A Reinforcement Learning Hyper-heuristic for the Optimisation of Flight Connections

Yaroslav Pylyavskyy*, Ahmed Kheiri*, Leena Ahmed†
*Lancaster University, Department of Management Science, UK
{y.pylyavskyy, a.kheiri}@lancaster.ac.uk
†Cardiff University, School of Computer Science, UK
ahmedlh@cardiff.ac.uk

***Abstract*—Many combinatorial computational problems have been effectively solved by means of hyper-heuristics. In this study, we focus on a problem proposed by Kiwi.com and solve this problem by implementing a Reinforcement Learning (RL) hyper-heuristic algorithm. Kiwi.com proposed a real-world NP-hard minimisation problem associated with air travelling services. The problem shares some characteristics with several TSP variants, such as *time-dependence* and *time-windows* that make the problem more complex in comparison to the classical TSP. In this work, we evaluate our proposed RL method on kiwi.com problem and compare its results statistically with common random-based hyper-heuristic approaches. The empirical results show that RL method achieves the best performance between the tested selection hyper-heuristics. Another significant achievement of RL is that better solutions were found compared to the best known solutions in several problem instances.**

***Index Terms*—Hyper-heuristics, Metaheuristics, TSP**

I. Introduction

Over the last few decades, the airline industry has experienced a vast growth, and air travel is considered by far the best option for long distance journeys. Every year more than 30 million flights are scheduled worldwide, and this number keeps increasing [1]. With such complex air network, it is extremely hard for air travellers to find the cheapest possible connection between two airports, and this becomes even more challenging for a multi-city journey. As a result, several online travel agencies have developed search engines to provide cheap flights for travellers, such as Kiwi.com, Skyscanner.net, and many others. Some of these agencies in an attempt to improve their customer service have launched different challenges and projects, which have drawn the attention of many researchers from optimisation and computer science fields. OpenFlights.org is an online searching tool associated with air travel, which launched the *Air-Travelling Salesman* project [2]. Similarly, Kiwi.com ran the *Travelling Salesman Challenge* in 2017, which contributed to the development of the current algorithm Kiwi.com uses, named *NOMAD* [3]. In 2018, Kiwi.com proposed the *Travelling Salesman Challenge 2.0*, which is the subject of this study. The challenge aims to determine the cheapest connection between specific areas.

While some similarities with the ordinary TSP exist, this problem is much more complex and essentially different. It is *time-dependent* with *time-windows*, *asymmetric* and *generalised* problem described as follows: Given a list of areas, a list of airports for each area, the travelling costs between the listed airports for the corresponding days, and the starting city, the objective is to minimise the total travelling cost by finding the cheapest possible journey that visits exactly one city of each given area and terminates at the starting area. Combinatorial problems, such as this, are of a great interest for practitioners and researchers. Such problems are solved with exact methods, heuristic algorithms or a combination of them. Exact methods provide optimal solutions, but they have the limitation of performing inefficiently in terms of time complexity. This phenomena is more profound in large size problems resulting in an unreasonably computational time [4]. Heuristics are used to improve computational time efficiency and provide decent or near-optimal solutions [5]. Collaborative combinations make use of both exact and heuristic algorithms. Particularly, these methods are combined in order to exchange information. The work in [6] refers to such combination used to solve TSP to optimality. In this study, Kiwi.com problem is solved by means of selection hyper-heuristics.

II. Background

A. Travelling Salesman Problem

Travelling Salesman Problem (TSP) is a well-known $\mathcal{NP}$-hard problem and has been studied by many researchers due to its various applications in real-world problems. It was firstly defined by the two mathematicians William R Hamilton and Thomas Kirkman in the 19th century, and it is until now considered one of the most challenging problems in operations research [7]. Over the last few decades, many variations of TSP have been created in order to fit with more real-world problems. Some of them are [8]:

- General: The distance or cost is arbitrarily assigned between cities.
- Metric: The distance or cost is metric satisfying the triangle inequality; $\forall x, y, z \in X, d(x, z) \leq d(x, y) + d(y, z)$.
- Symmetric TSP (STSP): The cost of travelling from city i to city j is the same as travelling from city j to city i.
- Asymmetric TSP (ATSP): The cost of travelling from city i to city j is not the same as travelling from j to i.
- TSP with multiple visits (TSPM): It is allowed to visit cities more than once.
- Multiple TSP: Multiple salesmen are allowed.

- Open tour TSP: The salesman does not have to end the tour from the point it started.
- Time-dependent TSP (TD-TSP): The travel cost depends on distance and the day of travel [9].
- Generalised TSP (GTSP): Cities are divided into clusters and salesman visits exactly one city of each cluster [10].
- TSP with time-windows (TSPTW): Salesman must visit each city within a specified time window [11].

Kiwi.com problem shares some characteristics with several TSP variants. One can think of the Kiwi.com problem as a generalised, asymmetric, time-dependent TSP with time-windows, combining four variants of TSP problems, and hence making this problem difficult enough to solve.

