; Germany in week 6; Australia in week 11.	China before USA; South Korea before Malaysia.	71.0
6	20	Thailand in week 1; France in week 8; South Korea in week 13.	Australia before New Zealand; Hong Kong before United Kingdom.	71.0

algorithms of [1] to solve the TD-ATSP-TWPC. The method receives the set of low level heuristics (in this case, SWAP and INSERT) and the initial solution. For a better comparison with Saradatta and Pongchairerks [1], the same stopping criterion based on reaching N^2 consecutive iterations without improvement was used, where N is the number of vertices. The initial solution was also generated randomly as done in [1] in their local search algorithms. However, while their initial solutions are feasible roundtrips for TD-ATSP-TWPC, our initial solutions accepted infeasibility, i.e., tours including artificial connections with high costs $\tilde{c}_{ijk}$ penalized in the objective function, as introduced in Section 2.1.

The solution costs obtained were compared when the low level heuristics are used separately and together, to show the improvement provided by combining these heuristics in each execution of the method. The combined use of low level heuristics brought improvements both in terms of quality of solutions found and in the computational time to get them. In fact, we were able to achieve solutions with better values for 4 and tied in 2 out of 6 instances.

Table 4.2 shows the results when reproducing the Local Search methods of [1] (values in bold highlight the best solutions cost for each instance). Table 4.3 exhibits the results when applying the hyper-heuristic framework. The selection methods' variants included: SR, RD, RP, RPD and RL, and the best of them were included in the result table (all parameter values for the RL selection were the same as those used by Pylyavskyy et al. [10], i.e., $\alpha = 10^4, reward = 0.003, penalty_f = 0.000625$ and $penalty_w = 0.0005$). Columns "Best", "Worst", "Avg.", "Std.(%)" and "Avg. Time (s)" correspond to the cost of the best solution achieved, the cost of the worst, the average cost, the standard deviation in percentage and the average execution time in seconds, respectively. Column "Best Known" is the cost of the best solution found in the literature, in this case by Saradatta and Pongchairerks [1]. Lastly, column "Method" indicates the selection strategy responsible for obtaining the best result for each instance.

The results depicted in Table 4.3 prove that the combined use of low level heuristics in the Hyper-heuristic framework was able to provide better results and average solution costs for all instances compared to the Local Search algorithms used in Table 4.2. Only instances 1 and 2 had similar cost to the best known. We can also see that methods RD and RL were the ones that presented the best results more often.

Even with high standard deviation values in Table 4.3, we observed that the results reproducing Saradatta and Pongchairerks [1] experiments (Table 4.2) also include high values for the standard deviation. Moreover, our Hyper-heuristic framework was able to produce a slight improvement, as shown by the average standard deviations 12.44 in Table 4.2 and 10.39 in Table 4.3. In conclusion, for solving the TD-ATSP-TWPC, we can affirm that combining low level heuristics within a Hyper-heuristic framework leads to much better results.

Table 4.2: Results for the TD-ATSP-TWPC when reproducing the local search methods of [1].

Instance	Method	Best	Worst	Avg.	Std.Dev.(%)†	Avg.Time(s)	Best Known[1]
1	INSERT	**102280**	176970	128559	15.83	0.0037	**102280**
2	SWAP	127209	174103	157778	9.01	0.0039	**115122**
3	INSERT	137722	230970	175672	16.67	0.0036	**133329**
4	SWAP	219975	300723	262077	10.02	0.0087	**196620**
5	INSERT	205049	322688	258241	13.55	0.0081	**194682**
6	SWAP	176333	243099	206925	9.60	0.0084	**149065**

† The average standard deviation is 12.44.

Table 4.3: Results for the TD-ATSP-TWPC when applying our hyper-heuristic framework.

Instance	Method	Best	Worst	Avg.	Std.Dev.(%)†	Avg.Time(s)	Best Known[1]
1	SR	**102280**	139186	121859	9.71	0.0042	**102280**
2	RD	**115122**	164564	138332	11.93	0.0039	**115122**
3	RL	**129547**	174690	150226	10.19	0.0038	133329
4	RP	**191324**	277484	245733	11.18	0.0087	196620
5	RL	**191637**	243495	213667	7.23	0.0084	194682
6	RD	**147192**	213858	176065	12.15	0.0089	149065

† The average standard deviation is 10.39.

4.2 Results for the GTD-ATSP-TW

When Kiwi.com proposed the Travelling Salesman Challenge 2.0 [2], they provided 14 instances, with a range from 10 to 300 areas and airports, as shown in Table 4.4. Each instance has a different distribution of airports per area, and adjacency matrices whose density varies weekly, for the vast majority of instances, as indicated in columns "Airp. per area" and "Avg. Dens.", respectively. Column "Std. Dev. Dens." stands for the densities' standard deviations. Lastly, column "Time Lim." shows the time limit we used for the execution of the experiments on this problem, which were set by the challenge as the following rules:

- 3 seconds for instances with number of areas $N \leq 20$ and number of airports $M \leq 50$;
- 5 seconds for instances with $N \leq 100$ and $M \leq 200$; and
- 15 seconds for instances with $N > 100$.

