A Monte Carlo Tree Search for the Optimisation of Flight Connections

Arnaud Da Silva*, Ahmed Kheiri*
*Lancaster University, Department of Management Science, Lancaster LA1 4YX, UK
{a.dasilva, a.kheiri}@lancaster.ac.uk

***Abstract*—In 2017, Kiwi.com proposed the Traveling Salesman problem 2.0. Despite some similarities with the classic Traveling Salesman Problem (TSP), the problem is more complex. It can be characterised as an asymmetric, time-constrained and generalised TSP. Moreover, infeasibility adds further complexity to the problem as there are no flights available between specific airports at specific days. Exact methods often fail in solving these $\mathcal{NP}$-Hard problems. Therefore, alternative approaches, such as heuristics, are preferred to approach these problems. A Monte Carlo Tree Search (MCTS) was implemented to tackle Kiwi's problem, an algorithm traditionnaly used in board games, but here adapted to solve an air travel optimisation problem.**

The focus of this paper is on the first eight instances of the problem, without taking into account the time constraints set by Kiwi.com. The MCTS has been chosen for its proven effectiveness in handling complex and high dimensionnal search spaces.

***Index Terms*—Optimisation, Travelling Salesman Problem, Monte Carlo Tree Search, Air Travel Optimisation.**

I. Introduction

The number of flight connections keep increasing every year [1], more than 38 million flights have been scheduled in 2023. Therefore, creating a challenge for traveler's to find the best and cheapest flight connections for their specific journey, especially when one has to visit a big number of cities. Consequently, travel agencies have deployed online trip planner algorithms in order to find flights connection that match the traveler's requirements. Example of these are, Google Flights, OpenFlights.org, Skyscanner, Kayak and Kiwi.com.

These agencies have launched different challenges to create and build powerful trip planner algorithms. For instance, as mentionned in [2], OpenFlights.org launched the Air Travelling Salesman project. Furthermore, Kiwi.com has launched a project in 2017, called Traveling Salesman Challenge, where the current algorithm used by Kiwi.com was developed. In 2018, Kiwi.com launched a new challenge, the Traveling Salesman Problem 2.0 which is the focus of this study. Despite the large number of competitors in the challenge, there is limited literature on the methods used. The winners employed a Breadth-first search (BFS) algorithm in C++ [3], while other participants used heuristics like Simulated Annealing, Genetic Algorithms, and Reinforcement Learning. Only two papers were published ([2] and [4]), focusing on Local Search and Reinforcement Learning. These points motivated our choice to implement a novel solution, using a MCTS to solve this challenge.

The given problem is a variant of the well known TSP. It can be characteristed as a generalised, assymetric and time dependant TSP. A traveler has to visit a list of areas, one per day, given a starting airport and all the possible flight connections between these areas at different days. The goal is to determine what is the cheapest flights connection for the traveler to come back to the starting area. Regarding the number of possible journeys, solving this problem by exploring every single potential solution is impossible. This is why a heuristic approach is implemented.

II. Problem description

A. Description of the minimisation problem

Kiwi's traveler wants to visit N different areas in N days, let's denote A the set of areas the traveler wants to visit: $A = \{A_1, A_2, \dots A_N\}$ where each A_j is a set of airports in area j: $A_j = \{a_{j,1}, a_{j,2}, \dots, a_{j,k_j}\}$ where a_{j,k_j} being airports in area j and k_j is the number of airports in area j.

The traveler has to visit one area per day. He has to leave this area to visit a new area by flying from the airport he flew in. He leaves from a known starting airport and has to do his journey and come back to the starting area, not necessarly the starting airport. There are flight connections between different airports, with different prices depending on the day of the travel: we can write c_{ij}^d the cost to travel from $city_i$ to $city_j$ on day d. We do not necessarly have $c_{ij}^d = c_{ji}^d$ neither $c_{ij}^{d_1} = c_{ij}^{d_2}$ if $d_1 \neq d_2$. Furthermore, $\exists d, c_{ij}^d = \infty$ i.e. there is no flight connection between city i and j at day d.

The aim of the problem is to find the cheapest route for the traveler's journey.

The problem itself had not been mathematically defined in previous research. Hence the problem can be formulate as follow:

- $\mathcal{A} = \{1, 2, \dots, N\}$: Set of areas.
- $A_j = \{a_{j,1}, a_{j,2}, \dots, a_{j,k_j}\}$: Set airports in area $j \in \mathcal{A}$.
- $\mathcal{D} = \{1, 2, \dots, N\}$: Set of days.
- $U_d \subseteq \mathcal{A}$: Set of areas that have not been visited by the end of day d.