B. Optimisation in Air Travel

Over the last few years, more researchers became interested in applying optimisation techniques in air travel. One of the main reasons for this growing interest is that air travelling nowadays, is one of the most important and affordable transportation means for the majority of the global population. Air travellers usually refer to online search engines to find the cheapest trips. While finding the cheapest flight connection between two airports might not be difficult, finding the cheapest possible multi-city trip is extremely complicated. Therefore, several studies have focused on finding better quality solutions for multi-city trips by air travel.

Kiwi.com released a challenge in 2017, called *Travelling Salesman Challenge* [12]. This was a previous version of the current Kiwi.com challenge, *Travelling Salesman Challenge 2.0*, which is the subject of this study. The main difference between the two challenges, is that while in the former version the traveller had to visit a number of cities, in the current version cities are divided into a number of areas from which exactly one city has to be visited. In [12], Simulated Annealing (SA), Ant Colony Optimisation (ACO), and a hybrid algorithm combining SA and ACO were applied to solve the problem on large instances up to 100 cities. Additionally, they parallelised the ACO algorithm and meta-optimised the parameters of SA and ACO algorithms with the aid of a genetic algorithm. To improve further the quality of their solutions, they applied a K-opt technique to the good solutions obtained by the algorithms. They used greedy search and backtracking as benchmarks and compared all these algorithms. Their results showed that their hybrid algorithm outperformed the other algorithms. The K-opt and meta-optimisation techniques only improved solutions of medium size instances.

The *Flying Tourist Problem* (FTP) [3] involves a tourist who wants to make a multi-city flight journey with the best possible schedule, route and set of flights. The tourist should also spend a number of days in each city. The objective of this problem is not only to minimise the total cost but also the flight duration and the cost-duration combination. The study in [3] tested three well-known algorithms on their problem: SA, ACO and Particle Swarm Optimisation (PSO). PSO proved to be the most effective algorithm between them. Their proposed algorithm was compared to the current Kiwi.com algorithm (*NOMAD*), and the results provided cheaper solutions in 95% of the times. However, their proposed algorithm was only tested for up to 20 cities in total.

The work in [9] discussed the time-dependent ATSP with time-window and precedence constraints in air travel. However, the traveller visits countries instead of cities, where some countries might have to be visited straightaway after some other countries are visited. The traveller must spend one week in each country and then travel to the next one. The authors suggested two Local Search algorithms for the problem: LS with Swap Operator, and LS with Insert Operator. A modified version of the Nearest Neighbour was used as a benchmark. Both LS algorithms performed better than the Modified Nearest Neighbour when tested on instances of up to 20 countries. Their results suggest that LS with Insert is more appropriate for instances with easy constraints, and LS with Swap is better for hard constraints. This is mainly because the Insert Operator has a greater probability of generating an infeasible solution when applied on instances with hard constraints. Specifically, when Insert Operator is used, all travel dates in the solution have to change, and thus the infeasibility chance increases.

Another interesting study is the *Air-Travelling Salesman* project launched by the OpenFlights.org [2]. In this project, they are given a list of airports to visit and the goal is to find the best possible route that minimises the travel distance. For some airports, a direct flight does not exist and thus an intermediate airport might need to be visited. The Nearest Neighbour (NN) heuristic algorithm with a 3-opt Swap Operator was used to solve the problem. The swap operator simply changes the visiting order of 3 airports in the solution. The algorithm was tested on several STSP and ATSP instances and the results showed that the algorithm performed well on both types of instances with small number of cities. However, the algorithm did not perform as well as expected for larger size problems, specifically on ATSP instances.

The *Travel Itinerary Problem* was studied in [13], which involves a traveller who wants to make a multi-city trip without having any preference about the means of transportation. The aim is to find the cheapest possible combination of itineraries for the whole trip within the time frame specified. The authors solved the problem via IP and an implicit enumeration algorithm. In addition, the authors developed a smart travel system, where users input their travel preferences and the optimal tour is returned by using real online data. The algorithm was tested on real case studies based on real-life transport data.

C. Hyper-heuristics

Many hard combinatorial computational problems have been effectively solved by means of simple heuristics and metaheuristic algorithms. Yet, these algorithms, such as genetic algorithms, simulated annealing, and tabu search, share the disadvantage of the need of advanced knowledge in order to adjust them when the problem changes [14], [15]. Because of these weaknesses, many researchers focused on building methods which can automatically design heuristics and are

general enough to be applied in a wide range of problems without much expertise intervention. This idea was firstly stated in 1960s but was not implemented until 2000 [16]. Nowadays, this automated process is known as *hyper-heuristic* and is widely applied to many combinatorial problems.

