Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 223 (2023) 35–42

Procedia Computer Science

www.elsevier.com/locate/procedia

XII Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS 2023)

# Hyper-heuristics with Path Relinking applied to the Generalised Time-Dependent ATSP in air travel

Matheus Simões[a,*], Laura Bahiense[a], Celina Figueiredo[a]

[a]*Federal University of Rio de Janeiro, Rio de Janeiro, Brazil*

**Abstract**

In this work we propose the use of path relinking within a hyper-heuristic framework to solve the Generalised Time-Dependent Asymmetric Traveling Salesman problem applied to Air Travel. We implemented several heuristic selection methods, such as Simple Random, Random Descent, Random Permutation and Reinforcement Learning. We were able to achieve very good solutions for 13 of the 14 instances from a well-known benchmark set, evidencing that the adequate use of a hyper-heuristic framework with path relinking can be very efficient in improving the final results.

*Keywords:* Hyper-heuristics; Path Relinking; Asymmetric Traveling Salesman Problem; Time-Dependent; Air Travel

## 1. Introduction

Heuristic and metaheuristic algorithms have been used to solve large instances of many hard combinatorial problems, when exact methods are not capable of finding the optimal solution within a reasonable computational time. However, metaheuristic algorithms need some specific knowledge of the problem and its peculiarities in order to adjust their parameters and procedures, when there are changes in the problem description or in the scope of the instances used. 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.

A hyper-heuristic is a general-purpose problem-independent heuristic search framework which operates on a set of low level heuristics to solve computationally hard problems [1]. It uses only limited problem information, such as the objective function and the direction of the optimization. Thus, the hyper-heuristic raises the level of generality compared to other search algorithms like metaheuristics, as it is able to adapt to different problem instances and domains in a more automated way. 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 low level heuristic was used and their

* Corresponding author.
*E-mail addresses:* simoesmc@cos.ufrj.br (Matheus Simões)., laura@cos.ufrj.br (Laura Bahiense)., celina@cos.ufrj.br (Celina Figueiredo).

respective impact in the solution, generating scores for each heuristic based on its performance. These scores may be used to select the low level heuristic to be applied at each step, generating a reinforcement learning procedure.

Lastly, as in many search algorithms, the acceptance strategy is a component with great impact in the quality of the final solution, since it is used to determine if a new solution generated in an iteration of the algorithm is suitable to replace the current solution. Deterministic strategies always make the same decision for acceptance based in the input, while a non-deterministic approach might generate different decisions for the same input. Additionally, there are many cases in the literature where the acceptance strategy used in the hyper-heuristic is based on metaheuristics such as simulated annealing [2, 3] and tabu search [4, 5].

It is common to find the use of path relinking methods within a metaheuristic algorithm or as a post-optimization technique. 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. [6] 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.

In 2018, Kiwi.com proposed the Travelling Salesman Challenge 2.0 (TSC) [7] based on a variant of the TSP with application in air travel, the Generalised Time-Dependent Asymmetric Traveling Salesman Problem with Time Windows (GTD-ATSP-TW). Given 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 area and terminates at the starting area. During the trip, it is not possible to arrive at an airport and then continue the trip by departing from another one 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. The time dependence is based on the different costs of traveling between two cities in distinct days. 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, an artificial edge, with a high artificial cost is created to penalize the objective function if this flight is present in a solution.