Parameters and variables

- c_{ij}^d: Cost to travel from airport i to airport j on day $d \in \mathcal{D}$.
- x_{ij}^d: Binary variable which is 1 if the traveler flies from airport i to airport j on day d, and 0 otherwise.
- v_j^d: Binary variable which is 1 if area j is visited on day d, and 0 otherwise.

Objective Function

The goal is to minimise the journey's total travel cost:

$$\min\left(\sum_{d=2}^{N-1}\sum_{i\in\bigcup_{k=2}^{N-1}A_k}\sum_{j\in\bigcup_{k=3}^{N}A_k}c_{ij}^{d}x_{ij}^{d}\right.$$
$$\left.+\sum_{j\in A_1}c_{S_0,j}^{1}x_{S_0,j}^{1}+\sum_{i\in A_N}\sum_{j\in A_1}c_{ij}^{N}x_{ij}^{N}\right)$$

Constraints

- Starting at the known starting airport S_0 and take an existing flight connection: $\sum_{j\in A_1}x_{S_0,j}^{1}=1$ and $\forall d\in\mathcal{D}, c_{S_0,j}^{d}\in\mathbb{R}^{+*}$
- Visit exactly one airport in each area each day: $\sum_{i\in A_d}\sum_{j\in A_{d+1}}x_{ij}^{d}=1 \quad \forall d\in\{1,2,\ldots,N-1\}$
- Ensure the traveler leaves from the same airport they arrived at the previous day: $\sum_{k\in A_d}x_{ik}^{d} = \sum_{k\in A_{d-1}}x_{ki}^{d-1} \quad \forall i\in\bigcup_{j=1}^{N}A_j, \forall d\in\{2,3,\ldots,N\}$
- Return to an airport in the starting area on the final day with an existing flight connection: $\sum_{i\in A_N}\sum_{j\in A_1}x_{ij}^{N}=1$ and $\forall(i,j)\in A_N\times A_1, c_{i,j}^{N}\in\mathbb{R}^{+*}$
- Ensure each area is visited exactly once: $\sum_{d\in\mathcal{D}}v_j^d=1 \quad \forall j\in\mathcal{A}$
- Update the unvisited list: $v_j^d=1 \implies j\notin U_d \quad \forall j\in\mathcal{A}, \forall d\in\mathcal{D}$
- Ensure a flight on day d between i and j exists only if the cost exists and j is in the unvisited areas on day d: $x_{ij}^{d}\le c_{ij}^{d}\cdot v_j^d \quad \forall i,j\in(\bigcup_{j=1}^{N}A_j)^2, \forall d\in\mathcal{D}$ and $x_{ij}^{d}\le v_j^d \quad \forall j\in\bigcup_{j=1}^{N}A_j, \forall d\in\mathcal{D}$
- Binary variable constraints: $x_{ij}^{d}\in\{0,1\} \quad \forall(i,j)\in(\bigcup_{j=1}^{N}A_j)^2, \forall d\in\mathcal{D}$ and $v_j^d\in\{0,1\} \quad \forall j\in\mathcal{A}, \forall d\in\mathcal{D}$

B. Instances description

We are given a set of 14 Instances $I_n=\{I_1,I_2,...,I_{13},I_{14}\}$ to solve this challenge. Every instances has the same overall structure.

For every instance I_i, we know what connections exist between two airports for a specific day and the associated cost. There might be in some instances flights connections at day 0, this means these connections exist for every day of the journey at the same price.
Furthermore, there could be multiple same flight connections at a specific day but with different prices. Therefore, only the more relevant connections i.e. the flight connection with the lowest fare is considered.

III. Methodology

In this section the different steps of the MCTS won't be developed, however different policies of the algorithm are detailed.

A. Selection policy

The selection phase of the MCTS algorithm uses the Upper Confidence Bound (UCB) strategy to balance exploration and exploitation. This involves selecting the node that minimises the UCB or the UCB1-Tuned score, calculated as:

$$UCB=\overline{X}_i+C_p\sqrt{\frac{2\ln N}{n_i}}$$

$$UCB1T=\overline{X}_i+\sqrt{\frac{\ln N}{n_i}\min\left(\frac{1}{4},\mathrm{Var}(X_i)+\sqrt{\frac{2\ln N}{n_i}}\right)}$$

Where:

- $\overline{X}_i$: Average reward of node i.
- N: Total number of visits to the root node.
- n_i: Number of visits to node i.
- C_p: Exploration parameter
- $\mathrm{Var}(X_i)$: Variance of the rewards at node i, representing the variability of the rewards.