Hyper-heuristics are divided into two main categories; 1) *generation* hyper-heuristics which generate new heuristics and 2) *selection* hyper-heuristics which select a heuristic for application from a set of low level heuristics [17]. In addition to this, hyper-heuristics are further divided into two subcategories; 1) *constructive* methods which construct a solution from scratch by implementing a set of heuristics at different phases of the construction process and 2) *perturbative* methods which use a complete initial solution and apply a set of heuristics in a perturbative way to improve this solution [18]. Selection perturbative hyper-heuristics, which are the focus of this work, are methods that choose a heuristic from a set of low level heuristics, and apply it to the solution in hand for improvement. This particular method is very forceful since certain heuristics might perform better than others at certain stages. Hence, the sequence in which the low level heuristics are applied is essential, as different sequences can vary in their performances [19]. Hyper-heuristics are capable of learning which sequences or heuristics are more effective during the search process by using feedback mechanisms. Based on the origin of feedback, the learning process is classified as *online* or *offline*. In the former class, the algorithm directly learns during the search while solving the main problem, whereas in the latter class the algorithm learns by solving a set of training instances which prepare the algorithm to effectively encounter new instances [16]. Hyper-heuristics operate without the need of any information regarding the functionality of the low level heuristics. They only require the type of optimisation (i.e. minimise or maximise) and the number of the low level heuristics [20]. In return, they provide useful feedback such as the amount of time required to apply a heuristic, the utilisation rate of each heuristic, the change in the objective function. These are vital information for the learning process [16].

The selection hyper-heuristic framework usually has two sequential steps: 1) *heuristic selection* and 2) *move acceptance* [16]. The first step is accountable for selecting a low level heuristic from the set of low level heuristics and applying it to the solution, whereas the second step is a decision regarding the acceptance or rejection of the new solution. Some examples of heuristic selection strategies developed previously [15]: Simple Random (SR) which uses a uniform probability distribution to randomly select a low level heuristic at each step. Random Descent (RD) that selects a low level heuristic randomly and applies it repeatedly as long as an improvement is found. Random Permutation (RP) which selects a low level heuristic one at a time from an initial generated permutation of the low level heuristics. Random Permutation Descent (RPD) that forms an initial permutation of the low level heuristics similar to RP, but applies each low level heuristic repeatedly until no improvement is found similar to RD. Finally, Reinforcement Learning (RL) that assigns an initial score to each low level heuristic in the initialisation of the algorithm. These scores are increased or decreased based on the improvement or deterioration of the solution respectively. RL chooses and applies the heuristic with the best score at each stage [20]. The study in [20] highlighted a main drawback of the RL strategy. While at early stages some heuristics might be more effective than others, on later stages this condition might change leading RL to repeatedly select ineffective heuristics until their score decreases sufficiently. To overcome this issue, the authors recommend to place any non-improving heuristics in a tabu list, until a better solution is found. The work in [18] proposed another way to overcome this problem, through utilising shorter term memories. This way, the algorithm will gradually forget any repetitive improvements performed by certain heuristics at early stages.

Several move acceptance strategies have been developed in previous studies [17], for example, Improving or Equal (IE) accepts non-worsening moves; and Great Deluge (GD) accepts all moves within a dynamic level of the objective value. The initial level is equal to the initial objective value and is linearly updated towards the expected objective value by using the formula: $\tau_t = f_0 + \Delta F \times (1 - \frac{t}{T})$ where τ_t is the threshold level at time t, T is the time limit, ΔF is the expected range for the maximum change in the objective value, and f_0 is the final expected objective value.

III. Problem description

Kiwi.com problem seeks to find the best possible flight routes between specifically given areas in order to minimise the travelling cost. Each area includes a set of cities from which exactly one has to be visited everyday and the cost of travelling between areas is different based on the direction and day of travel. During the trip, it is not possible to arrive to a city and then continue the trip by departing from another city of the same area. The final destination of the trip is the starting area but not necessarily the starting city.

Mathematically, let $Area = \{area_1, area_2, \ldots, area_n\}$ be a set of n areas, where each area $r \in Area$ is composed of a set of airports $\{airport_1, airport_2, \ldots\ airport_m\}$; and let c_{ij}^d be the flight cost between the departure airport i and the arrival airport j on day d, which has two properties: c_{ij}^d is not necessarily equal to c_{ji}^d; and for $d1 \neq d2$, c_{ij}^{d1} is not necessarily equal c_{ij}^{d2}. In some instances there are multiple flights between the same cities for the same day with different costs (i.e. different airline companies for the same connection). The objective of the problem is to find the best possible flight route that connects all the given areas and minimises the cost within the time given, subject to:

- Trip starts from a given city.
- Exactly one city of our choice is visited in each area.
- A different area is visited on daily basis.
- The arrival city of each area is also the departure city.
- Trip terminates in any city of the starting area.

A simple problem instance consisting of 4 areas and 8 airports is presented in Figure 1. The 4 areas are; Greece, Italy, Spain, and the UK, where each area contains two cities; Athens

Fig. 1: Simple problem instance with a possible solution

and Thessaloniki, Rome and Milan, Madrid and Barcelona, and London and Liverpool, respectively. The trip begins from Athens to Madrid, then continues from Madrid to London, then from London to Rome and ends in Thessaloniki. Notice that in this trip the starting and finishing airports are different, but the trip is acceptable since it ends in the starting area.