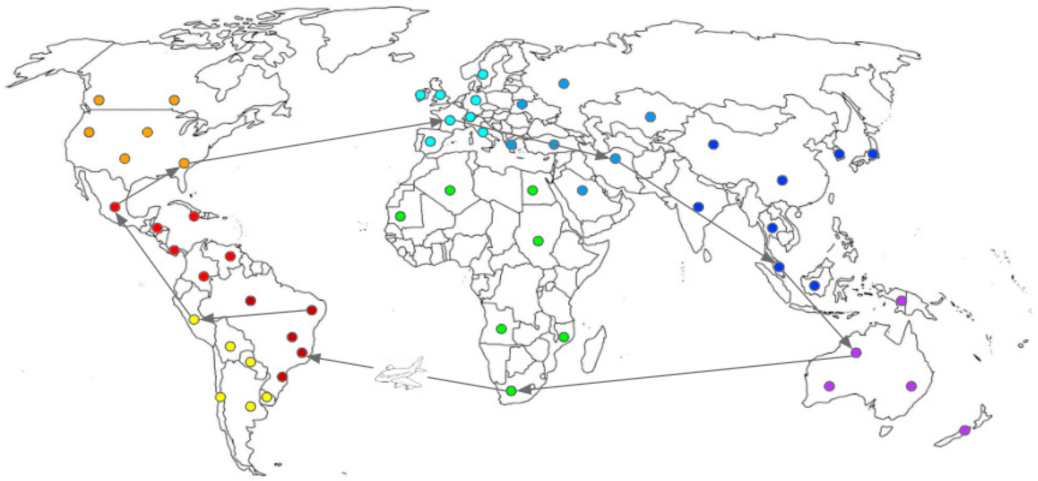
Fig. 1. Example of an instance of the GTD-ATSP-TW showing a feasible solution that starts and ends in the same area over Brazil.

There are several variants of the TSP in the literature, depending on each real-world application that is modeled by this very important problem. The problem studied in this work is a variant of the Asymmetric Traveling Salesman Problem (ATSP) including some of the following characteristics: Asymmetric TSP (ATSP), since 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), as the travel cost depends on the distance and the day of travel; Generalised TSP (GTSP), since the cities are divided into clusters and the salesman visits exactly one city of each cluster; and TSP with Time Windows (TSP-TW), as the salesman must visit a city within a specified time window, caracterized by the definition of an origin in the specific data set.

The challenge comprised 14 instances from small and simple to large and complex. In particular, the large instances are difficult to solve via exact methods considering the execution time limit imposed by the challenge. Usually, a strict running time 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.

Ahmad et al. [8] developed a hybrid heuristic combining Tabu Search and Simulated Annealing to solve the instances introduced in TSC [7]. They also showed that their hybrid approach could improve the initial solutions quite a bit compared to the great deluge algorithm [9]. Moreover, Pylyavskyy et al. [10] proposed a hyper-heuristic algorithm to solve the same problem. They tested six selection hyper-heuristics that controlled a set of four low level heuristics to improve the initial solutions.

There is an extensive literature on heuristics for the TSP (we may cite, for instance, [11]) that can be adapted 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. We refer to Kheiri and Keedwell [12] for a presentation of the Selection Hyper-heuristics and some examples of how to apply low level heuristics in an iteration of the search for the optimal solution.

In this work we propose the use of path relinking within a hyper-heuristic framework to solve the Generalised Time-Dependent Asymmetric Traveling Salesman problem based on the Travelling Salesman Challenge 2.0 (TSC) by kiwi.com [7]. We compare our results with the ones found in the competition and by the simpler hyper-heuristic without path relinking proposed by Pylyavskyy et al. [10] to solve this problem. They used random initial solutions further improved by a local search procedure. Diversely, we embedded their low level heuristics into our new hyper-heuristic framework, producing better initial solutions generated by simple well known heuristics. On top, we used the path relinking to connect the local optimal solutions found during the search. As a result, we were able to find very good solutions for 13 of the 14 instances, in terms of solution quality, since the stopping criterion was the computational times specified in the challenge. The outcome showed that the adequate use of a hyper-heuristic framework with path relinking can be very efficient in improving the final results.

The remainder of this paper is organized as follows. In Section 2 we present the hyper-heuristic framework with path relinking developed in this work, and a brief description of the constructive heuristics used to generate initial solutions and the low level heuristics operated in the hyper-heuristic. In Section 3 we present our computational experiments, comparing our results on the GTD-ATSP-TW to the best available in the literature. Finally, Section 4 shows our conclusions.