Table 4.4: Instances of the GTD-ATSP-TW [2].

Instance	Areas	Airports	Airp. per area	Avg. Dens.(%)	Std. Dev. Dens.(%)	Time Lim.(s)
1	10	10	1	100.0	0.0	3
2	10	15	1 to 2	35.0	1.5	3
3	13	38	1 to 6	92.0	0.0	3
4	40	99	1 to 5	100.0	0.0	5
5	46	138	3	12.3	0.0	5
6	96	192	2	17.6	0.0	5
7	150	300	1 to 6	11.8	1.0	15
8	200	300	1 to 4	54.8	0.4	15
9	250	250	1	25.7	0.5	15
10	300	300	1	78.7	0.0	15
11	150	200	1 to 4	28.5	0.3	15
12	200	250	1 to 4	28.5	0.2	15
13	250	275	1 to 3	19.0	0.1	15
14	300	300	1	13.3	15.1	15

Usually, a strict execution time limit is not very impactful for metaheuristics and Hyper-heuristics, since these algorithms already respond much faster than exact methods. However, in this challenge the time limit is an important factor, as it is necessary to return a high quality solution in the total time of 3 to 15 seconds, depending on the instance size.

The instances from the TSC were already tested with a hyper-heuristic by Pylyavskyy et al. [10] and their method presented a good performance by improving the best known solution in 4 of the 14 instances and matching 3 others. In their case, the focus was on a selection pertubative Hyper-heuristic with Reinforcement Learning and random initial solutions refined by a Local Search to reach feasibility. However, they do not make it clear how the Local Search algorithms guaranteed feasibility and which stopping criteria was used to allow both Local Search and Hyper-heuristic execution within the time limit. Therefore, the objective in this work is to show the improvement in the final solutions when we apply different constructive heuristics to generate the initial solutions to the Hyper-heuristic instead of using random methods and propose a new Hyper-heuristic framework with Path Relinking that uses local minima solutions found during the search procedure.

Firstly, we implemented the generation heuristics described in Subsection 3.1.1 to the GTD-ATSP-TW. Table 4.5 compares the results from a random approach against the results when applying the constructive heuristics NN, BN and CI. Column "Art. edges" indicated the number of artificial connections needed to make the solution feasible. For the purpose of optimization and feasibility of the final solution, a high cost $\tilde{c}_{ijk}$ was attributed to the artificial connections, as done for the TD-ATSP-TWPC. These artificial costs are not included in the values presented in "Cost" column, as it is already highlighted in column "Art. edges", i.e., the values in column "Cost" are the sum of the costs from non-artificial edges in the solution.

The random approach generated solutions with too many artificial connections, deteriorating as the instance size grows or the density decreases. All the three constructive heuristics tested (NN, BN and CI) showed a significant improvement in the quality of the solution and may help the Hyper-heuristic to find good final solutions faster.

Table 4.5: Solutions of the GTD-ATSP-TW when using different constructive heuristics.

	Random		NN		BN		CI	
Instance	Cost	Art. edges	Cost	Art. edges	Cost	Art. edges	Cost	Art. edges
1	21313	0	5267	0	3498	0	1872	0
2	6196	3	8193	1	8193	1	6137	3
3	16882	0	8582	0	8536	0	8104	0
4	60776	0	18176	0	17089	0	16422	0
5	391	42	703	16	1082	1	1038	0
6	2607	74	1756	31	2289	1	2802	0
7	18188	121	29898	4	33041	1	34420	0
8	11269	73	4332	0	3234	1	4602	0
9	84791	184	84017	1	83996	1	78713	0
10	324202	66	52048	1	52872	0	13396	0
11	40149	100	40923	3	41501	4	22523	92
12	48387	137	60661	3	57760	2	31912	112
13	31088	206	86944	8	81134	7	28909	173
14	35737	257	113808	6	120512	8	27248	240

The NN and BN heuristics presented a similar behavior in terms of the amount of artificial connections in each instance. The difference between them in Instances 5 and 6 made us choose the BN heuristic over the NN. Conversely, the CI heuristic had another behavior in the column "Art. edges". In this case, almost all up to the tenth instance had no artificial connection and low costs. However, in more complex instances (11th to 14th), the heuristic had difficulty in generating good solutions and resulted close to the random case. Therefore, the BN and CI heuristics were used to generate the initial solutions for our Hyper-heuristics to enhance the search for good solutions and improve their cost compared to the best known ones. The choice for these constructive heuristics was very opportune because, unlike more sophisticated heuristics, they are simple, fast and do not return solutions too close to local minima that could affect the search.

Tables 4.6 and 4.7 show the results of the Hyper-heuristic with initial solutions from BN and CI heuristics respectively. The selection strategies used were the same as in the TD-ATSP-TWPC, i.e., SR, RD, RP, RPD and RL, and initially the move acceptance strategy adopted was the standard Improve or Equal with the algorithm executing until reaching the time limit of each instance.