Kiwi.com provided 14 instances in total, covering a range from 10 to 300 areas[1]. The time limits suggested by Kiwi.com for the instances are the following:

- 3 seconds for small instances (number of areas $\leq$ 20 & number of airports $<$ 50)
- 5 seconds for medium instances (number of areas $\leq$ 100 & number of airports $<$ 200)
- 15 seconds for large instances (number of areas $>$ 100)

IV. Methodology

Hyper-heuristics are fast and relatively easy to implement. They exploit the search space of low level heuristics, and are capable of learning during the search by gathering information on the performance of the low level heuristics. These features make hyper-heuristics an ideal candidate to solve Kiwi.com problem and overcome the time limit challenges, while providing good solutions. In total, six selection hyper-heuristic algorithms with different combinations of selection and move acceptance methods are implemented. The solution is represented as a two dimensional vector. The first dimension $S_{area} = \{area_1, area_2 \dots area_n\}$ indicates the index of the area, and the second dimension $S_{airport} = \{airport_1, airport_2 \dots airport_m\}$ indicates the airports of the corresponding area. Each pair $(area, airport)_i$ represents the area and the corresponding airport visited in day i. Initially, a random solution is produced and further improved using a local search procedure to ensure its feasibility.

[1]Datasets are available at https://code.kiwi.com/ and can also be downloaded from https://ahmedkheiri.bitbucket.io/publications/KIWI.zip

A. Low Level Heuristics

- Swap: randomly selects two areas in the solution and their corresponding airports and swaps them.
- Insert: randomly selects an area in the solution and a position within the solution. Then the area is inserted to the determined position by moving all the areas between the previous area position and the determined position to the left or right as required.
- Reverse: randomly selects two areas in the solution and reverses between these two areas.
- Change Airport: randomly selects an airport and replaces it with another airport of the corresponding area.

B. Hyper-heuristics for Kiwi.com

In total, six selection hyper-heuristics are implemented: SR-IE, SR-GD, RD-IE, RP-IE, RPD-IE, and RL. After applying the selected low level heuristic (LLH), if the new solution is feasible, its cost is compared with the cost of the previous solution. The new solution is saved depending on the move acceptance criterion. Otherwise, it is scrapped and the previous solution is restored. In RL method, the low level heuristics compete with each other for selection and the best one is selected. Each low level heuristic is assigned a score, and based on its performance is either rewarded or penalised. In addition, any low level heuristic that produces an infeasible solution is penalised higher as suggested in [18]. All rewards and penalties are weighted by a factor based on the iteration number. This idea is introduced to avoid over-rewarding the low level heuristics in the early phase of the search process, as mentioned in [20], and maintain a fair reward and penalty system. However, penalties are less weighted than rewards, this is for giving the low level heuristic a second chance. In this method, all improving moves are accepted, and the worsening solutions are accepted in some scenarios. The procedure of accepting a worsening solution is controlled by a counter β, which keeps track of the consecutive number of iterations without improvements. β increases more when an infeasible solution is generated. α is a parameter which represents the tolerance number for consecutive non-improvements. Whenever the solution is not much worse than the previous, and $\beta > \alpha$, a worsening solution is accepted and all scores of LLHs are set to their default values. This idea is implemented so as to escape local minima.

RL method is described with details in Algorithm 1. Table I lists the values of the parameters used in RL method.

TABLE I: The algorithm parameters and the chosen values

Parameter	Value
α	10^4
$reward$	0.003
$penalty_f$	0.000625
$penalty_w$	0.0005
$LHHS$ initial values	[0.5, 0.5, 0.5, 0.5]

Algorithm 1: Reinforcement Learning

Let LLH be the set of LLHs
Let $LLHS$ be the scores of LLHs
Let j be the current iteration
Let β be the number of iterations without improvement
Let α be a tolerance number for non-improvements
Let $reward$ be the factor for rewarding LLH
Let $penalty_w$, $penalty_f$ be the factor for penalising LLH generating worsening, or non-feasible solutions
Let S, S_{new}, S_{best} be the current, new, and best solutions
Let $f(S)$ be the cost of the solution S
Let $F(S)$ be the feasibility of the solution S
$S \leftarrow$ Initialise();
$\beta \leftarrow 0$;
repeat
  $LLH_i \leftarrow$ SelectBestScore($LLHS$);
  $S_{new} \leftarrow$ ApplyLLH(LLH_i, S);
  if $F(S_{new}) = 0$ **then**
    if $f(S_{new}) < f(S)$ **then**
      $S \leftarrow S_{new}$;
      $LLHS_i \leftarrow LLHS_i + j \times reward$;
      $\beta \leftarrow 0$;
      if $f(S_{new}) < f(S_{best})$ **then**
        $S_{best} \leftarrow S_{new}$;
    else if $\beta > \alpha$ **&&** $f(S_{new}) < 4 * f(S)$ **then**
      $S \leftarrow S_{new}$;
      $LLHS \leftarrow [0.5, 0.5, 0.5, 0.5]$;
      $\beta \leftarrow 0$;
    else
      $LLHS_i \leftarrow LLHS_i - j \times penalty_w$;
      $\beta \leftarrow \beta + 1$;
  else
    $LLHS_i \leftarrow LLHS_i - j \times penalty_f$;
    $\beta \leftarrow \beta + 10$;
until *TimeLimit*;
return S_{best}