The UCB balances its exploration with the coefficient C_p, empirically $C_p=\sqrt{2}$. The term $C_p\sqrt{\frac{2\ln N}{n_i}}$ adds a confidence interval to the average reward, which encourages exploring less-visited nodes when $C_p>0$. When $C_p=0$, the tree search explores less but exploits more of the known part that seems promising for the problem in the tree. The UCB1-Tuned balances its exploration with $\min\left(\frac{1}{4},\mathrm{Var}(X_i)+\sqrt{\frac{2\ln N}{n_i}}\right)$, making the UCB1-Tuned more adaptable to environments with varying reward distributions. The C_p coefficient can also be considered in the UCB1-Tuned's formula. Hence in stochastic environments the UCB1-Tuned is more likely to have a better overall performance.

B. Simulation policy

When a simulation is runned from a given node in the tree, the goal is to find a feasible combinaison of airports that could be a solution to our problem. Three simulation policies have been implemented and are used to select the actions needed to reach a leaf node in the tree:

- **Random policy**: This policy selects a random action from the set of available actions, introducing variability and exploration in the simulation process.
- **Greedy policy**: This policy selects the action that corresponds to the cheapest available flight connection, thus prioritising cost minimisation at each step.
- **Tolerance policy** (with coefficient c): This policy selects an action randomly from a subset of actions that are within a certain tolerance level c of the minimum cost action. This policy introduces a more balanced approach than the random and greedy policies.

C. Expansion policy

When expanding a node, it's theoretically possible to expand all available child nodes. However, in practice, this can be computationally expensive and time-consuming, particularly in problems with a large number of possible actions. To address this, heuristic approaches often involve compromises that enhance the efficiency of the search process by selectively expanding certain nodes rather than all possible ones.

- **Top-K policy**: This policy expands the nodes corresponding to the cheapest flight connections available. It sorts all possible actions based on their associated costs and selects the top k actions with the lowest costs, where k is regulated by the allowed number of children N_c.
- **Ratio policy**: This policy takes a more balanced approach by combining the selection of the best actions with a degree of randomness. First, it calculates the number of top actions to select based on a predefined ratio, $c \in [0, 1]$, which reflects the proportion of Top-K Actions within the allowed N_c. After selecting these best actions, the policy randomly selects $(1 - c) * N_c$ actions from the remaining pool to reach the desired number of children.

A $\mathcal{MCTS}$ function can be defined. The function parameters include $S_p(C_p)$ for the selection policy, $E_p(c)$ for the expansion policy, R_p for the rollout policy, and N_c, defining the maximum number of children expanded per node.

D. *Parralelisation*

In computer science, parallelisation is a technique that divides a number of tasks into sub-tasks that can be both, independently and simultaneously run on mutiple cores of a computer. Due to the nature of the MCTS and its four phases, this algorithm is a good candidate for parallelisation.

For instance, after selecting a node to explore, rather than conducting a single simulation based on the one simulation policy, you can either run simulations using multiple different simulation policies and select the best outcome, or perform multiple simulations using the same policy (if it is stochastic), it is called a leaf parralelisation [5].

IV. Computational Results

The results in this section were generated on an i7-10700 CPU Intel Processor with 8 cores at 3.30 GHz with 16.00 GB RAM. We used Python 3.10 on VS Code 1.92.2.

Simulations for every considered instances have been conducted, testing different combinations of parameters in the grid search defined in Table I. One challenge, is the computational budget when using Python. Hence, the size of the grid search for the more complex studied instances is reduced as shown in Table I.

TABLE I
Grid search

	$(I_1 \dots I_6)$	(I_7, I_8)
selection_policy	top_k, ratio_k	top_k, ratio_k
simulation_policy	random, greedy, tolerance	greedy
selection_policy	UCB, UCB1T	UCB
C_p	0, 1.4, 2.8	1.4
N_c	5, 10, 15	10
Ratio c	0, .3, .5, .8, 1	.5
N° simulations	10	10

A. *Overview*

After running the various simulations with the grid search parameters defined in Table I, our results were compared with the best known solutions in Table II [2].