## 2. Hyper-heuristics

According to Burke et al. [1] and Drake et al. [13], there are two main categories of hyper-heuristics: generation hyper-heuristics, which generate new heuristics that create or change solutions, and selection hyper-heuristics, which select one heuristic from a set of low level heuristics (LLH) in each iteration of the search procedure. This categorization can be more detailed using subcategories. For example, selection constructive hyper-heuristics use constructive methods to gradually build solutions from scratch by choosing a low level heuristic at each moment of the construction. On the other hand, selection perturbative hyper-heuristics operate on complete solutions, using the available low level heuristics to slightly change them. In this case, the goal of the hyper-heuristic is to find an optimal sequence of low level heuristics, which can be applied to generate optimal or near-optimal solutions.

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. 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 illustrated in Figure 2. The following five heuristic selection strategies were used in the computational experiments in Section 3: the Simple Random (SR), which uses a uniform probability distribution to randomly select a low level heuristic at each step; the Random Descent (RD), which selects a low level heuristic randomly and applies it repeatedly as long as an improvement is found; the Random Permutation (RP), which generates a random ordering of the low level heuristics and, at each step, successively applies a low level heuristic in the provided order; the Random Permutation Descent (RPD), which 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 the Reinforcement Learning (RL), which 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 or equals a solution, its score is updated positively, while a worsening move decreases the score of a low level heuristic.

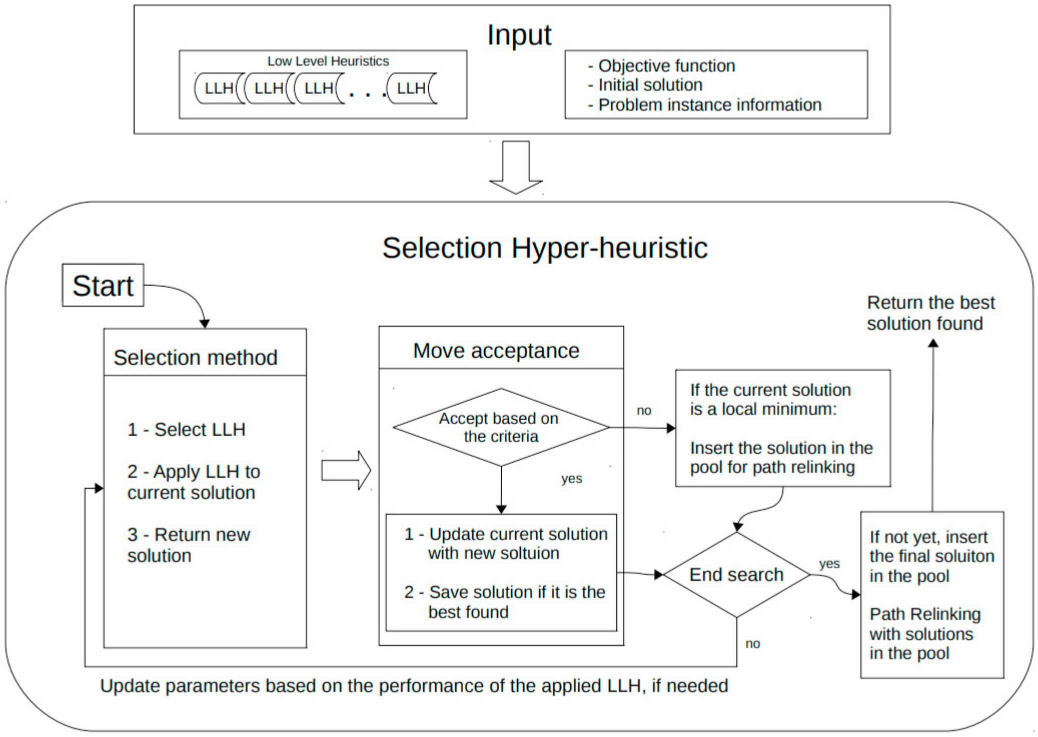
Input
Low Level Heuristics
LLH LLH LLH . . . LLH
- Objective function
- Initial solution
- Problem instance information
Selection Hyper-heuristic
Start
Selection method
1 - Select LLH
2 - Apply LLH to current solution
3 - Return new solution
Move acceptance
Accept based on the criteria
no
yes
1 - Update current solution with new soltuion
2 - Save solution if it is the best found
If the current solution is a local minimum:
Insert the solution in the pool for path relinking
End search
yes
no
Return the best solution found
If not yet, insert the final soluiton in the pool
Path Relinking with solutions in the pool
Update parameters based on the performance of the applied LLH, if needed

Fig. 2. Hyper-heuristic framework.