V. Empirical Results

In this section we present a performance comparison between the tested selection hyper-heuristics on a number of selected instances, and further analyse the best method. The experiments were performed on an Intel Core i5 at 2.3GHz with memory of 8GB. Each method was executed for 30 runs with different random seed values and within the time limits specified by Kiwi.com and described in Section III. The experiments were performed on a subset of instances with varying sizes, where instances 1, and 3 are selected to represent the small size instances, instances 4, and 6 representing medium size instances, and instances 7, 9, 10, and 13 represent large size instances. Our results identified the success of RL method in most of the instances compared to the other tested selection hyper-heuristics. In order to confirm this, Mann-Whitney-Wilcoxon (MWW) test is conducted to identify any significantly different behaviour between RL and the other tested methods. In Table II, the results of the MWW test at 5% significance level are presented, where $<$ symbol indicates that RL generates statistically better results than the compared method, and $*$ symbol indicates that no significant difference exists. It can be observed from the table that RL generates better results for most of the instances, and these results are statistically significant. The only exception is instance 9, where RL outperformed only three tested methods. SR-IE is the most competing method to RL, and RL was still able to outperform it in six instances out of eight. This comparison indicates the success of RL over other randomised selection methods, proving the efficiency of the employed reinforcement learning technique, and the implemented rewarding and penalising scheme.

TABLE II: Mann–Whitney-Wilcoxon test of RL at 5% significance level

Name	SR-IE	SR-GD	RD-IE	RP-IE	RPD-IE
Instance-1	$<$	$<$	$<$	$<$	$<$
Instance-3	$*$	$<$	$<$	$<$	$<$
Instance-4	$<$	$<$	$<$	$<$	$<$
Instance-6	$*$	$<$	$<$	$<$	$<$
Instance-7	$<$	$<$	$<$	$<$	$<$
Instance-9	$<$	$<$	$<$	$*$	$*$
Instance-10	$<$	$<$	$<$	$<$	$<$
Instance-13	$<$	$<$	$<$	$<$	$<$

A. An Analysis of RL Method

RL method was further tested in all fourteen instances, where 30 runs were conducted per instance. Table III displays these results. The first four columns of the table are the characteristics of each instance, followed by the time limit in seconds for performing a single run. A summary statistics are presented in the following columns, and the last column presents the best known cost. The results show that RL performed well on small and medium size instances finding better cost values than the best known, except for instance 6. RL also succeeded in finding better cost values in instances 7 and 9 than the best known.

The following analysis is performed on the selected subset of instances described above. In Figure 2, the cost versus time is illustrated when the low level heuristics are applied jointly and separately in small size instances. In addition, Figure 3 shows the utilisation rate of each low level heuristic when combined together. The plots indicate that in Instance-1, the best known solution is found by either combining the low level heuristics or Swap low level heuristic individually. Insert managed to reach close enough to the best known solution, while Reverse had the worst performance. Change Airport is not shown in the plot, as Instance-1 has only one airport in each area. The utilisation rate plot shows that Swap had the most contribution. Although, Reverse performed worse than

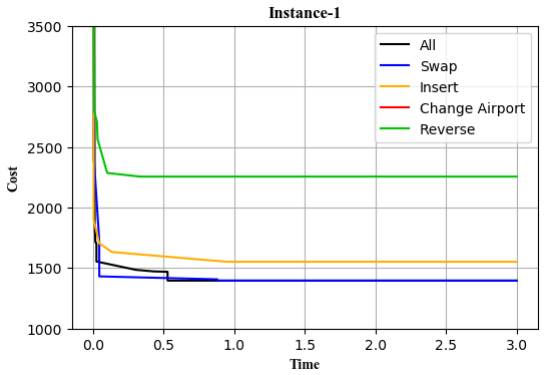
TABLE III: The performance of RL algorithm over 30 runs. The average, best and standard deviation of the thirty runs and the best known cost for each instance are reported. Best values per each instance are highlighted in bold

Name	areas	airports	airports in areas	time (s)	best	average	std.	best known
Instance-1	10	10	1 (min) – 1 (max)	3	**1396**	1396	0	**1396**
Instance-2	10	15	1 (min) – 2 (max)	3	**1498**	1498	0	**1498**
Instance-3	13	38	1 (min) – 6 (max)	3	**7672**	7672	0	**7672**
Instance-4	40	99	1 (min) – 5 (max)	5	**13952**	14017.1	61.66	14024
Instance-5	46	138	3 (min) – 3 (max)	5	**690**	697.1	5.96	698
Instance-6	96	192	2 (min) – 2 (max)	5	2610	3038.8	199.42	**2159**
Instance-7	150	300	1 (min) – 6 (max)	15	**30937**	31441.2	261.35	31681
Instance-8	200	300	1 (min) – 4 (max)	15	4081	4123.4	26.96	**4052**
Instance-9	250	250	1 (min) – 1 (max)	15	**75604**	83364.5	10996.20	76372
Instance-10	300	300	1 (min) – 1 (max)	15	58304	64828.9	2881.00	**21167**
Instance-11	150	200	1 (min) – 4 (max)	15	59361	63679.7	2045.50	**44153**
Instance-12	200	250	1 (min) – 4 (max)	15	86074	91068.3	2907.99	**65447**
Instance-13	250	275	1 (min) – 3 (max)	15	166543	173575.0	5230.30	**97859**
Instance-14	300	300	1 (min) – 1 (max)	15	198787	209428.9	4784.02	**118811**