TABLE II
Best results vs State of the art

Instance	Best known	Best found	Gap (%)	Mean	Std
I_1	1396	**1396**	0	1396	0
I_2	1498	**1498**	0	1498	0
I_3	7672	**7672**	0	7672	0
I_4	13952	15361	10.1	15361	0
I_5	690	-	-	-	-
I_6	2159	-	-	-	-
I_7	30937	31924	3.19	30937	0
I_8	4052	**4037**	-0.52	4052	0

B. *Deep dive on I_1,I_2,I_3 and I_4*

For instances I_1, I_2 and I_3, solutions were found and the various simulations were carried out successfully. Therefore, the influence of the parameters on the $\mathcal{MCTS}$ function and the final solution was investigated. However, only few parametrisation of the $\mathcal{MCTS}$ allowed finding a solution for I_4: the UCB1T selection policy and tolerance or random simulation policy created a tree too large to find solutions in a reasonable time.

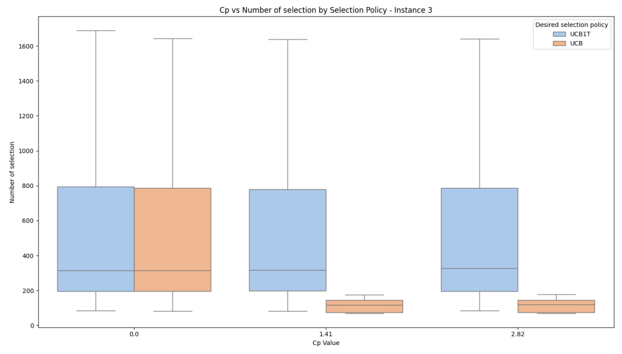

Fig. 1. C_p vs Number of selection

Analysis on C_p: In Figure 1, the box plots illustrate the relationship between the exploration constant C_p and the number of selection phases under the UCB and UCB1T selection policies:

- $C_p = 0$ **lead to the same performance:** When the $C_p = 0$, the selection policy of the UCB and the UCB1T are equal, leading to the same decision-making during the MCTS.
- **Higher C_p values lead to faster convergence for UCB:** As C_p increases from 0.0 to 2.82, the median number of selection phases under the UCB policy decreases.
- **UCB1T encourages more exploration:** UCB1T consistently results in a higher number of selection phases compared to UCB, especially at higher C_p values. This is consistent with UCB1T's definition to promote broader exploration before converging.

Although a higher exploration parameter C_p may lead to faster convergence under the UCB selection policy, it often results in worse outcomes compared to the UCB1T algorithm.

While UCB1T may require more time to converge, it generally explores the search tree more effectively, leading to better overall performance.

Analysis of expansion ratio c: When comparing the relationship between ratio expansion (the proportion of expanded child nodes that has the cheapest flight connection over the chosen number of children) and the time to find a solution for the UCB and UCB1T policies we draw few conclusions:

- **UCB finds solution faster than UCB1T:** Across all ratio expansion values, the UCB policy consistently finds solutions quicker than UCB1T. This suggests that UCB, being less aggressive in exploration, converges on solutions faster.
- **Higher ratios lead to a faster convergence:** For both policies, the time to find a solution generally decreases as the ratio expansion increases, indicating a more efficient search process when expanded nodes are less chosen randomly from the set of available actions. However, in more complex instances, it is crucial to have a ratio $r \in [0.3, 0.7]$ to escape potential leaf node.

Finally, the UCB policy is more correlated to the expansion ratio than the UCB1T as shown in Figure 2. UCB's overall performance is worst than UCB1T because it relies heavily on the exploitation compared to UCB1T that even if it converges slower gives better results.

