Lancaster University

A Monte Carlo Tree Search algorithm to solve Kiwi's Traveling Salesman Challenge 2.0

Arnaud Da Silva

36471977

Supervisor

Ahmed Kheiri

A thesis submitted in fulfillment of the requirements for the degree of Master of Science Business Analytics

in the

Lancaster University Management School
Department of Management Science

September 2024

Declaration of Authorship

I, **Arnaud Da Silva**, hereby declare that this thesis entitled, **“Title”**, is all my own work, except as indicated in the text.

The report has been not accepted for any degree and it is not being submitted currently in candidature for any degree or other reward.

Signed: ______________________________

Date: ______________________________

Abstract

(TODO) Give a short (1 page) overview of the work. This should summarise (not advertise) your research project. After reading the abstract the reader should know what problem you are tackling, the techniques you are using, the results you have achieved.

Acknowledgements

(TODO) Thanking anyone who has helped you in any way

Contents

List of Figures

List of Tables

Abbreviations

AK	Ahmed Kheiri
MS	Management Science

Dedicate this to someone here.

Chapter 1

Introduction

(TODO) Set the scenes. Explain why you are doing this work and why the problem being solved is difficult. Most importantly you should clearly explain what the aims and objectives of your work are.

(TODO) Structure of the thesis. Academic publications produced (if any), including any achievements/highlights

1.1 Background

The number of flight connections keep increasing every year [4], 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, skyscanner, Kayak and Kiwi.com.

These agencies have launched different challenges to create and build powerful trip planner algorithms. For instance, as mentionned in [5], 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.

The given problem is a variant of the Traveling Salesman Problem. 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 often used to solve such TSP problem. In this paper, the Kiwi.com challenge is solved using a Monte Carlo Tree Search.

1.2 Research objectives

This dissertation explores:

- Innovative solution: Solve the Kiwi.com Traveling Salesman problem 2.0 with a new (or never used) heuristic method with no focus on the time limit (as we are using Python and we cannot compete against other participants that mostly used faster program like C).
- Try to find better solutions than the state of the art for some instances.
- Some instances are not really representative of potential use case for this algorithm ($I_9 \dots I_{14}$) because they represent unrealistic scenarios.

1.3 Academic publication

- Discuss with Ahmed - implentation in C to speed up the code
- + think of new smart policies to solve the instances in more reasonnable time

1.4 Dissertation structure

The dissertation is structured as follow:

- Section 2 is the litterature review where the Air Travel optimisations problem are introduced, TSP and its variants are redefined and finally the Monte Carlo Tree Search and an example are presented.

- Section 3 is the problem and instances description to highlight the problem complexity in detail.
- Section 4 is the methodology of our algorithm implementation, where we explain the code's structure, explain the general flow of the algorithm.
- Section 5 is the result and performance of our implementation compared to the state of the art solution.

Chapter 2

Literature Review

(TODO) Present a survey of your main approach and an overview of the approaches proposed previously for solving the problem dealt with in this work

(TODO) Identify the practical and research motivation of this work and the literature gaps

(TODO) How convincing is the authors' argument? (Critical response - comparisons with other research, strengths or weaknesses but in relation to your research)

2.1 Optimisation in Air Travel

In this section, we discuss some common challenges faced by airline companies and demonstrate the importance of optimisation in decision-making for the success and competitiveness of airline companies.

2.1.1 Fleet Assignment Problem

The Fleet Assignment Problem (FAP), as discussed in [6] involves assigning different types of aircraft, to flights based on their capabilities, operational costs, and revenue potential. This decision greatly influences airline revenues and is a vital part of the overall scheduling process. The complexity of FAP is driven by the large number of flights an airline manages daily and its interdependencies with other processes like maintenance and crew scheduling.

2.1.2 Crew Scheduling Problem

The Crew Scheduling Problem (CSP), as discussed in [7], involves assigning crews to a sequence of tasks, each with defined start and end times, with the primary objective of ensuring that all tasks are covered while adhering to regulations on maximum working hours for crew members.

This problem is particularly critical for low-cost airlines, for example in the United Kingdom in 2023, low-cost flights comprise 48% of the scheduled capacity (total number of seats offered) [8], which rely heavily on optimised crew schedules to maintain competitiveness. Efficient crew scheduling is essential not only for low cost carriers and for cost minimisation but also for ensuring operational reliability and flexibility in response to unexpected disruptions. [9]

2.1.3 Disruption Management

Disruptions in airline operations, as noted in [10], can occur due to various factors, including crew unavailability, delays from air traffic control, weather conditions, or mechanical failures. Given that flight schedules are typically planned months in advance [11], effective disruption management is crucial to minimise the impact on passengers and overall airline operations.

The two mains drivers of disruption management are aircraft and crew recovery.

- Aircraft recovery: Optimisation tools help manage the complex logistics of matching available aircraft with rescheduled flights, considering factors like airport availability and maintenance requirements.
- Crew recovery: Optimisation tools are used to adjust crew schedules, taking into account factors such as legal working hours, crew availability, and the need to cover all flights efficiently. These tools help in developing feasible and compliant crew rosters that adapt to the new flight schedules.

These optimisation strategies, supported by advanced software, for instance [12] and [13], are crucial for reducing the impact of disruptions and boosting operational resilience in the airline industry.

2.1.4 Airline adaptation to new demand

Airline companies must continuously adapt their schedules to meet evolving market demands, particularly with the growing dominance of leisure travel over business travel, which has introduced new patterns of demand as shown on Figure 2.1 in Europe. This seasonality poses a challenge for airlines as they have to balance high demand during peak seasons with the risk of underutilisation during off-peak times.

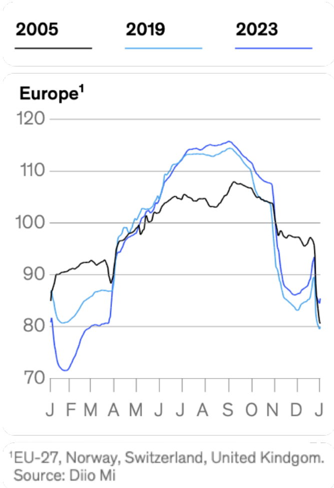


FIGURE 2.1: European demand seasonality [1]

Since travel demand varies throughout the year, airlines use a variety of techniques to achieve operational efficiency while maximising revenue [1]. For instances, airlines sell nearly 65% more seats. To ensure their operatios remain efficient during periods of heightened demand, airline companies make the required allowance for additional aircraft and crew by optimisation models that specify priority routes and requirements for additional flights, alongside effective crew rotation management.
In contrast, winter months pose a different type of problem where demand drops, which can potentially lead to underutilisation of aircrafts. To manage this, airlines are known to turn to ACMI leasing (agreement between two airlines, where the lessor agrees to provide an aircraft, crew, maintenance and insurance [14]) during periods of low demand to temporarily reduce fleet size by outsourcing their capacity. Alongside this, they also increase maintenance activities and incentivise crews to take holidays or undergo training to maximise productivity across the operation. Equally, on a year-round basis, airlines apply dynamic pricing algorithms to vary fares in reaction to real-time demand patterns. In high-demand summer months, fares are tactically set so as to maximise revenues from travelers willing to pay more, while in winter, pricing strategies are aimed at stimulating demand with fare reductions to fill seats that otherwise would have gone empty. Such

adaptive strategies are critical to the airlines for effectively beating the seasonal ebbs and flows in the travel industry.