In each iteration, these selection 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 procedure of selecting different low level heuristics is interesting because it takes advantage of the fact that some heuristics may perform better at some stages of the search while other ones work better in other moments. Therefore, the sequence in which the low level heuristics are applied is important since different sequences may generate diverse solutions. 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, as done in the reinforcement learning selection method.

An adapted Improve or Equal was used as acceptance criterion. In this deterministic approach, two kinds of solutions are accepted: (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. In the second case, as all artificial edges have the same high value and we only allow the exchange when the number of artificial edges is equal, the sum of artificial costs dominates the final solution, and the possible slight worsening from the standard edges does not deteriorate the solution cost much.

The motivation for using a path relinking post-optimization method lies in the fact that better solutions are likely to be found when we explore the path between two high quality solutions found during the hyper-heuristic search. These high quality solutions are stored in a pool during the search to be used in the path relinking procedure after the search. At each iteration, the path relinking is applied to two solutions in the pool - one is the initial solution, while the other is the guiding solution. To create the path between these two solutions, small changes are made in the first of them that approximate it to the second one. For example, in a sequence of vertices, swap two of them to make the positions in the two sequences more similar. In the construction of this path between the initial and guiding solutions, new sequences that connect the vertices in ways not explored before may lead to better solutions.

As the problem 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, we stopped the hyper-heuristic $10^{-4} \cdot N \cdot PS$ 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. This solution enter the pool in three situations: (i) if the pool is not full; or (ii) if the pool is full, and the solution has the best cost overall; or (iii) 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.

Differently from [10], that used a random initial solution improved by a local search procedure, we use generation heuristics to produce better initial solutions and to facilitate the discovery of high quality solutions during the search. We implemented the following two constructive heuristics: the Bellmore and Nemhauser (BN) heuristic [14], which starts in some vertex (the origin, if it exists) and, 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; and the Cheapest Insertion (CI) heuristic [15], which 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.

The low level heuristics in a selection perturbative hyper-heuristics are typically used in each iteration of the search procedure to create little changes in the current solution in a direction of improvement of the objective function. We use the following four simple low level heuristics, commonly found in the literature: the SWAP, which randomly selects two vertices in the solution and swaps them. If the solution contains artificial edges, then one of the selected vertices must be an endpoint of an artificial edge; the INSERT, which randomly selects a vertex and a position in the solution. If the solution contains artificial edges, the selected vertex must be an endpoint of an artificial edge. 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 the REVERSE, which randomly selects two vertices in the solution and reverses all vertices between these two, including them.

Our hyper-heuristic uses SWAP, INSERT and REVERSE selection heuristics with a 50% probability of changing the airport of the area corresponding to the vertex selected in each operation to solve the GTD-ATSP-TW from kiwi.com [7] challenge. In addition to these selection heuristics, we use a CHANGE AIRPORT heuristic that selects an area and changes its airport, as done by Pylyavskyy et al. [10].

## 3. Computational Experiments

In this section we describe the experiments and results from our selection perturbative hyper-heuristic framework with path relinking applied to the Generalized Time-Dependent ATSP with Time Windows (GTD-ATSP-TW) from the Travelling Salesman Challenge 2.0 hosted by Kiwi.com [7]. All methods were coded in C++ language and 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 aiming 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 problem. The results for all strategies can be found in the GitHub via links presented in the text.