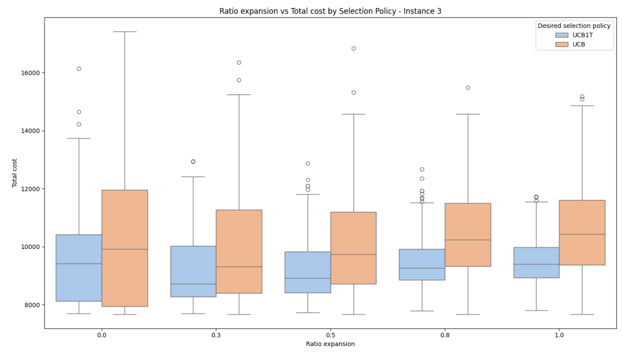

Fig. 2. Expansion ratio vs Total cost

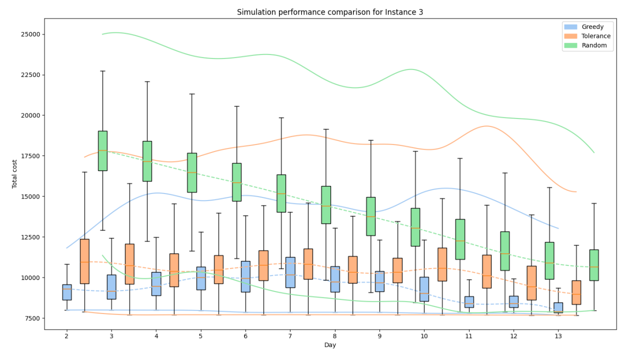

Fig. 3. Simulation performance - Instance 3

Analysis of simulations performances: Box plots for the tree simulations policies are represented on Figure 3. For each day, the distribution of the simulated outcome is plotted regarding the simulation policy. Colored curves represent the minimum and maximum of these distributions, while dashed lines indicate the medians.

In Figure 3, the greedy simulation policy is more performant because the distribution of simulations at every day has a lower min, max and median. The convergence of the random policy is more pronounced due to the policy's inherent randomness. In addition, with the greedy and tolerance policies, at day two or three, the minimum has already almost been reached during simulations. Therefore, a well-calibrated set of parameters for the $\mathcal{MCTS}$ should converge towards the minimum cost found during the simulations. If this is not the case, it indicates that the parameterisation of $\mathcal{MCTS}$ is not optimal. In Figure 5, the distributions of the simulated outcomes are represented for a misparameterised MCTS (I_4).

In Figure 4, the median distributions for the different scenarios have been plotted. One can observe that having a value c too close to 1, does not on average converge to this minimum-cost solution. A contrario, lower c values appears to guide the tree search more effectively during the first days of simulations, which is crucial to not overexpand the size of the tree, which can lead to an inefficient and time-consuming MCTS.

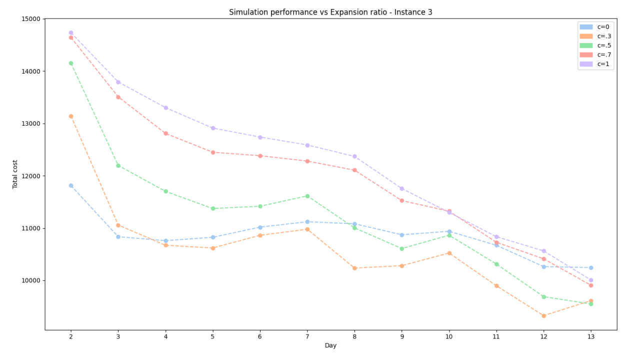

Fig. 4. Simulation performance vs Expansion Ratio - Instance 3

These conclusions can be drawn for small instances, however for I_4, we can clearly see in Figure 5 that having $c = 0$ for a greedy selection policy is inefficient in this tree search because it diverges from the min-simulated cost. The tree search is therefore unable to find a solution after 10 minutes. Based on the median comparison, $c = 1$ is a more optimal parameter for guiding the tree search (for I_3).

Parralelisation: In our implementation, for I_4, we parralelised a MCTS on five cores. The set of parameters has been chosen to demonstrate the behavior of parallelisation in a stochastic environment. The parralelisation has been implemented during the simulation phase of the MCTS. Therefore, we chose the minimum outcome of the five simulations.

In Figure 6, the five cores parallelised's distribution better performs than the non-parralelised approach. It confirms that parallelisation guides the MCTS more effectively in the first days of the tree search.

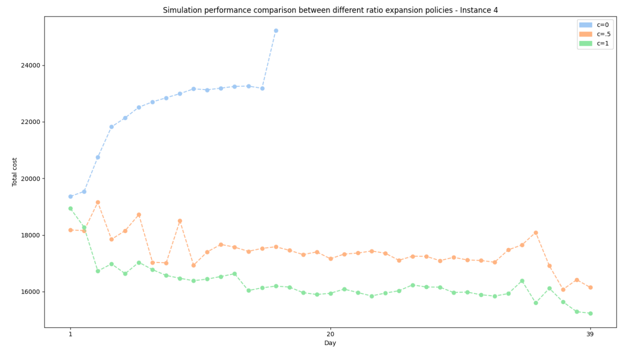

Fig. 5. Simulation performance vs Expansion Ratio - Instance 4

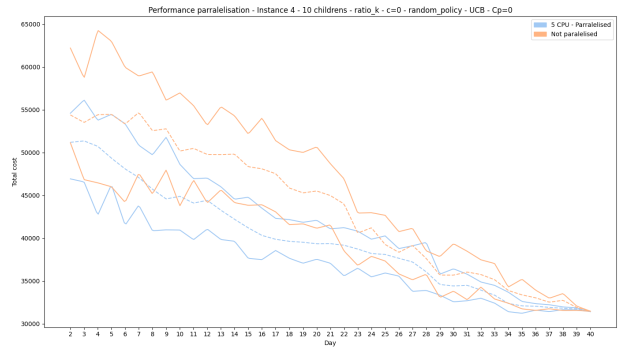

Fig. 6. Comparison of the distributions for the simulated outcomes without parralelisation and with 5 cores - Instance 4