Table 4.6: Results of the GTD-ATSP-TW when using the Hyper-heuristic with the BN heuristic.

		Solution found			Best Known	
Instance	Method	Best Cost	Avg. Cost	Std (%)[†]	Best of [10, 11, 13]	TSC [2]
1	RL	**1396**	1415.6	2.28	**1396**	**1396**
2	RL	**1498**	1498.0	0.00	**1498**	**1498**
3	RL	8071	8071.0	0.00	**7672**	**7672**
4	RD	14134	14791.1	2.49	**13952**	14024
5	SR	731	920.5	12.06	**690**	698
6	RP	*2262*	2338.7	2.26	2610	**2159**
7	SR	*30969*	31349.2	0.93	**30937**	31681
8	RD	*4043*	4079.8	0.46	**4041**	4052
9	RP	77864	80723.9	1.64	**75604**	76372
10	RP	*43614*	46350.3	1.97	58304	**21167**
11	SR	*44913*	46617.0	1.67	49453	**44153**
12	RDP	**57221**	59526.4	2.02	70082	65447
13	RD	**89279**	93534.1	2.74	164764	97859
14	-	-	-	-	198787	**118811**

[†] The average standard deviation is 2.35.

The solutions cost are compared with the best values found in the literature [10, 11, 13] and the available information given in the TSC by Kimi.com [2]. The value in bold represents the lowest cost found for the respective instance and the values in column "Best Cost" in italics are highlighted because they are solutions found with a cost between the two sources. As done in Table 4.3, the column "Method" in Table 4.8 indicates the selection method responsible for obtaining the result for each instance.

Table 4.7: Results of the GTD-ATSP-TW when using the Hyper-heuristic with the CI heuristic.

	Solution Found				Best Known	
Instance	Method	Best Cost	Avg. Cost	Std (%)†	Best of [10, 11, 13]	TSC [2]
1	RL	**1387**	1406.5	1.14	1396	1396
2	RL	**1498**	1498.0	0.00	**1498**	**1498**
3	RL	7936	7936.0	0.00	**7672**	**7672**
4	RP	14453	14563.2	0.69	**13952**	14024
5	RPD	804	972.0	6.43	**690**	698
6	RD	**1990**	2207.9	5.88	2610	2159
7	RD	***31564***	31782.3	0.38	**30937**	31681
8	RD	**4028**	4054.9	0.40	4041	4052
9	RPD	***76226***	76677.3	0.32	**75604**	76372
10	RPD	**12331**	12525.6	0.57	58304	21167
11	RD	60749	65929.0	3.83	49453	**44153**
12	SR	89790	93656.0	3.20	70082	**65447**
13	RP	137116	146671.0	3.44	164764	**97859**
14	-	-	-	-	198787	**118811**

† The average standard deviation is 2.02.

These results show that the initial solution generated from heuristics have a impact in the final cost. Good solutions were found in a few different cases. The "Art. edges" column in Table 4.5 shows that the performance of the CI heuristic is better in medium instances but much worse in large complex instances when compared to the BN heuristic. This behavior in the initial solution was reproduced in the cost of the final solutions. The Hyper-heuristic with CI initial solution was better for most instances from 1 to 10 (achieving the greatest difference in the tenth instance), and the Hyper-heuristic with BN initial solution was significantly better for instances 11, 12 and 13.

However, this first experiment was not satisfactory due to the absence of viable solutions at the end of the search for instance 14 in all cases. Furthermore, the success rate for other complex instances, mainly 13, was very low with viable solutions returning in few of the 10 runs. Finally, there is a visible deficiency in the solutions found for instances 3 to 5 compared to the references is visible.

4.2.1 Hyper-heuristic with modified improve or equal

Following the previous experiments, new tests were made using the Hyper-heuristic with Simulated Annealing (most common variation in the literature). Besides the difficulties commented in Subsection 3.2.1 regarding the need for parameter tuning, the characteristics of the instances from the TSC made this approach even more challenging. Each instance presented a different behavior on the costs associated to its edges. In some large instances the edges had a low cost, while in some small instances the edges had a high cost, making it necessary to use different initial temperatures for each instance. In addition, the low density for some adjacency matrices, added to the use of artificial edges with a very high cost, meant that at high temperatures the algorithm led to solutions with a quality close to random (present in Table 4.5), causing the deterioration of the initial solutions, and impacted the search performance at low temperatures as it needed to remove the many artificial connections before improving the cost of feasible solutions.

Regarding the Hyper-heuristic with Simulated Annealing, some other attempts were made to try to change the acceptance probability based on the presence of artificial edges in the current solution. Even managing to find solutions for instance 14, the low success rate remained and the costs of the best solutions increased. Then a new strategy was chosen, instead of restrict the acceptance criteria from the Simulated Annealing, it may be better to relax the previous improve or equal criteria used before.