A total of 14 instances were used in the original competition, with a range from 10 to 300 areas and airports, as shown in Table 1. Each instance has a different distribution of airports per area, and adjacency matrices with varying density, as indicated in columns “Airp. per area” and “Avg. Dens”, respectively. Column “Std. Dens” stands for the densities’ standard deviations. Lastly, column “Time Lim.” shows the time limit set by the challenge to solve the problem for each instance. In this problem, the execution time limit is an important constraint, since it stipulates a very restricted time limit to return a high quality solution, varying from a total of 3 to 15 seconds, depending on the instance size.

This problem was already solved by 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 three others. In their case, the focus was on a selection perturbative hyper-heuristic with reinforcement learning and random initial solution refined

Table 1. Instances of the GTD-ATSP-TW [7].

| Instance | Areas | Airports | Airp. per area | Avg. Dens.(%) | Std. 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 |

by a local search to reach feasibility. Therefore, the objective in this work is to introduce and compare the performance of our selection perturbative hyper-heuristic with a path relinking post-optimization, that uses initial solutions built from simple constructive heuristics for the TSP.

Table 2 presents the results of our hyper-heuristic with path relinking when using the Bellmore and Nemhauser (BN) heuristic to generate the initial solutions. We compared the costs of our best solutions with the best ones found by Pylyavskyy et al. [10] and Ahmad et al. [8], and the results from the Travelling Salesman Challenge 2.0 [7]. In each line, the value in bold represents the lowest final cost found for the respective instance. The values that appear in italic in column "Best Cost" highlight solutions that we found having a cost between the two references. We improved the cost in seven instances, matched one, and got four intermediates.

Table 2. Results of the GTD-ATSP-TW when using the Hyper-heuristic with the BN heuristic.

| | Initial solution | | | Our solution | | | Best Known | |
|---|---|---|---|---|---|---|---|---|
| Instance | Cost | Art. Edges | Method | Best Cost | Avg. Cost | Std (%)[†] | Best of [8, 10] | TSC [7] |
| 1 | 3498 | 0 | RL | **1367** | 1390.6 | 0.67 | 1396 | 1396 |
| 2 | 8193 | 1 | RL | **1498** | 1498.0 | 0.00 | **1498** | **1498** |
| 3 | 8536 | 0 | RL | **7047** | 7298.0 | 2.46 | 7672 | 7672 |
| 4 | 17089 | 0 | RPD | **8393** | 10112.8 | 8.08 | 13952 | 14024 |
| 5 | 1082 | 1 | RL | 732 | 998.7 | 15.65 | **690** | 698 |
| 6 | 2289 | 1 | RD | **2096** | 2458.7 | 9.59 | 2610 | 2159 |
| 7 | 33041 | 1 | SR | **30110** | 30629.7 | 0.88 | 30937 | 31681 |
| 8 | 3234 | 1 | RL | **3997** | 4060.8 | 0.91 | 4081 | 4052 |
| 9 | 83996 | 1 | RPD | 77398 | 84630.5 | 5.64 | **75604** | 76372 |
| 10 | 52872 | 0 | RP | ***43651*** | 46376.7 | 1.96 | 58304 | **21167** |
| 11 | 41501 | 2 | SR | ***47944*** | 53241.4 | 8.01 | 59361 | **44153** |
| 12 | 57760 | 4 | RD | **57362** | 68046.4 | 10.18 | 86074 | 65447 |
| 13 | 81134 | 7 | RPD | ***108772*** | 127453.0 | 8.60 | 164764 | **97859** |
| 14 | 120512 | 8 | RPD | ***142991*** | 165147.0 | 7.81 | 198787 | **118811** |

[†] The average standard deviation is 5.74.

The columns "Cost" and "Art. Edges" in Table 2 specify, respectively, the cost of the initial solution generate by the BN heuristic and the amount of artificial edges in this solution. The column "Method" indicates the selection method responsible for obtaining the result for each instance. To see the complete results when applying all variants described in Section 2, refer to https://github.com/MatheusCSimoes/LAGOS2023. Finaly, the columns "Best Cost", "Avg. Cost" and "Std" indicate the cost of the final solution, the average cost and the standard deviation of the cost in each instance. All final solutions found are feasible, so they have no artificial edges.