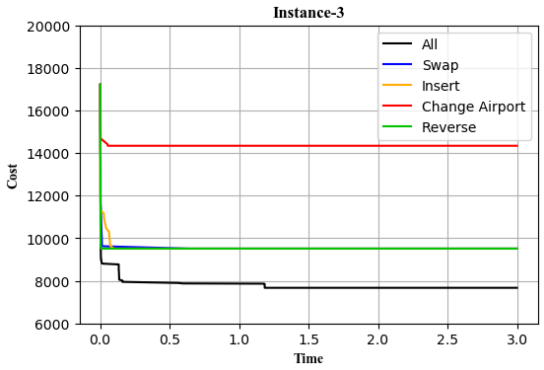
Insert when applied separately, it contributed more than Insert when all low level heuristics were combined. In Instance-3, the best known solution is reached only when all low level heuristics are combined. Swap, Insert, and Reverse low level heuristics show similar performance, by finding a cost value not far from the best known solution. Change Airport did not perform well when applied separately. The utilisation rate plot indicates that Change Airport had the biggest contribution among all low level heuristics in reaching the best known solution. Reverse and Swap had almost the same contribution, whereas Insert had the least.

In Figure 4, low level heuristics are applied jointly and separately in the medium size instances 4 and 6. Furthermore, Figure 5 depicts the utilisation rate of each low level heuristic. Change Airport did not improve the solution when applied separately. Overall, the plots show that in Instance-4, the best solution is found by the combination of all low level heuristics. Insert and Reverse have a similar performance, while Swap is slightly worse unlike in the small instances. Despite the poor performance of Change Airport individually, the utilisation rate plot in Figure 5 reveals that it has the largest contribution rate, followed by reverse and swap low level heuristics. In Instance-6, the best cost is achieved through the combination of all low level heuristics. When applied separately, Insert and Swap stood out from the rest of the low level heuristics. Likewise, as can be seen from Figure 5, Insert has the greatest utilisation rate followed by Swap. Change Airport performed poorly on this instance.

In Figure 7, the cost versus time is displayed when the low level heuristics are applied jointly and separately. Also, Figure 6 illustrates the utilisation rates of the low level heuristics. The plots suggest that in a similar manner to the medium size instances, Change Airport does not improve the cost when applied separately. In Instance-7, Figure 7 shows that the best solution is achieved via a combination of low level heuristics. Insert yields the second best solution and Reverse and Swap follow respectively. As shown in Figure 6, Reverse contributes the most in cost reduction, followed by Insert and Swap respectively. In Instance-9, Swap achieved a



(a)



(b)

Fig. 2: Plots of cost versus time with combined low level heuristics and without for small size instances

better solution than the combined low level heuristics. The rest of the low level heuristics did not contribute in decreasing the cost when applied separately. Furthermore, Figure 6 shows that Swap is also very successful when combined with the other low level heuristics achieving the highest utilisation. Instance-9 has only one airport in each area and hence, Change Airport is not utilised. In Instance-10, Insert obtained a better solution than combined low level heuristics as shown in Figure 7.

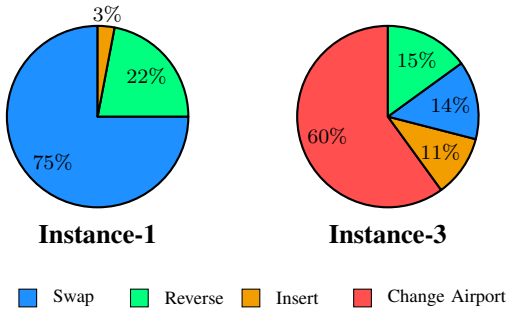
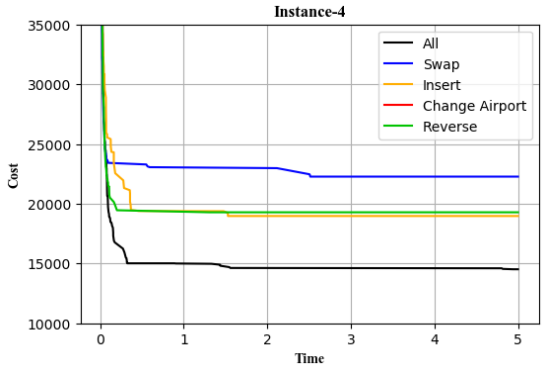
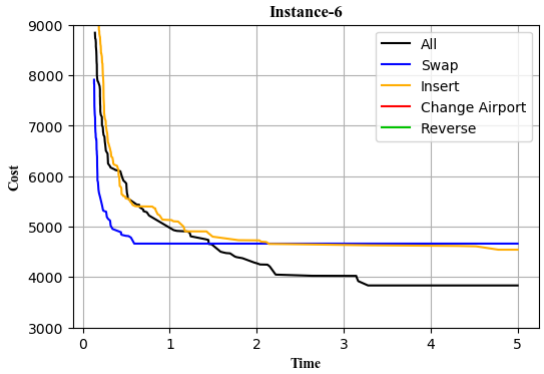