As presented in Section 3.2, the adapted Improve or Equal acceptance criterion only accepts two kinds of solutions: (i) the ones with better or equal cost; or (ii) the ones having the same number of artificial edges, without increasing the number of artificial edges even if the solution has a slightly worse cost. This way, it accepts solutions with a slight deterioration while viability is not reached leading to a better performance of the algorithm, without compromising the quality of the initial solution. This allowed a better controlled increase in the search space compared to strategies based on Hyper-heuristics with SA which initially have greater flexibility to explore worsening movements and ended up losing qualities of the initial solution.

Tables 4.8 and 4.9 show the results of the Hyper-heuristic with initial solutions from BN and CI heuristics using the adapted improve or equal acceptance criteria (using the same layout as Tables 4.6 and 4.7). In the first, we improved the cost in five instances, matched two and got five with a cost between the two references. In the second, we improved the cost in eight instances and matched one. Moreover, the behavior in the initial solution from each constructive heuristic was reproduced in the cost of the final solutions as in the first experiment.

The number of low level heuristics used in both problems (GTD-ATSP-TW and TD-ATSP-TWPC) was not large enough for the variation in the heuristic selection method to express a significant pattern of improvement. This is well evidenced by the variety in the method that obtained the best result for each instance (column "Method" of Tables 4.3, 4.8 and 4.9). Yet, in the first instances with less vertices than the last ones, the Reinforcement Learning strategy was dominant in all experiments and was the one with most better solutions between the five selection strategies in all tables combined.

Instance 5 is a particular case for which both references [10] and [2] found solution costs below 700, and we were not able to achieve such values. Given the size of this

Table 4.8: Results of the GTD-ATSP-TW when using the Hyper-heuristic with the BN heuristic.

	Soluton Found				Best Known	
Instance	Method	Best Cost	Avg. Cost	Std (%)†	Best of [10, 11, 13]	TSC [2]
1	RL	**1396**	1409.8	1.61	**1396**	**1396**
2	RL	**1498**	1498.0	0.00	**1498**	**1498**
3	RL	**6857**	7284.5	2.70	7672	7672
4	RPD	**8180**	9499.8	8.05	13952	14024
5	RD	735	911.9	13.59	**690**	698
6	RPD	***2334***	2664.1	9.23	2610	**2159**
7	RP	**28941**	30479.2	1.98	30937	31681
8	SR	**3996**	4048.5	0.72	4041	4052
9	RD	77439	82633.6	2.88	**75604**	76372
10	RL	***43588***	46369.5	2.00	58304	**21167**
11	RP	***48774***	58284.9	8.31	49453	**44153**
12	SR	**59241**	69866.4	8.29	70082	65447
13	RPD	***106825***	128609.0	8.86	164764	**97859**
14	RPD	***146788***	171520.0	7.58	198787	**118811**

† The average standard deviation is 5.41.

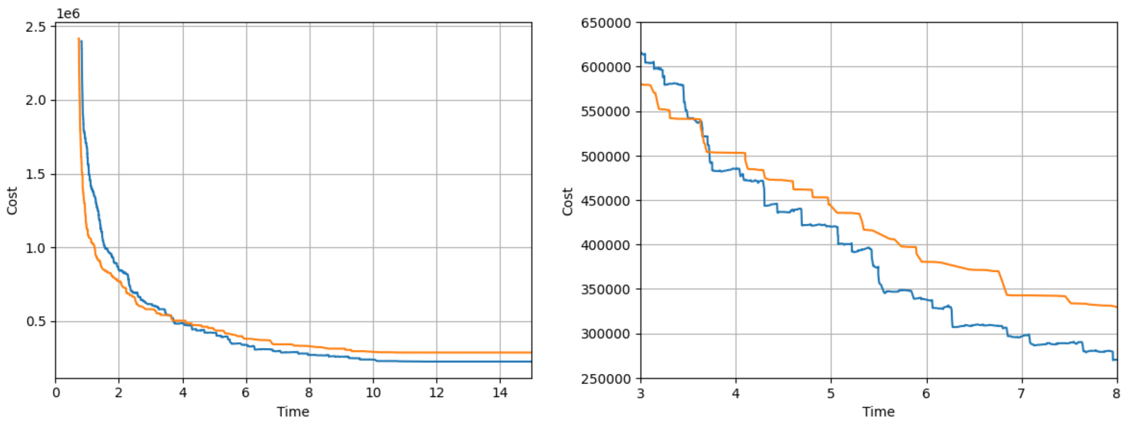
Table 4.9: Results of the GTD-ATSP-TW when using the Hyper-heuristic with the CI heuristic.