Table 3 presents the results of our hyper-heuristic with path relinking when using initial solutions generated by the Cheapest Insertion (CI) heuristic. The costs of solutions for each instance were compared in the same way as in Table 2. In this case, we improved the cost in eight instances and matched one.

Table 3. Results of the GTD-ATSP-TW when using the Hyper-heuristic with the CI heuristic.

| | Initial solution | | | Solution Found | | | Best Known | |
|---|---|---|---|---|---|---|---|---|
| Instance | Cost | Art. Edges | Method | Best Cost | Avg. Cost | Std (%)[†] | Best of [8, 10] | TSC [7] |
| 1 | 1872 | 0 | RL | **1384** | 1402.4 | 1.89 | 1396 | 1396 |
| 2 | 6137 | 3 | RL | **1498** | 1498.0 | 0.00 | **1498** | **1498** |
| 3 | 8104 | 0 | RL | **6577** | 7227.2 | 4.70 | 7672 | 7672 |
| 4 | 16422 | 0 | RL | **7809** | 9113.0 | 8.01 | 13952 | 14024 |
| 5 | 1038 | 0 | RPD | 813 | 958.6 | 6.73 | **690** | 698 |
| 6 | 2802 | 0 | RPD | **1971** | 2252.0 | 6.81 | 2610 | 2159 |
| 7 | 34420 | 0 | RP | **30526** | 30738.2 | 0.47 | 30937 | 31681 |
| 8 | 4602 | 0 | RL | **3976** | 4020.7 | 0.46 | 4081 | 4052 |
| 9 | 78713 | 0 | SR | **75098** | 75686.7 | 0.41 | 75604 | 76372 |
| 10 | 13396 | 0 | RP | **11923** | 12109.8 | 1.19 | 58304 | 21167 |
| 11 | 22523 | 92 | RPD | 63763 | 67648.9 | 3.17 | 59361 | **44153** |
| 12 | 31912 | 112 | RD | 96788 | 103769.0 | 3.82 | 86074 | **65447** |
| 13 | 28909 | 173 | SR | 169229 | 178635.0 | 2.65 | 164764 | **97859** |
| 14 | 27248 | 240 | RD | 213747 | 224962.0 | 2.20 | 198787 | **118811** |

[†] The average standard deviation is 3.03.

The motivation to use the BN and CI heuristics to generate the initial solutions lies in the fact that they produce solutions with different patterns: while the BN heuristic generated almost all initial solutions with few artificial edges, the CI heuristic produced almost all initial solutions (up to the tenth instance) with no artificial edges and low costs (however, in more complex instances, the CI heuristic had great difficulty in generating good solutions). These heuristics distinct and complementary behaviors had visible impact in the final solution cost. In the first 10 instances, the CI heuristic provided better initial solutions and the final cost is slightly better in most of them. However, from instances 11 to 14, the BN heuristic provided much better initial solutions and the difference in the final cost is considerably greater.

The number of instances available in the TSC [7] and the number of low level heuristics used to solve the GTD-ATSP-TW for these instances were not large enough for the variation in the heuristic selection strategy 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 2 and 3).

Combining the results shown in Table 2 and Table 3, we were able to improve the solution cost of nine instances, match the cost in one, and obtain three intermediaries. Moreover, when compared with the well documented methods from the literature, we improved the solutions in 13 of the 14 instances of the GTD-ATSP-TW. This evidences that the adequate use a hyper-heuristic framework with path relinking can be very efficient in improving the final results.

## 4. Conclusions and future work

We proposed a hyper-heuristic framework with path relinking as post-optimization and used the Travelling Salesman Challenge 2.0 from kiwi.com [7] as a case study to show the potential of the method compared to a more traditional hyper-heuristic used by Pylyavskyy et al. [10] and the results from the competition itself, where diverse methods were used. The problem was based on the Generalized Time-Dependent Asymmetric Travelling Salesman problem with Time Windows applied to Air Travel. We implemented several heuristic selection strategies and used an adapted Improve or Equal move acceptance criterion to solve the instances varying from simple with dozens of vertices to complex with hundreds of vertices. We were able to achieve very good solutions for 13 of the 14 instances compared to results known in the literature, in terms of solution quality, since the stopping criterion was a fixed computational time specified in the challenge. We evidenced that the hyper-heuristic framework with path relinking can perform very well compared to others hyper-heuristics, such as [10], and other methods [8] used in the challenge.