2.2 Traveling Salesman problem and its adapation

The Traveling Salesman Problem is a well known problem in the Operational Research and Computer Science fields. A simple description of the TSP is to find the best roundtrip for a saleman that has to travel around a given number of cities while minimising the overall journey's distance. This problem is characterised as $\mathcal{NP}$-Hard [15]. This means that there is no known polynomial-time algorithm that can solve all instances of the problem efficiently . Regarding time complexity, if we were to solve it exploring all the possible solutions, the time complexity would have been $\mathcal{O}(\frac{(n-1)!}{2})$ where n represents the number of cities.

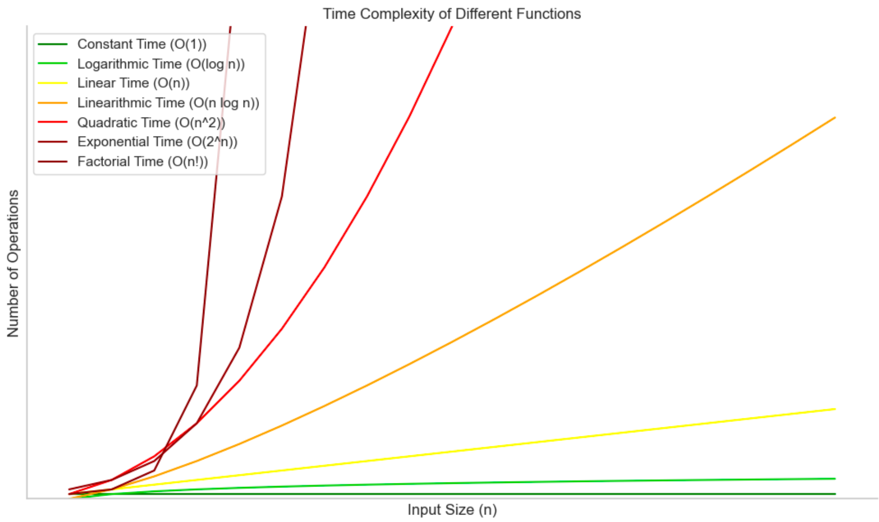


FIGURE 2.2: Time complexity of different functions [2]

On Figure 2.2, different time complexities are compared and demonstrates that the factorial time complexity is the worst. Therefore, these kinds of $\mathcal{NP}$-Hard problem are typically not solved exploiting all the search area but using heuristics algorithms. Heuristics solutions do not guarantee to find the absolute optimal solution but can find near-optimal solutions within more reasonnable timeframes.

The TSP has been studied extensively, and, many variants can be derived from it:

- Symmetric TSP (STSP): The distance between cities are symmetric, meaning that the distance to travel from city A to city B is the same as from city B to city A.
- Assymetric TSP (ATSP): The distance between cities are assymetric, meaning that the distance to travel from city A to city B is different than the distance to travel from city B to city A.[16]

- Multiple TSP (mTSP): Instead of one salesman, multiple salesman are starting from one city, they visit all the cities such that each city is visited exactly once. [17]
- Time Window TSP (TWTSP): Each city has to be visited in a defined time slot. [18]
- Price-collection TSP (PCTSP): Not all the cities have to be visited, the goal is to minimise the overall traveler's distance while maximising the price collected earned when visiting a city. [19]
- Stochastic TSP (STSP): The distances between the cities or the cost of travels are stochastic (ı.e random variables) rather than deterministic. [20]
- Dynamic TSP (DTSP): The problem can change over time, that means that new cities can be added or distances between cities can change while the salesman has already started his journey. [21]
- Generalised TSP (GTSP): The cities are grouped into clusters, the goal is to visit exactly one city from each cluster. [22]
- Open TSP (OTSP): The traveler does not have to end his journey at the starting city. [23]

Multiple algorithms have been developed to address these TSP variants, we can classify them into two categories:

- **Exact Algorithms**: These algorithms aim to find the optimal solution to the TSP by exploring all possible routes or by using mathematical techniques to prune the search space efficiently. Examples include:
  - **Branch and Bound**: This method systematically explores the set of all possible solutions, using bounds to eliminate parts of the search space that cannot contain the optimal solution. It is often used for smaller instances of TSP due to its computational intensity. [24]
  - **Cutting Planes**: This technique adds constraints (or cuts) to the TSP formulation iteratively to remove infeasible solutions and converge to the optimal solution. This approach is particularly effective for symmetric TSPs. [25]

  - **Dynamic Programming**: Introduced by Bellman, this approach breaks down the TSP into subproblems and solves them recursively, which is highly effective for specific TSP variants, though its complexity grows exponentially. [26]

- **Approximation and Heuristic Algorithms**: These algorithms are designed to find near-optimal solutions within a reasonable time frame, specifically for large-scale problems where exact methods are computationally infeasible. Examples include:

  - **Greedy Algorithms**: These algorithms make a series of locally optimal choices in the hope of finding a global optimum. An example is the Nearest Neighbor algorithm, which selects the nearest unvisited city at each step. [27]
  - **Genetic Algorithms**: Inspired by the process of natural selection, these algorithms evolve a population of solutions over time, using operations such as mutation and crossover to explore the solution space. [28]
  - **Simulated Annealing**: This probabilistic technique searches for a global optimum by allowing moves to worse solutions based on a temperature parameter that gradually decreases. It is particularly useful for escaping local optima. [29]
  - **Ant Colony Optimization**: This metaheuristic is inspired by the foraging behavior of ants and uses a combination of deterministic and probabilistic rules to construct solutions, which qre grqdually refined through updates based on pheromone trails. [30]

Some TSP problems (or its variants) have been solved using other algorithms.

2.3 The Monte Carlo Tree Search algorithm

The Monte Carlo Tree Search (MCTS) algorithm can be characterised as less traditionnal than the previously enounced methods in Section 2.2 because MCTS is typically used in games. MCTS' (and its variants) have been successfully implemented across a range of games, such as Havannah [31], Amazons [32], Lines of Actions [33], Go, Chess, and Shogi [34], establishing it as the state-of-the-art algorithm [35], [36], [37]. It is widely used in board games and is increasingly popular since Google DeepMind developed AlphaGo. AlphaGo is a software that was created to beat the best Go's player in the world.

Go is a board game from China where two players take turns placing black or white stones on a grid. The goal is to capture territory by surrounding empty spaces or the opponent's stones. Despite its simple rules, Go is a complex game, with countless possible moves and strategies. It is known for its balance between intuition and logic, hence why it has been a significant focus of artificial intelligence research ,[38]. In 2016, Lee Sedol [39] - the best Go's player in the world was been beaten by AlphaGo 4-1 [40].