Fig. 3: Utilisation rate of each low level heuristic for small size instances



(a)



(b)

Fig. 4: Plots of cost versus time with combined low level heuristics and without for medium size instances

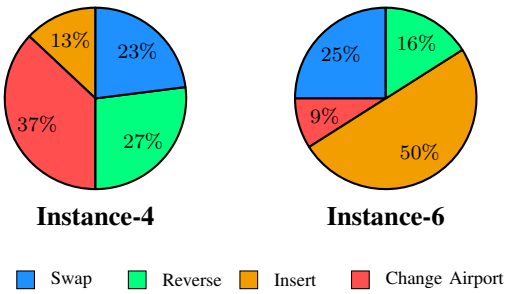


Fig. 5: Utilisation rate of each low level heuristic for medium size instances

The third best solution was achieved by Swap. A similar behaviour is noticed in Figure 6, where Swap and Insert are the main contributors when all low level heuristics are combined. Reverse contributes by only 1%. Instance-10 has only one airport in each area and Change Airport is therefore not utilised. In Instance-13, Figures 6 and 7 show the successful application of Swap individually, achieving better cost than combining the low level heuristics, and collectively with the other low level heuristics achieving the highest utilisation. Change Airport contributes successfully in this instance with 24% utilisation rate.

VI. Conclusion

In this work, we solved Kiwi.com problem by the means of selection hyper-heuristics. Six selection hyper-heuristics were tested on a dataset of fourteen instances of varying sizes provided by Kiwi.com. The developed selection hyper-heuristics controlled a set of four low level heuristics (Swap, Insert, Change Airport, and Reverse). A Reinforcement Learning hyper-heuristic (RL) is proposed in this work with specifically designed set of parameters to escape local minima, and compared with the other implemented selection hyper-heuristics. The empirical results showed the success of RL method, achieving the best performance with a statistical significance for most of the instances. RL method also proved its efficiency in comparison to the best known solutions, where it successfully found better solutions than the best known in four instances. The proposed RL method can be further investigated in future research in other problem domains to prove its generality. Moreover, Kiwi.com problem can be extended by adding further constraints such as allowing multiple internal trips to different cities within one area before moving to the next area.