	Solution Found				Best Known	
Instance	Method	Best Cost	Avg. Cost	Std (%)†	Best of [10, 11, 13]	TSC [2]
1	RL	**1387**	1395.1	0.19	1396	1396
2	RL	**1498**	1498.0	0.00	**1498**	**1498**
3	RL	**7013**	7416.8	2.92	7672	7672
4	RL	**8190**	9360.3	10.41	13952	14024
5	RL	723	872.4	9.57	**690**	698
6	RPD	**2017**	2267.1	5.47	2610	2159
7	RPD	**30343**	30708.1	0.81	30937	31681
8	RL	**3977**	4025.5	0.58	4041	4052
9	RD	**75107**	75672.2	0.32	75604	76372
10	RL	**11864**	12059.2	1.06	58304	21167
11	RD	62717	67348.1	5.00	49453	**44153**
12	SR	95327	103714.0	3.85	70082	**65447**
13	RP	169539	177369.0	3.45	164764	**97859**
14	RD	219683	225618.0	2.07	198787	**118811**

† The average standard deviation is 3.26.

instance, comprising 46 areas and 138 airports, we thought our method would be able to find solutions in the same range of [10] and [2]. Probably the initial solutions generated by the BN and CI heuristics for this case were already too close to a local minima, so more than one LLH should be needed and the acceptance strategy should be adapted to tolerate multiple worsening in the cost of the solution, including a controlled increase in the number of artificial edges to escape that local minima.

Figure 4.1a shows the improvement of the solution cost during the 15 seconds execution for instance 14. The initial solutions was built using the CI heuristic. The



(a) Cost during all execution time (b) Cost between 3 and 8 seconds

Figure 4.1: Plots of cost versus time for the Hyper-heuristic

curve in orange is for a run with the standard improve or equal and the curve in blue is for the adapted improve or equal. The final cost was better in the second one, returning a feasible solution, while the first returned a solution with artificial edges responsible for the gap in the cost between the two final solutions. Moreover, analyzing executions between 3 and 8 seconds in Figure 4.1b, it is possible to notice the slight increase in the cost allowed in the modified improve or equal and the sharp cost drops due to the removal of artificial edges. In this case, the artificial edges were at a cost of 10000 and the SWAP LLH composed 70% of the improvements, while the other three LLH got approximately 10% of the improvements in the solution cost. This behavior may be explained by the sparsity in the adjacency matrix, the asymmetry, the time dependence characteristic of the problem and the fact that the INSERT and REVERSE LLHs not only impact the vertices selected in the operation, but also change the position of all vertices between them. It is also related to the dominance of the RL selection because it uses the performance from SWAP to prioritize it over the other less efficient heuristics.

Combining the results shown in Tables 4.8 and 4.9, we were able to improve the best solutions known in the literature for instances of the Generalized Time-dependent TSP problem from the Kiwi competition. Actually, we were better at 12 and tied at 1 out of 14 instances compared to the three papers in the literature and were better at 9 and tied at 1 out of 14 for the information provided in the competition. This evidences that the adequate use of good initial solutions can be very efficient in improving the final results. In fact, the strategy included the use of different constructive heuristics to quickly generate good initial solutions and feed the Hyper-heuristic with the modified improve or equal criterion to speed up the search and improve the cost of the final result when compared to the methods used in [10], [11], and [13]. This use of different constructive heuristics combined with the characteristics of our acceptance criterion was crucial to find better solutions in the end, since these heuristics presented distinct and complementary behaviors in relation to the tested instances.

4.2.2 Hyper-heuristic with Path Relinking and modified improve or equal

This experiment is to introduce the selection perturbative Hyper-heuristic with a Path Relinking post-optimization, that uses initial solutions built from simple constructive heuristics for the TSP and local minima found during a single search. The objective is to obtain at least one of the following improvements: (i) lower costs with the solutions found in Path Relinking and (ii) increase the quantity of high quality solutions to enable the return of a set of good solutions.

As done in the previous subsection, Tables 4.10 and 4.11 presents the results of the Hyper-heuristic with Path Relinking when using the BN and CI heuristics respectively. The same layout was used to highlight solutions cost and compare with the best known in the literature.

Table 4.10: Results of the GTD-ATSP-TW when using the Hyper-heuristic with BN heuristic and Path Relinking.

		Solution Found			Best Known	
Instance	Method	Best Cost	Avg. Cost	Std (%)†	Best of [10, 11, 13]	TSC [2]
1	RL	**1367**	1390.6	0.67	1396	1396
2	RL	**1498**	1498.0	0.00	**1498**	**1498**
3	RL	**7047**	7298.0	2.46	7672	7672
4	RPD	**8393**	10112.8	8.08	13952	14024
5	RL	732	998.7	15.65	**690**	698
6	RD	**2096**	2458.7	9.59	2610	2159
7	SR	**30110**	30629.7	0.88	30937	31681
8	RL	**3997**	4060.8	0.91	4041	4052
9	RPD	77398	84630.5	5.64	**75604**	76372
10	RP	***43651***	46376.7	1.96	58304	**21167**
11	SR	***47944***	53241.4	8.01	49453	**44153**
12	RD	**57362**	68046.4	10.18	70082	65447
13	RPD	***108772***	127453.0	8.60	164764	**97859**
14	RPD	***142991***	165147.0	7.81	198787	**118811**

† The average standard deviation is 5.74.

The same improvements as Subsection 4.2.1 were found. In the first case, we improved the cost in seven instances, matched one, and got four intermediates. In the second case, we improved the cost in eight instances and matched one. Moreover, combining these two cases, the Hyper-heuristic with Path Relinking and the modified improve or equal acceptance criteria was able to improve the costs from 12 instances compared to the well documented methods from the literature, i.e., the methods

Table 4.11: Results of the GTD-ATSP-TW when using the Hyper-heuristic with the CI heuristic.