MCTS with policy and value networks are at the heart of AlphaGo decision-making process, enabling AlphaGo's to pick the optimal moves in the complex search of Go. [41]

2.3.1 Overview

The MCTS' process is conceptually straightforward. A tree is built in an incremental and assymatric manner (Figure 2.3). For every iteration, a selection policy is used to determine which node to select in the tree to perform simulations. The selection policy, typically balances the exploration (looking into parts of the tree that have not been visited yet) and the exploitation (looking into parts of the trees that appear to be promising). Once the node is selected, a simulation - a sequence of available actions, based on a simulation policy, is applied from this node until a terminal condition is reached e.g no further actions are possible. [42]

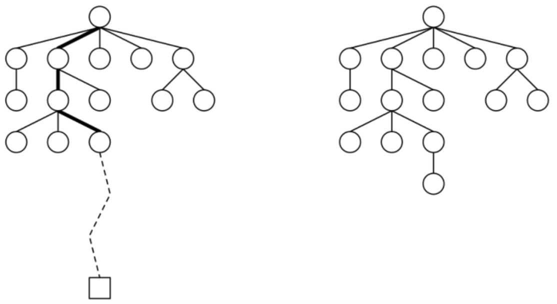


FIGURE 2.3: Assymetrical growth of MCTS - Simulation and Expansion - [3]

To ensure a clearer understanding of MCTS algorithm's stages, we will start by exploring a detailed example [43]. This example will illustrate each component of the algorithm in action. Furthermore, we will generalise the principles discussed, as the methodology of this paper is built on the application of the MCTS algorithm.

2.3.2 Example

Let's say we are given a maximisation problem. When beginning the game, you have two possible actions a_1 and a_2 from the node $S_0^{0,0}$ in the tree $\mathcal{T}$. Every node is defined like so: $S_i^{n_i,t_i}$ where n_i represents the number of times node i has been visited, t_i the total score of this node. Moreover, for every node - we can compute a selection metric, for instance the UCB value: $UCB(S_i^{n_i,t_i}) = \bar{V}_i + 2\sqrt{\frac{\ln N}{n_i}}$ where $\bar{V}_i = \frac{n_i}{t_i}$ represents the average value of the node, n_i the number of times node i has been visited, $N = n_0$ the number of times the root node has been visited (which is also equal to the number of iterations).

Before the first iteration, none node have been visited - $\forall i \in \mathcal{T}, S_i^{0,0}$. At the beginning

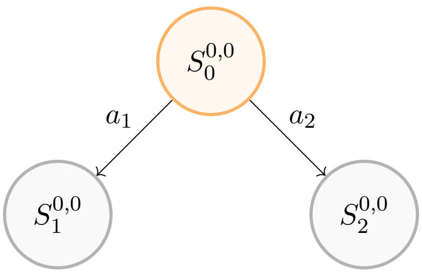


FIGURE 2.4: Selection - $I1$

of $I1$, we have to choose between these two child nodes (or choose between taking a_1 or a_2). After, we have to calculate the UCB value for these two nodes and pick the node that maximises the UCB value (as we are dealing with a maximisation problem).

In Figure 2.4, neither of these have been visited yet so $UCB(S_1^{0,0}) = UCB(S_2^{0,0}) = \infty$. Hence we decide to choose randomly $S_1^{0,0}$.

$S_1^{0,0}$ is a leaf node that has not been visited - then we can simulate from this node, which means selecting actions from this node based on the simulation policy to a terminal state as shown on Figure 2.5:

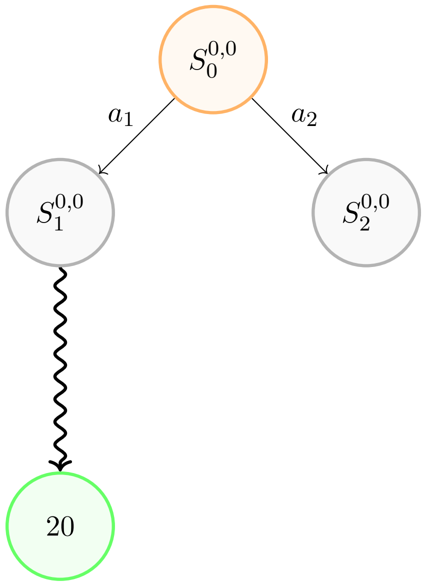


Figure 2.5: Simulation - $I1$

The terminal state has a value of 20, we can write that the rollout/simulation from node $S_1^{0,0}$ node is $\mathcal{R}(S_1^{0,0}) = 20$. The final step of $I1$ is backpropagation. Every node that has been visited in the iteration is updated. Let $\mathcal{N}_{\mathcal{R},j}$ be the indexes of the nodes visited during the $j-th$ iteration of the MCTS:

- Before backpropagation:

$$\forall i \in \mathcal{N}_{\mathcal{R},j}, S_{i,old}^{n_i,t_i} \tag{2.1}$$

- After backpropagation:

$$\forall i \in \mathcal{N}_{\mathcal{R},j}, S_{i,new}^{n_i+1,t_i+\mathcal{R}(S_{i,old}^{n_i,t_i})} \tag{2.2}$$

We can then define a backpropagation function:

$$\begin{array}{rccc} \mathcal{B} & : & \mathcal{N}_{\mathcal{R},j} & \rightarrow & \mathcal{N}_{\mathcal{R},j} \\ & & S_i^{n_i,t_i} & \mapsto & S_i^{n_i+1,t_i+\mathcal{R}(S_i^{n_i,t_i})} \end{array}$$

Then, back to the example on Figure 2.6 we update the nodes $\mathcal{B}(S_1^{0,0}) = S_1^{\mathbf{1,20}}$ and $\mathcal{B}(S_0^{0,0}) = S_0^{\mathbf{1,20}}$.

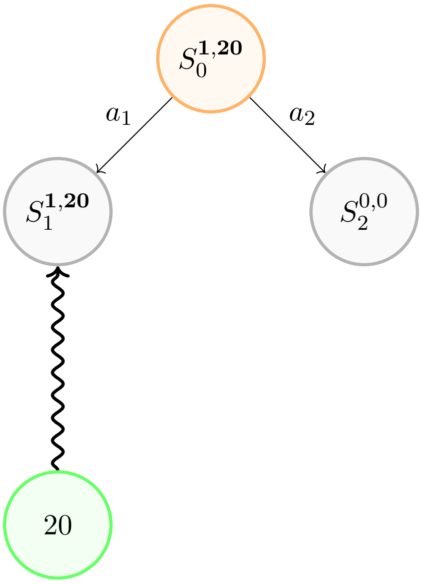
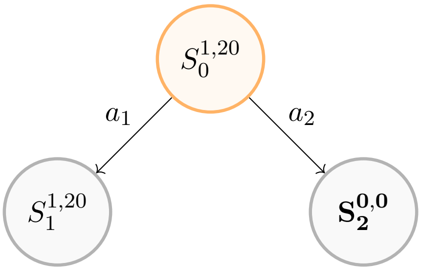


FIGURE 2.6: Backpropagation - I1

The fourth phase of the algorithm has been done for $I1$. Therefore, we can then start the 2^{nd} iteration $I2$. On Figure 2.7, we can either choose a_1 or a_2. When a child node



$$UCB(S_1^{1,20}) = 20 + 2\sqrt{\frac{\ln 1}{1}} = 20$$
$$UCB(S_2^{0,0}) = \infty$$

FIGURE 2.7: Selection - I2

has not been visited yet, you pick this node for the Selection or you can compute the UCB value, it leads to the same conclusion.