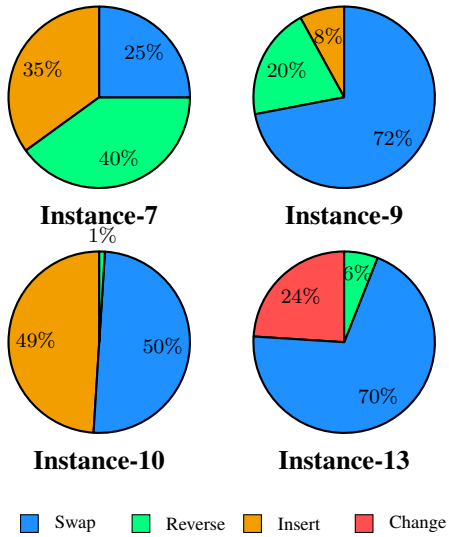


Fig. 6: Utilisation rate of each low level heuristic for large size instances

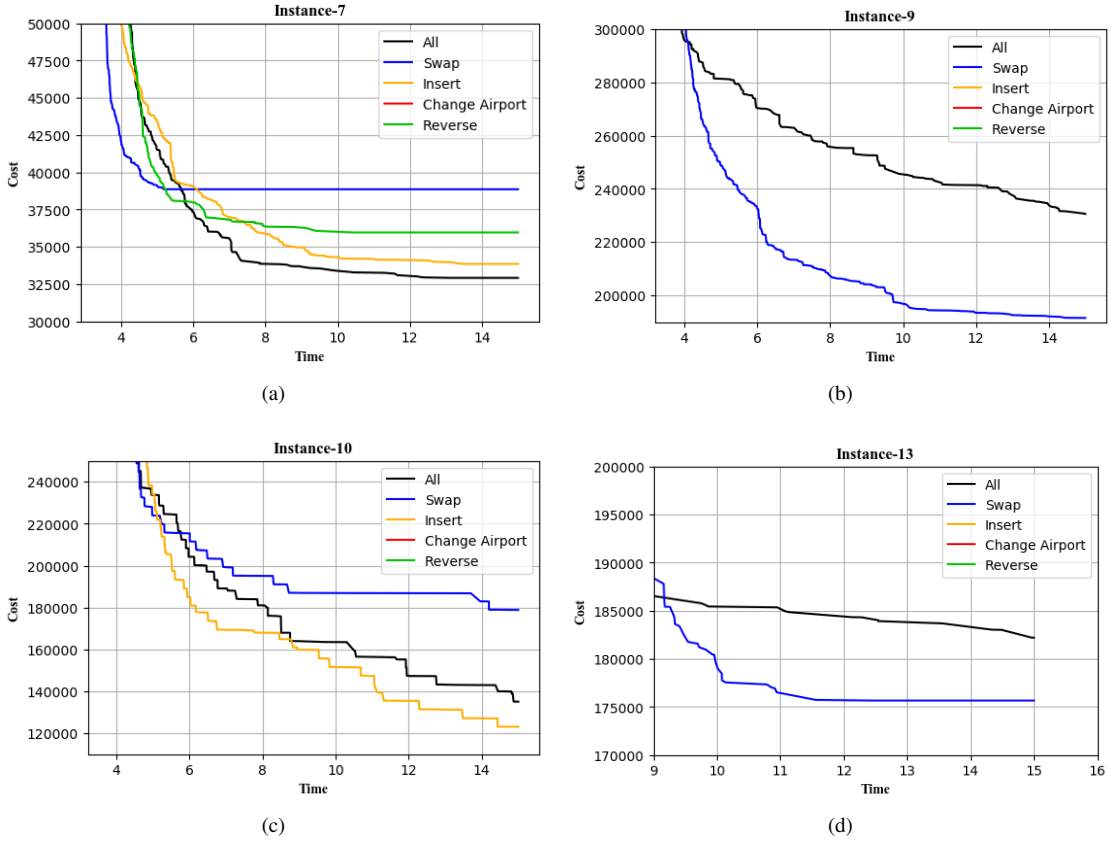


Fig. 7: Plots of cost versus time with combined low level heuristics and without for large size instances

REFERENCES

[1] M. C. D., "Computational complexity of air travel planning," 2003, public Notes on Computational Complexity. Retrieved June, 14, 2019 from http://www.demarcken.org/carl/papers/.

[2] OpenFlights, "The air-traveling salesman," 2015, retrieved June,13,2019 from https://sites.google.com/site/travellingcudasalesman/.

[3] R. Marques, L. Russo, and N. Roma, "Flying tourist problem: Flight time and cost minimization in complex routes," *Expert Systems with Applications*, vol. 130, pp. 172–187, 2019.

[4] M. Muneeb Abid and M. Iqbal, "Heuristic approaches to solve traveling salesman problem," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 15, pp. 390–396, 2015.

[5] I. Dumitrescu and T. Stützle, "Combinations of local search and exact algorithms," in *Applications of Evolutionary Computing*, S. Cagnoni, C. G. Johnson, J. J. R. Cardalda, E. Marchiori, D. W. Corne, J.-A. Meyer, J. Gottlieb, M. Middendorf, A. Guillot, G. R. Raidl, and E. Hart, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 211–223.

[6] J. Puchinger and G. R. Raidl, "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification," in *First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005*, J. R. Á. José Mira, Ed., vol. 3562. Springer-Verlag, 2005, pp. 41–53.

[7] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[8] R. Rasmussen, "TSP in spreadsheets - a guided tour," *International Review of Economics Education*, vol. 10, no. 1, pp. 94–116, 2011.

[9] T. Saradatta and P. Pongchairerks, "A time-dependent atsp with time window and precedence constraints in air travel," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 9, no. 2-3, pp. 149–153, 2017.

[10] P. C. Pop and S. Iordache, "A hybrid heuristic approach for solving the generalized traveling salesman problem," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. ACM, 2011, pp. 481–488.

[11] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon, "An optimal algorithm for the traveling salesman problem with time windows," *Operations Research*, vol. 43, no. 2, pp. 367–371, 1995.

[12] D. Duque, J. A. Cruz, H. L. Cardoso, and E. Oliveira, "Optimizing metaheuristics for the time-dependent tsp applied to air travels," in *Intelligent Data Engineering and Automated Learning – IDEAL 2018*, H. Yin, D. Camacho, P. Novais, and A. J. Tallón-Ballesteros, Eds. Springer International Publishing, 2018, pp. 730–739.

[13] X. Li, J. Zhou, and X. Zhao, "Travel itinerary problem," *Transportation Research Part B: Methodological*, vol. 91, pp. 332–343, 2016.

[14] A. Kheiri, "Heuristic sequence selection for inventory routing problem," *Transportation Science*, vol. 54, no. 2, pp. 302–312, 2020.

[15] L. Ahmed, C. Mumford, and A. Kheiri, "Solving urban transit route design problem using selection hyper-heuristics," *European Journal of Operational Research*, vol. 274, no. 2, pp. 545–559, 2019.

[16] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, in press.

[17] A. Kheiri and E. Keedwell, "A hidden Markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems," *Evolutionary Computation*, vol. 25, no. 3, pp. 473–501, 2017.

[18] R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum, "A simulated annealing hyper-heuristic methodology for flexible decision support," *4OR*, vol. 10, pp. 43–66, 2012.

[19] A. Kheiri and E. Keedwell, "A sequence-based selection hyper-heuristic utilising a hidden markov model," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. Association for Computing Machinery, 2015, pp. 417–424.

[20] K. Chakhlevitch and P. Cowling, *Hyperheuristics: Recent Developments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 3–29.