	Solution Found				Best Known	
Instance	Method	Best Cost	Avg. Cost	Std (%)[†]	Best of [10, 11, 13]	TSC [2]
1	RL	**1384**	1402.4	1.89	1396	1396
2	RL	**1498**	1498.0	0.00	**1498**	**1498**
3	RL	**6577**	7227.2	4.70	7672	7672
4	RL	**7809**	9113.0	8.01	13952	14024
5	RPD	813	958.6	6.73	**690**	698
6	RPD	**1971**	2252.0	6.81	2610	2159
7	RP	**30526**	30738.2	0.47	30937	31681
8	RL	**3976**	4020.7	0.46	4041	4052
9	SR	**75098**	75686.7	0.41	75604	76372
10	RP	**11923**	12109.8	1.19	58304	21167
11	RPD	63763	67648.9	3.17	49453	**44153**
12	RD	96788	103769.0	3.82	70082	**65447**
13	SR	169229	178635.0	2.65	164764	**97859**
14	RD	213747	224962.0	2.20	198787	**118811**

[†] The average standard deviation is 3.03.

presented in [10, 11, 13]. Finally, considering the score system used in the TSC, where 100 points were given for each solution cost lower than shown in the last column of the result tables, and the winner score of 640 points over more than 500 teams, our set of best solutions would have made at least 900 point in the competition and at least 800 points if considering the average cost for each instance.

All final solutions found are feasible, so they have no artificial edges. This way, with only an average of 1% enhancement in the best solutions cost, there is no significant improvement in the final solution cost. However, new good solutions were generated for almost all instances, resulting in pools with better options for returning to a possible user. This set of high quality solutions is of great importance in real-world problems because, even modeling the problem as detailed as possible, a feasible solution with the best known cost may not offer the best conditions of implementation. Therefore, the return of a pool with different solutions and good costs is an advantage offered by the algorithm. In Table 4.12 is an example of the top 5 solutions in the pool for an execution of instance 14 with BN initial solution and RL selection strategy, showing the significant improvements made comparing the values before and after the Path Relinking.

Table 4.12: Solutions pool before and after the Path Relinking.

Before	After
164179	163572
164252	163634
164287	163635
164325	163650
164351	163733

Chapter 5

Conclusion and Future work

The TSP is a $\mathcal{NP}$-hard optimization with many real-world application, as in the context of air travel with flight connections. This utilization is studied by researches and interests business with the objective of improving their products. In such manner, emerged some challenges for the research and development communities from companies like Kiwi.com.

In this work was implemented the algorithms used by Saradatta and Pongchairerks [1] and embedded them into a new hyper-heuristic framework to solve the Time-dependent Asymmetric Traveling Salesman Problem with Time Windows and Precedence Constraints, showing that the combined use of heuristics can produce better solutions than their use separately. We were able to find better results in all instances, both in terms of solution quality and computational time. In particular, we were not able to find articles exploring the use of Hyper-heuristics for this problem, i.e., to the best of our knowledge, this would be the first work to present the performance of a Hyper-heuristic for it.

The Generalized Time-dependent ATSP with Time Windows was also explored, being the focused problem. The low level heuristics of Pylyavskyy et al. [10] were embedded into some Hyper-heuristic variations to solve the instances proposed in the Travelling Salesman Challenge 2.0 [2]. In particular, a standard Hyper-heuristic, a Hyper-heuristic with Simulated Annealing, a Hyper-heuristic with modified improve or equal acceptance criterion and a new Hyper-heuristic framework with Path Relinking using local minima from a single search procedure. These methods produced better initial and final solutions compared to three paper in the literature [10, 11, 13] focused in the same problem proposed by Kiwi.com [2].

Tests with the common hybrid between Hyper-heuristics and Simulated Annealing showed the disadvantage of this approach since The number of artificial edges increased a lot at the beginning of the algorithm, when the probability of accepting worsening moves was greater, causing the deterioration of the initial solutions. Due to these results, the modified improve or equal acceptance criterion was proposed to allow slight worse cost while viability is not reached. It led to a better performance of the algorithm, without compromising the quality of the initial solution and the objective of using constructive heuristic to easy the search.

Moreover, the tests included two constructive heuristics, BN and CI, to generate the initial solutions. Their use combined with the characteristics of the modified acceptance criterion were crucial to find better solutions in the end, since they presented distinct and complementary behaviors in relation to the tested instances. By using this all these strategies for the problem, we were able to find better results in the vast majority of instances, in terms of solution quality, since the stopping criterion is a fixed computational time specified in the challenge.

Lastly, a Hyper-heuristic framework with Path Relinking as post-optimization based on local minima solutions found during a single search was proposed as an alternative to the Hyper-heuristic with Path Relinking presented in the literature that uses the results of various searches to compose the pool for the Path Relinking procedure. This way, the new framework take out the need for multiple executions of the Hyper-heuristic, which return the final solution much faster. Even not improving significantly the cost of the best solutions, the method was able to refine the high quality solutions pool that can be further returned to a potential user to easy the implementation efforts usually associated with the best solution overall.