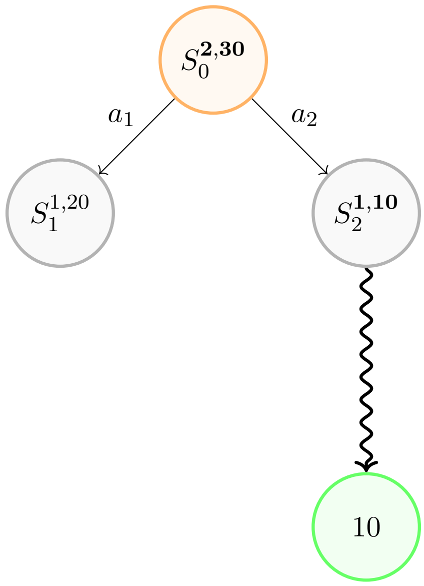


FIGURE 2.8: Simulation and Backpropagation - I2

We can simulate (Figure 2.8) from the chosen node $S_2^{0,0}$ and $\mathcal{R}(S_2^{0,0}) = 10$ and back-propagate all the visited nodes: $\mathcal{B}(S_2^{0,0}) = S_2^{1,10}$ and $\mathcal{B}(S_0^{1,20}) = S_0^{2,30}$. Next, we start the 3^{rd} iteration, based on the UCB score we decide to choose a_1.

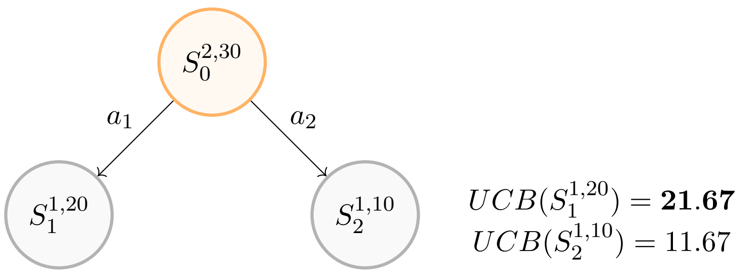


FIGURE 2.9: Selection - I3

$S_1^{1,20}$ is a leaf node and has been visited so we can expand this node.

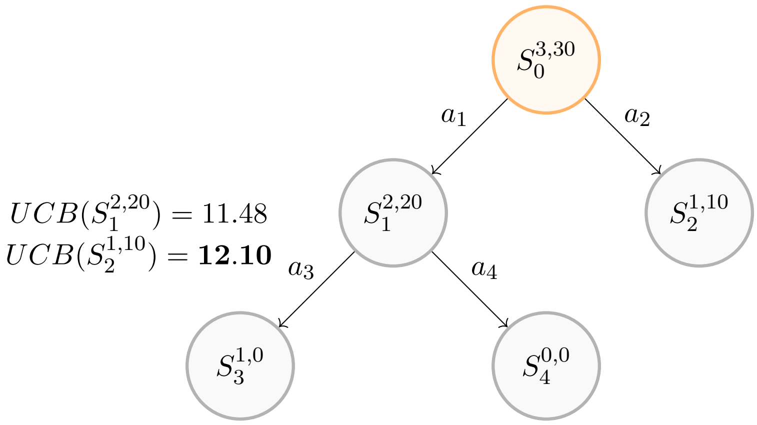


FIGURE 2.10: Selection and Expansion - I3

Based on UCB score we decide to simulate from $S_3^{0,0}$ on Figure 2.11

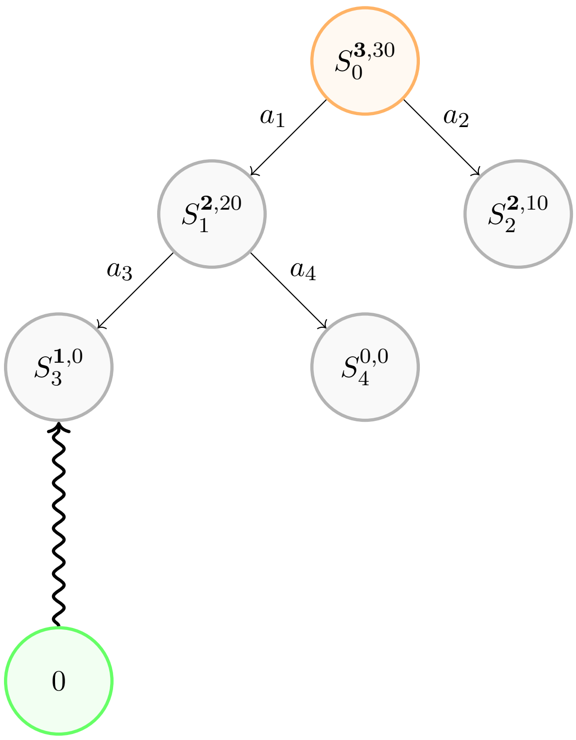


FIGURE 2.11: Simulation and Backpropagation - I3

This is the fourth iteration $I4$ represented on Figure 2.12:

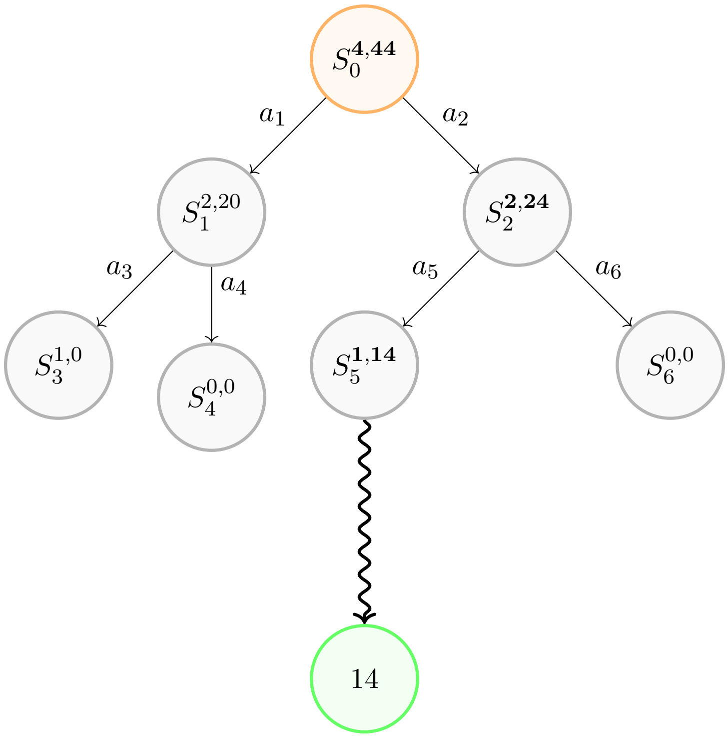


FIGURE 2.12: Selection - Simulation - Backpropagation - I4

The MCTS algorithm can either be stopped because you are running out of time or because you have no more available actions. For instance, if we were to stop at this stage of the algorithm, the best action to undertake is a_2 because it has the higher average value: $\bar{V_1} = \frac{20}{2} \leq \bar{V_2} = \frac{24}{2}$.