Comparative analysis of five-core and ten-core parallelisations of MCTS, evaluated using Mann-Whitney and Kolmogorov-Smirnov tests in Figure 7, revealed no statistically significant improvements at a 5% level. As discussed in [5], too many modifications to the MCTS can lead to undesirable behaviour.

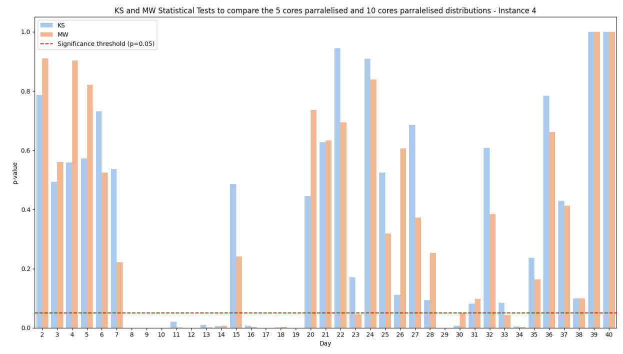

Fig. 7. Statistical tests to compare the 5 and 10 cores distribution - Instance 4

Analysis I_5, I_6, I_7 and I_8

The MCTS function struggled to effectively search the tree with grid search for I_5 and I_6, as nodes simulated under random or tolerance policies that reached final states did not facilitate further tree expansion. I_7 and I_8 were solved using a UCB selection policy with a C_p of 1.41 and a $N_c = 5$ under the top_k expansion policy. For I_7, a solution higher than 3.19% to the state of the art has been found. Finally for I_8, a new state-of-the-art solution was found, improving on the best known solution by 0.52%.

V. Conclusion

In this paper, a Monte Carlo Tree Search solution was implemented to solve the Kiwi.com Traveling Salesman Problem 2.0, focusing on the first eight instances without imposing time constraints. In certain situations, the MCTS finds solutions close to the state of the art solution, or reaches the state of the art solution. For one instance, I_8, a new best solution has been found.

Regarding the selection policy, the UCB1-Tuned outperformed the classic UCB, guiding the tree search more accurately by taking into account the variability of the simulations. However, this selection policy explores the tree search more broadly and takes more time to converge compare to UCB. Regarding the expansion ratio, for small instances I_1, I_2, I_3 a lower ratio was preferred to find solutions faster. Nonetheless, for other instances, a balanced ratio of 0.5 was effective in allowing new potential candidates within the solution space, speeding up the tree search. Altough, it never found comparable solution to the top-k policy, which found solutions for I_7 and I_8 that are close to and better than the best known solutions, respectively. Regarding simulation policies, the greedy approach is the best function accross the different instances with a low risk of the search getting stuck in local optima thanks to the effectiveness of the selection policies. While the tolerance policy provides a more balanced approach, it can have undesirable behaviour for more complex instances (e.g I_4). The random policy has performed well in resolving small instances, but has generally not produced the best results, making it less favourable overall. Finally, we recommend developing parallelisation within the MCTS, which is particularly beneficial when employing stochastic simulations, to better estimate node values and guide the tree search more effectively.

References

[1] Statista, "Number of flights performed by the global airline industry from 2004 to 2023, with a forecasts for 2024," https://www.statista.com/statistics/564769/airline-industry-number-of-flights/, 2024.

[2] Y. Pylyavskyy, A. Kheiri, and L. Ahmed, "A reinforcement learning hyper-heuristic for the optimisation of flight connections," 07 2020, pp. 1–8.

[3] Kiwi.com, "Travelling salesman challenge 2.0: Award ceremony," Youtube, 2019. [Online]. Available: https://www.youtube.com/watch?v=Fp7LaEUwCjE

[4] M. Alrasheed, W. Mohammed, Y. Pylyavskyy, and A. Kheiri, "Local search heuristic for the optimisation of flight connections," 09 2019, pp. 1–4.

[5] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.