5.1 Future work

Based on the selection Hyper-heuristic using GRASP with Path Relinking proposed by Jiang et al. [15] where multiple Hyper-heuristic searches are executed to produce the solutions pool for the Path Relinking and the idea of Reinforcement Learning used in the selection strategy in [10], a new framework could be developed to remove the disadvantages of the parameter tuning in the Hyper-heuristic with Simulated Annealing. It would be a Hyper-heuristic with GRASP and Simulated Annealing assisted by a Reinforcement Learning that uses the result from a search as feedback to adjust the parameters (initial temperature, final temperature and cooling rate) before starting a new search loop.

We actually started working in this idea, leading to some changes after initial tests. However, it was not completed yet due to the need for further improvements in the performance of parameter tuning. Initially, tests on the Minimum Latency Problem [34] (a relative of TSP) were made on small and big instances with up to 1084 vertices. In the larger instances, not only the parameters adjustments were bad, but also the greedy randomized approach from GRASP to generate initial solution was returning costs too high even with really restricted candidate lists. Moreover, the introduction of the parameter to determine the candidate list size into the tuning mechanism made the results worse.

Other ways of generating good initial solutions including some variance could be compared to the strict heuristics described in Section 3.1 and the constructive approach based on GRASP. A selection constructive Hyper-heuristic, using the operators of the generation heuristics from Subsection 3.1.1 as LLHs and a simple random selection, turned out to be a better method to create good initial solutions with some variance. This new approach, based on a double selection Hyper-heuristic, not only stabilized the initial solution cost, but also removed one parameter from the tuning set.

Finally, as commented in Subsection 4.2.1, the difference in the edges costs of distinct instance for the same problem impacted the parameters choice. So, the following equations are used to standardize the range of values for each parameter: (i) $T_0 = \frac{-\Delta_{avg}}{\ln p_0}$ and (ii) $\alpha = \left(\frac{-\Delta_{avg}}{\ln p_f \cdot T_0}\right)^{\frac{1}{k}}$, where Δ_{avg} is the average absolute difference in the objective function for a perturbation in the solution, T_0 is the initial temperature for the probability p_o of accepting worse costs and α is the cooling rate for the the probability p_f of accepting worse costs after k outer iterations of the Simulated Annealing. This way, the cooling system can be defined by parameters (p_0 and p_f) ranged within 0 and 1 independent of the instance characteristics, and the amount of total iteration, as well as k, can be fixed based on the instance size.

References

[1] SARADATTA, T., PONGCHAIRERKS, P. "A time-dependent ATSP with time window and precedence constraints in air travel", *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, v. 9, n. 2-3, pp. 149–153, 2017.

[2] HANOUSKOVÁ, B. "Travelling Salesman Challenge 2.0 wrap-up. Tech community behind Kiwi.com". 2018. Online available at: `https://code.kiwi.com/travelling-salesman-challenge-2-0-wrap-up-cb4d81e36d5b`, accessed on March 2023.

[3] BOUSSAÏD, I., LEPAGNOT, J., SIARRY, P. "A survey on optimization metaheuristics", *Information sciences*, v. 237, pp. 82–117, 2013.

[4] CIRASELLA, J., JOHNSON, D. S., MCGEOCH, L. A., et al. "The asymmetric traveling salesman problem: Algorithms, instance generators, and tests". In: *Algorithm Engineering and Experimentation: Third International Workshop (ALENEX)*, pp. 32–59. Springer, 2001.

[5] ROSENKRANTZ, D. J., STEARNS, R. E., LEWIS II, P. M. "An analysis of several heuristics for the traveling salesman problem", *SIAM Journal on Computing*, v. 6, n. 3, pp. 563–581, 1977.

[6] PURKAYASTHA, R., CHAKRABORTY, T., SAHA, A., et al. "Study and analysis of various heuristic algorithms for solving travelling salesman problem—a survey". In: *Proceedings of the Global AI Congress 2019*, pp. 61–70. Springer, 2020.

[7] MARQUES, R., RUSSO, L., ROMA, N. "Flying tourist problem: Flight time and cost minimization in complex routes", *Expert Systems with Applications*, v. 130, pp. 172–187, 2019.

[8] VESELÝ, O. "Travelling Salesman Challenge. Tech community behind Kiwi.com". 2017. Online available at: `https://code.kiwi.com/travelling-salesman-challenge-recap-7956f433bc10`, accessed on October 2022.

[9] DUQUE, D., CRUZ, J. A., CARDOSO, H. L., et al. "Optimizing meta-heuristics for the time-dependent TSP applied to air travels". In: *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 730–739. Springer, 2018.

[10] PYLYAVSKYY, Y., KHEIRI, A., AHMED, L. "A reinforcement learning hyper-heuristic for the optimisation of flight connections". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2020.

[11] AHMAD, E. D., MUKLASON, A., NURKASANAH, I. "Route Optimization of Airplane Travel Plans Using the Tabu-Simulated Annealing Algorithm to Solve the Traveling Salesman Challenge 2.0". In: *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, pp. 217–221. IEEE, 2020.