2.3.3 The different parameters in the MCTS

As outlined in the previous example, node's selection is crucial in the MCTS process and can significantly influence the performance of the algorithm. The selection function traditionnaly used is the Upper Confidence Bound 1 (UCB). However, there are a lot of different MCTS' selection functions as mentionned in this survey [44]. Every selection function, is based on the upper confidence bound principle, which balances the dual aspect of exploration and exploitation in the tree search.

The UCB and is variants, the UCB1-Tuned are defined as follow:

$$UCB = \overline{X}_i + C_p\sqrt{\frac{2\ln N}{n_i}} \tag{2.3}$$

$$UCB\text{-Tuned} = \overline{X}_i + \sqrt{\frac{\ln N}{n_i}\min\left(\frac{1}{4}, \mathrm{Var}(X_i) + \sqrt{\frac{2\ln N}{n_i}}\right)} \tag{2.4}$$

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.

Other selection policies, such as the Beta policy or Single Player MCTS, also play significant roles in various applications of the Monte Carlo Tree Search. However, these policies will not be the focus of this study due to their probabilistic nature, which does not align well with our specific problem context.

Chapter 3

Problem Description

3.1 Overview

Kiwi's traveler wants to travel in 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 $d_1 \neq d_2$. The problem can hence be characterised as an generalised, assymetric and time dependant TSP - as discussed in Section 2.2.

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, and we found it particularly valuable in our study to rigorously formulate the problem mathematically, as it provided a clear framework to analyse and understand its complexities.

We can then formulate the problem as follow:

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

Parameters

- c_{ij}^d: Cost to travel from airport i to airport j on day $d \in \mathcal{D}$.

Variables

- 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.

Constraints

1. Starting and Ending Constraints:

- The traveler starts at the known starting airport S_0.
- The traveler must return to an airport in the starting area on the final day N.

2. Flow Constraints:

- The traveler must leave each area and arrive at the next area on consecutive days, the next area has not been visited yet.
- Ensure that the traveler can only fly into and out of the same airport within an area.

- Ensure each area is visited exactly once.
- Update the unvisited list as areas are visited.

Objective Function

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

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

Constraints

- Starting at the known starting airport S_0 at take an existing flight connection:

$$\sum_{j \in A_1} x_{S_0,j}^1 = 1$$

$$\forall d \in \mathcal{D}, c_{S_0,j}^d \in \mathbb{R}^{+*}$$

- Visit exactly one airport in each area each day:

$$\sum_{i \in A_d} \sum_{j \in A_{d+1}} x_{ij}^d = 1 \quad \forall d \in \{1, 2, \dots, N-1\}$$

- Ensure the traveler leaves from the same airport they arrived at the previous day:

$$\sum_{k \in A_d} x_{ik}^d = \sum_{k \in A_{d-1}} x_{ki}^{d-1} \quad \forall i \in \bigcup_{j=1}^{N} A_j, \forall d \in \{2, 3, \dots, N\}$$

- Return to an airport in the starting area on the final day with an existing flight connection:

$$\sum_{i \in A_N} \sum_{j \in A_1} x_{ij}^N = 1$$

$$\forall (i,j) \in A_N \times A_1, c_{i,j}^N \in \mathbb{R}^{+*}$$

- 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:

$$x_{ij}^d \leq c_{ij}^d \cdot v_j^d \quad \forall i, j \in (\bigcup_{j=1}^{N} A_j)^2, \forall d \in \mathcal{D}$$

$$x_{ij}^d \leq v_j^d \quad \forall j \in \bigcup_{j=1}^{N} A_j, \forall d \in \mathcal{D}$$

- Binary variable constraints:

$$x_{ij}^d \in \{0, 1\} \quad \forall (i, j) \in (\bigcup_{j=1}^{N} A_j)^2, \forall d \in \mathcal{D}$$

$$v_j^d \in \{0, 1\} \quad \forall j \in \mathcal{A}, \forall d \in \mathcal{D}$$

3.2 Instances

3.2.1 Description

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

For example, the first few lines of I_4 are:

13 GDN
first
WRO DL1
second
BZG KJ1
third
BXP LB1

That means that the Traveller will visit 13 different areas, he starts from airport GDN, that belongs to the starting area. Then we are given the list of airports that are in every zone. For example, the second zone is named second and has two airports: WRO and DL1.

After all the information regarding the areas and the airports we have the flight connections informations. In Table 3.1, few flights are displayed from I_6 for illustrative purposes.

Table 3.1: Flight connections sample I6

Departure from	Arrival	Day	Cost
KKE	BIL	1	19
UAX	NKE	73	16
UXA	BCT	0	141
UXA	DBD	0	112
UXA	DBD	0	128
UXA	DBD	0	110

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, we could have the same flight connections at a specific day but with different prices. Furthermore, we have to consider solely the more relevant connections i.e. the flight connection with the lowest fare, on 3.1 we only consider the flight from UXA to DDB with the associated cost of 110.

3.2.2 General formulation

We decided to formulate the problem mathematically because it was not done in the existing papers, and we found it useful to clearly understand the problem's instances and their characteristics.

An instance I_i can be mathematically defined as follows:

$$I_i = (N_i, S_{i0}, A_i, F_i)$$

where:

- **Number of Areas** N_i:
$$N_i \in \mathbb{N}$$
The total number of distinct areas in instance I_i.

- **Starting Airport** S_{i0}:
$$S_{i0} \in \text{Airports}$$
The starting airport of the traveller.

- **Airports in Each Area**:
$$A_i = \{A_{i,1}, A_{i,2}, \dots A_{i,N_i}\}$$
where each $A_{i,j}$ is a set of airports in area j for instance i:
$$A_{i,j} = \{a_{i,j,1}, a_{i,j,2}, \dots, a_{i,j,k_j}\}$$
with a_{i,j,k_j} being airports in area j and k_j is the number of airports in area j.

- **Flight Connections**:
$$F_i = \{F_{i,0}, F_{i,1}, F_{i,2}, \dots, F_{i,N_i}\}$$

where each flight matrix $F_{i,k}$ represents the flight information of instance i on day k:

$$F_{i,k} = \begin{pmatrix} a^d_{i,k,1} & a^a_{i,k,1} & f_{i,k,1} \\ a^d_{i,k,2} & a^a_{i,k,2} & f_{i,k,2} \\ \vdots & \vdots & \vdots \\ a^d_{i,k,l_{k,i}} & a^a_{i,k,l_{k,i}} & f_{i,k,l_{k,i}} \end{pmatrix}$$

- **Columns**:
  * Departure Airport: $a^d_{i,k,j}$ (Departure airport for the j-th flight on day k)
  * Arrival Airport: $a^a_{i,k,j}$ (Arrival airport for the j-th flight on day k)
  * Cost: $f_{i,k,j}$ (Cost of the j-th flight on day k), where $j \in [1, l_{k,i}]$
- **Rows**: Each row corresponds to a specific flight on day k. The number of rows $l_{k,i}$ depends on the number of flights available on that day.