In our hyper-heuristic framework, we used a deterministic acceptance criterion, and the number of iterations to determine local optimal solutions during the search and include them in the pool for the path relinking. Based on the potential of improvement in the solution cost found with our method in the greatest instance, we suggest for future work the application of non-deterministic acceptance criteria, such as the Monte Carlo-based move acceptance strategies, which accept all improving moves and some non-improving ones with a certain probability. We think that this kind of acceptance strategy can enhance the variability of the solutions in the pool for the path relinking. Lastly, we observed that the variation in the selection strategy in the hyper-heuristic did not express a significant pattern of improvement. We suspect this was caused by the low number of instances available in the challenge and the number of low level heuristic used. Therefore, we suggest introducing more LLHs and generating synthetic instances to better analyse the impact in the behavior of the different selection strategies.

## Acknowledgements

This work was partially supported by Brazilian agencies CAPES (Finance Code 001), CNPq (Grant Numbers 305356/2021-6 and 308792/2022-0, and Scholarship Number 130498/2023-8), and FAPERJ (Grant Number: E-26/010/001271/2016).

## References

[1] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724.

[2] F. Garza-Santisteban, R. Sánchez-Pámanes, L. A. Puente-Rodríguez, I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, H. Terashima-Marin, A simulated annealing hyper-heuristic for job shop scheduling problems, in: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 57–64.

[3] R. Bai, E. K. Burke, G. Kendall, B. McCollum, A simulated annealing hyper-heuristic for university course timetabling, in: Practice and Theory of Automated Timetabling VI, Springer, 2006, pp. 345–350.

[4] G. Kendall, N. M. Hussin, 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, Springer, 2005, pp. 270–293.

[5] K. Z. Zamli, B. Y. Alkazemi, G. Kendall, A Tabu Search hyper-heuristic strategy for *t-way* test suite generation, Applied Soft Computing 44 (2016) 57–74.

[6] H. Jiang, J. Qiu, J. Xuan, A hyper-heuristic using GRASP with path-relinking: A case study of the nurse rostering problem, Journal of Information Technology Research 4 (2) (2011) 31–42.

[7] B. Hanousková, Travelling salesman challenge 2.0 wrap-up. tech community behind kiwi.com, online available at: `https://code.kiwi.com/travelling-salesman-challenge-2-0-wrap-up-cb4d81e36d5b`, accessed on March 2023 (2018).

[8] E. D. Ahmad, A. Muklason, I. Nurkasanah, 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), IEEE, 2020, pp. 217–221.

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

[10] Y. Pylyavskyy, A. Kheiri, L. Ahmed, A reinforcement learning hyper-heuristic for the optimisation of flight connections, in: 2020 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2020, pp. 1–8.

[11] J. Cirasella, D. S. Johnson, L. A. McGeoch, W. Zhang, The asymmetric traveling salesman problem: Algorithms, instance generators, and tests, in: Algorithm Engineering and Experimentation: Third International Workshop (ALENEX), Springer, 2001, pp. 32–59.

[12] A. Kheiri, E. Keedwell, Selection hyper-heuristics, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, 2022, pp. 983–996.

[13] J. H. Drake, A. Kheiri, E. Özcan, E. K. Burke, Recent advances in selection hyper-heuristics, European Journal of Operational Research 285 (2) (2020) 405–428.

[14] M. Bellmore, G. L. Nemhauser, The traveling salesman problem: a survey, Operations Research 16 (3) (1968) 538–558.

[15] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis II, An analysis of several heuristics for the traveling salesman problem, SIAM Journal on Computing 6 (3) (1977) 563–581.