[12] DUECK, G. "New optimization heuristics: The great deluge algorithm and the record-to-record travel", *Journal of Computational Physics*, v. 104, n. 1, pp. 86–92, 1993.

[13] ALRASHEED, M., MOHAMMED, W., PYLYAVSKYY, Y., et al. "Local search heuristic for the optimisation of flight connections". In: *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1–4. IEEE, 2019.

[14] SIMÕES, M., BAHIENSE, L., FIGUEIREDO, C. "Hyper-heuristics for the Time-dependent ATSP variants applied to air travel", *Submitted to RAIRO Operations Research*, 2023.

[15] JIANG, H., QIU, J., XUAN, J. "A hyper-heuristic using GRASP with path-relinking: A case study of the nurse rostering problem", *Journal of Information Technology Research*, v. 4, n. 2, pp. 31–42, 2011.

[16] KHEIRI, A., KEEDWELL, E. "Selection hyper-heuristics". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 983–996. Association for Computing Machinery, 2022.

[17] ABELEDO, H., FUKASAWA, R., PESSOA, A., et al. "The time dependent traveling salesman problem: polyhedra and algorithm", *Mathematical Programming Computation*, v. 5, n. 1, pp. 27–55, 2013.

[18] KANOH, H., OCHIAI, J. "Solving Time-Dependent Traveling Salesman Problems Using Ant Colony Optimization Based on Predicted Traffic". In: *Distributed Computing and Artificial Intelligence*, v. 151, pp. 25–32. Springer, 2012.

[19] DUMAS, Y., DESROSIERS, J., GELINAS, E., et al. "An Optimal Algorithm for the Traveling Salesman Problem with Time Windows", *Operations research*, v. 43, n. 2, pp. 367–371, 1995.

[20] MOON, C., KIM, J., CHOI, G., et al. "An efficient genetic algorithm for the traveling salesman problem with precedence constraints", *European Journal of Operational Research*, v. 140, pp. 606–617, 2002.

[21] DE MARCKEN, C. "Computational complexity of air travel planning", *MIT Lecture Notes, Fall*, 2003.

[22] SIMÕES, M., BAHIENSE, L., FIGUEIREDO, C. "Hyper-heuristics with Path Relinking applied to the Generalised Time-Dependent ATSP in Air Travel". In: *Proceedings of XII Latin-American Algorithms, Graphs and Optimization Symposium*, 2023.

[23] BELLMORE, M., NEMHAUSER, G. L. "The traveling salesman problem: a survey", *Operations Research*, v. 16, n. 3, pp. 538–558, 1968.

[24] BURKE, E. K., GENDREAU, M., HYDE, M., et al. "Hyper-heuristics: a survey of the state of the art", *Journal of the Operational Research Society*, v. 64, n. 12, pp. 1695–1724, 2013.

[25] DRAKE, J. H., KHEIRI, A., ÖZCAN, E., et al. "Recent advances in selection hyper-heuristics", *European Journal of Operational Research*, v. 285, n. 2, pp. 405–428, 2020.

[26] KOZA, J. R. "Survey of genetic algorithms and genetic programming". In: *Wescon conference record*, pp. 589–594. Western Periodicals Company, 1995.

[27] ĐURASEVIĆ, M., ĐUMIĆ, M. "Automated design of heuristics for the container relocation problem using genetic programming", *Applied Soft Computing*, v. 130, pp. 109696, 2022.

[28] DUFLO, G., KIEFFER, E., BRUST, M. R., et al. "A gp hyper-heuristic approach for generating tsp heuristics". In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 521–529. IEEE, 2019.

[29] KHEIRI, A., KEEDWELL, E. "A sequence-based selection hyper-heuristic utilising a hidden Markov model". In: *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pp. 417–424, 2015.

[30] GARZA-SANTISTEBAN, F., SÁNCHEZ-PÁMANES, R., PUENTE-RODRÍGUEZ, L. A., et al. "A simulated annealing hyper-heuristic for job shop scheduling problems". In: *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 57–64. IEEE, 2019.

[31] BAI, R., BURKE, E. K., KENDALL, G., et al. "A simulated annealing hyper-heuristic for university course timetabling". In: *Practice and Theory of Automated Timetabling VI*, pp. 345–350. Springer, 2006.

[32] KENDALL, G., HUSSIN, N. M. "A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA University of Technology". In: *Practice and Theory of Automated Timetabling V*, pp. 270–293. Springer, 2005.

[33] ZAMLI, K. Z., ALKAZEMI, B. Y., KENDALL, G. "A Tabu Search hyper-heuristic strategy for *t-way* test suite generation", *Applied Soft Computing*, v. 44, pp. 57–74, 2016.

[34] SANTANA, Í., PLASTINO, A., ROSSETI, I. "Improving a state-of-the-art heuristic for the minimum latency problem with data mining", *International Transactions in Operational Research*, v. 29, n. 2, pp. 959–986, 2022.