3.2.3 Kiwi's rules

When solving all the instances, Kiwi's defined time limits constraints based on the nature of the instance. We can summarise these constraints in the Table above:

TABLE 3.2: Time limits based on the number of areas and airports

Instance	nb areas	Nb Airports	Time limit (s)
Small	≤ 20	< 50	3
Medium	≤ 100	< 200	5
Large	> 100		15

All the useful information about the instances such as the starting airport, the associated area, the range of airports per area, the number of airports and the time limit constraints are defined in Table 3.3.

TABLE 3.3: Instances and their respective parameters

Instances	Starting Area - Airport	N° areas	Min - Max airport per area	N° Airports	Time Limit (s)
I1	Zona_0 - AB0	10	1 - 1	10	3
I2	Area_0 - EBJ	10	1 - 2	15	3
I3	ninth - GDN	13	1 - 6	38	3
I4	Poland - GDN	40	1 - 5	99	5
I5	zone0 - RCF	46	3 - 3	138	5
I6	zone0 - VHK	96	2 - 2	192	5
I7	abfuidmorz - AHG	150	1 - 6	300	15
I8	atrdruwkbz - AEW	200	1 - 4	300	15
I9	fcjsqtmccq - GVT	250	1 - 1	250	15
I10	eqlfrvhlwu - ECB	300	1 - 1	300	15
I11	pbggaefrjv - LIJ	150	1 - 4	200	15
I12	unnwaxhnoq - PJE	200	1 - 4	250	15
I13	hpvkogdfpf - GKU	250	1 - 3	275	15
I14	jjewssxvsc - IXG	300	1 - 1	300	15

Chapter 4

Methodology

(TODO) Describe implementation details

4.1 Monte Carlo Tree Search implementation

4.1.1 General flow

Based on the discussion in Chapter 2, the flow of the Monte Carlo Tree Search algorithm is summarised in Figure 4.1:

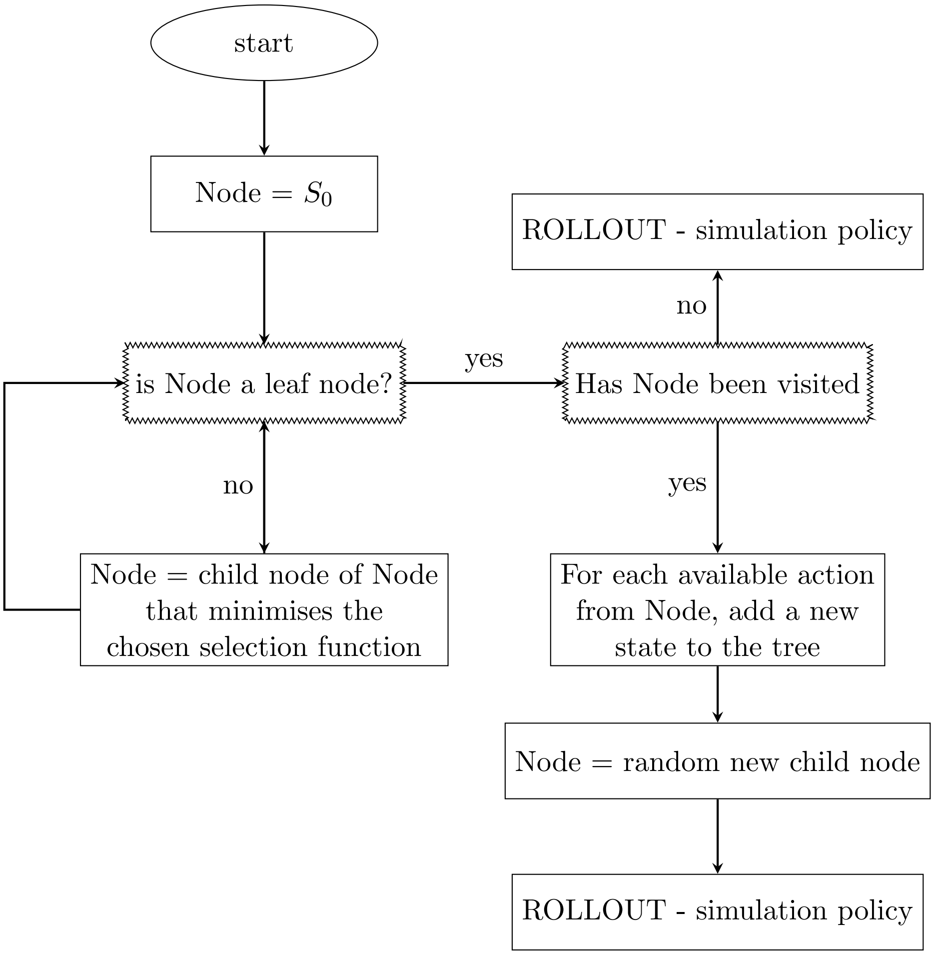


FIGURE 4.1: Flow MCTS

For every iteration of this algorithm, there are four different phases:

1. **Selection:** Starting from the root node (the starting airport S_{i0} for I_i), select successive child nodes (airports that are in unvisited areas) until a leaf node (the airport in the initial area, not necessarly the starting airport) is reached. Use the chosen Selection function to evaluate which node is the most promising. In the illustrative example in Section 2.3.2, the UCB1 function was used for the

selection function. Furthermore, the problem's goal was to maximise the objective function, hence the nodes with the highest UCB1 value was selected. A contrario, in Kiwi's minimisation problem, nodes are evaluated based on the lowest value of the selection function.

2. **Expansion:** If the selected node is not a terminal node, expand the tree by adding all possible child nodes.

3. **Simulation:** From the newly added node, perform a simulation (based on the simulation policy) until a feasible terminal node is reached.

4. **Backpropagation:** Update the values of the nodes along the path from the newly added node to the root based on the result of the simulation.

$$\mathcal{B}(S_i^{n_i,t_i}) = S_i^{n_i+1,t_i+\mathcal{R}(S_i^{n_i,t_i})} \tag{4.1}$$

where $\mathcal{R}(S_i^{n_i,t_i})$ is the cost of the solution found after performing a simulation from node $S_i^{n_i,t_i}$.

4.1.1.1 Data Preprocessing

To implement our MCTS' solution, the first thing to create is a data_preprocessing class to prepare the given instance to the problem at hand. Kiwi's challenge is solved using Python 3.10 on VS Code 1.92.2. Our Python code is structured using object-oriented programming following CamelCase's convention [45]. This data_preprocessing class is represented on Figure 4.2. The input is an instance I_i, as defined in Chapter 3:

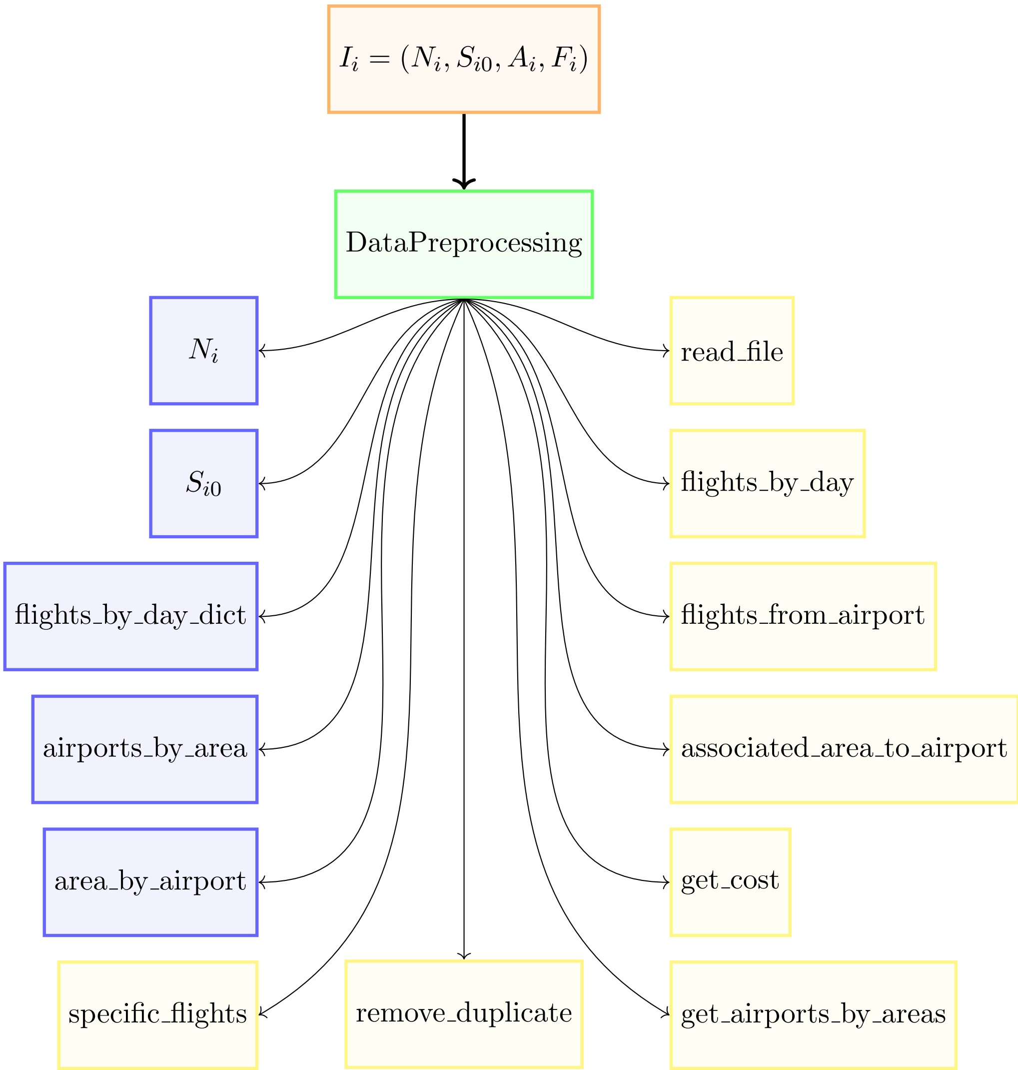


FIGURE 4.2: Explanation of the data preprocessing class

Different useful methods are implemented within the data_preprocessing class to compute and manage various attributes required for the problem at hand. These methods are designed to prepare and structure the data, making it easier to use in subsequent phases of the algorithm. For example, the remove_duplicate method ensures that only the cheapest flight connections are considered between two airports if multiple flight connections exist at different prices, on the same day. Other methods, such as flights_by_day_dict and get_airports_by_areas organise the data. The first method regroups all the flights by their respective days, creating a dictionary where each key represents a day and its corresponding value is a list of available flights. The second method regroups all the airports present in the different areas.

Finally, other methods, such as specific_flights, will be useful for developing the MCTS' algorithm. These give all the possible flight connections from a specific airport on a given day, taking into account the areas visited, so that all possible actions can be obtained from a node.

Given that Python is relatively slower, in terms of computation, compared to other programming languages, dictionnaries are used where possible. Dictionnaries allow for efficient data retrieval based on a key, with an average time complexity of $\mathcal{O}(1)$. This choice improves the performance of the data preprocessing step, enabling the algorithm to run more efficiently despite Python's inherent limitations.

4.1.1.2 Node

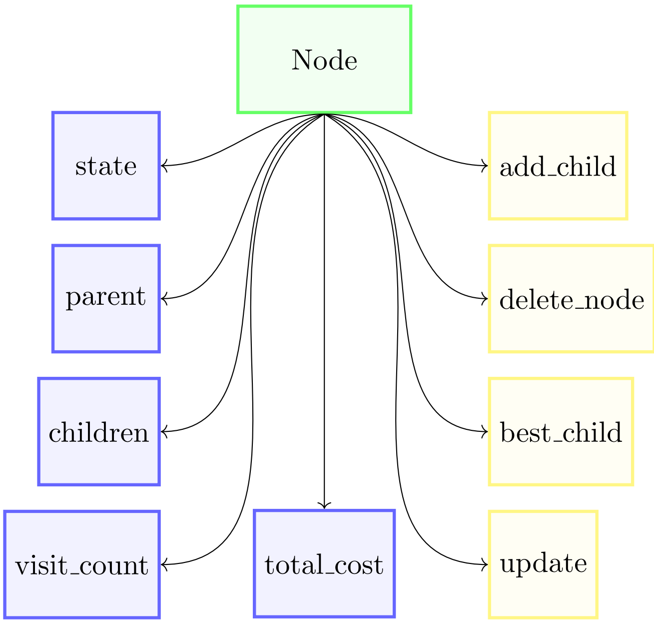


FIGURE 4.3: Explanation of the Node class

As mentionned earlier in Section 2.3.2,a Node structure is used in the algorithm, hence the implementation of a Node class. Each Node has a reference to a parent node (unless it is the root node) and may have one or more child nodes (unless it is a leaf node). These relationships form a tree structure where each node can expand into potential future states, guiding the search process. The visit_count tracks the number of times a node has been visited during the MCTS process. This is crucial for evaluating the node's importance and for calculating the score of the node with the selection function. The state is a dictionnary that contains the node's current information:

- current_airport: The airport where the traveler is at this node.
- current_day: The day of the trip at this node.
- remaining_zones: The zones that still need to be visited to complete the journey.
- visited_zones: The zones that have already been visited to ensure that all zones are visited exactly once during the trip.
- total_cost: It represents the accumulated cost of the current solution path leading to this node.

Additionally, to manage the expansion of child nodes, the add_child method is defined. This method generates new nodes based on the possible actions available from the current node. These new nodes represent the next possible states in the traveler's journey, allowing the search tree to expand and explore different travel routes. Finally, the delete_node method can be used to delete a node from the list of its parent's children.

4.2 The different policies

In the previous section, we outlined the general flow of the MCTS algorithm, focusing on two cores classes, DataPreprocessing and Node, that are central in MCTS' implementation.

In Section 2.3.3, we explored the various selection policies that guide the decision-making process within the MCTS Although there is a limited litterature review, we decided to parameterise not only the selection policy but also the simulation and expansion policy.

4.2.1 Simulation policies

When you simulate 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. From this node chosen for simulation, we obtain the current state (defined in section 4.1.1.2). The remaining actions must then be chosen to find a simulated solution based on the simulation policy.

Below is the definition of the three distinct simulation policies:

- 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 of the minimum cost action. The tolerance level is defined by a coefficient c, allowing for a balance between exploration and exploitation.

  The tolerance policy is defined as follows:

   - Identify the cheapest flight connection among the available actions c_{min}.
   - Filter the actions to include only those with a cost less or equal than $c_{min}(1+c)$.
   - Randomly select an action from this filtered set.

4.2.2 Expansion policies

When expanding a node, it's theoretically possible to expand all available child nodes i.e. add to the tree all the possible flight connections from this airport (that are in the available actions based on the visited areas). 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.

Firstly, we defined `number_of_children`, a parameter of our MCTS algorithm which regulates the maximum number of children that can be expanded from any given node. This limitation controls the size of the search tree, as expanding too many children for every selected node could make the algorithm computationally exhaustive.

In our implementation we defined two expansion policies:

- **Top-K Actions Policy**: This policy expands the nodes corresponding to the cheapest flight connections available. Specifically, 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 `number_of_children`. This approach ensures that only the most promising actions, in terms of cost efficiency, are considered during expansion. This policy narrows down the search space but can increase the chance to reach a leaf node.

- **Ratio Best-Random 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 `number_of_children`. After selecting these best actions, the policy randomly selects $(1 - c) * number\_of\_children$ actions from the remaining pool to reach the desired number of children. This policy is designed to explore a broader range of possibilities while still prioritising cost-effective options.

4.2.3 Pseudo-code

In this section, the implementation of the algorithm in practice is explored by examining the different functions of our MCTS class. The search function of the MCTS is defined:

Algorithm 1 Search_Function

```
Initialise Root_Node with Initial_State
while Tree is not fully explored do
    Node ← Select(Root_Node)
    if Node is not fully expanded then
        Node ← Expand(Node)
    end if
    Cost ← Simulate(Node)
    Backpropagate(Node, Cost)
end while
return Best_Leaf_Node
```

The `Search` function represents the general flow of the algorithm as mentionned on Figure 4.1.

The `Select` function (Algorithm 2), which selects the node to visit, returns two arguments: a boolean and a node. The boolean indicates to the expansion function whether expansion is necessary (True means no expansion needed, False means expansion needed).

Algorithm 2 Select_Function

```
Input: Node
Current ← Node
while Current.Children is not empty do
    if Current is not fully expanded then
        UnvisitedChildren ← Children with VisitCount = 0
        if UnvisitedChildren is not empty then
            SelectedChild ← Randomly select from UnvisitedChildren
            return True, SelectedChild
        end if
    else
        Current ← BestChild(Current)
    end if
end while
if Current.Children is empty and Current.State["current_day"] == N_Areas then
    return False, Current
else if Current.Children is empty and Current.State["current_day"] <> N_Areas then
    return False, Current
else if Current.State["current_day"] == N_Areas + 1 then
    return True, Current
end if
```

There are special cases to handle, when one approaches the final solution because one has to communicate the right information to the `Expand Node` function.

After simulating, the backpropagation function updates the node's attributes. The new node becomes the parent of this node, and so on until `Node` is `None`, i.e., all the information is backpropagated up to the root node.

Algorithm 3 Backpropagate_Function

```
while Node is not None do
    Node.Update(Cost)
    Node ← Node.Parent
end while
```

The transition function modifies the states of a node by updating the current airport, the visited zones, remaining zones, etc.

Algorithm 4 Transition_Function

$New\_State \leftarrow$ Copy of $State$
$New\_State.Current\_Day \leftarrow State.Current\_Day + 1$
$New\_State.Current\_Airport \leftarrow Action[0]$
$New\_State.Total\_Cost \leftarrow State.Total\_Cost + Action[1]$
Update($New\_State.Path$, $New\_State.Current\_Airport$)
Remove_Visited($New\_State.Remaining\_Zones$, $New\_State.Current\_Airport$)
Add_Visited($New\_State.Visited\_Zones$, $New\_State.Current\_Airport$)
return New_State

Finally, the Best Child function, defined in the Node class is based on the selection function UCB and UCB1_Tuned. They both, compute the score of the visited nodes and pick the one that minimises the selection function.

Algorithm 5 Best Child

Require: $Selection\_Function$
$Visited\_Children \leftarrow$ Children with $visitCount > 0$
$Choices\_Weights \leftarrow [Selection\_Function(child)$ for child in $Visited\_Children]$
$Best\_Child\_Node \leftarrow$ Child with minimum $Choices\_Weights$
return $Best\_Child\_Node$

Chapter 5

Results and performance

(TODO) Present the results and discuss any differences between the findings and your initial predictions/hypothesis

(TODO) Interpret your experimental results - do not just present lots of data and expect the reader to understand it. Evaluate what you have achieved against the aims and objectives you outlined in the introduction

5.1 Hypothesis

As mentionned in Section 1.2, the primary objective was to implement a new algorithm in order to find solutions, focusing on the more realistic instances ($I_1 \dots I_8$) without imposing time constraints.

Hence, simulations for every instances have been conducted, testing different combinations of parameters in what is called a grid search. Each combination of parameters was run 10 times to ensure the reliability and consistency of the results. One challenge, is that the computational budget is limited when using Python. Especially for the more complex instance, ($I_7 \dots I_{14}$) where the time to find a solution for a given set of parameters is more than 20 minutes. It becomes practically impossible to perform for each instance, 10 simulations for every combination of parameters in the grid search. Hence, the size of the grid search for the more complex instances were reduced as shown in Table 5.1.

TABLE 5.1: Grid search

	$(I_1 \dots I_6)$	$(I_7 \dots I_8)$	$(I_9 \dots I_{14})$
selection_policies	top_k, ratio_k	top_k, ratio_k	-
simulation_policies	random, greedy, tolerance	greedy	-
selection_policies	UCB, UCB1T	UCB, UCB1T	-
c_p	0, 1.4, 2.8	1.4	-
N_children	5, 10, 15	10	-
Ratio	0, .3, .5, .8, 1	.5	-

5.2 Results analysis

5.2.1 Overview

After running the various simulations with the search grid parameters defined in Table 5.1, our results were compared with those of Kiwi and RL (Reinforcement Learning) [5] - the only two official publications on this challenge.

TABLE 5.2: Best results vs State of the art

Instance	Kiwi's	RL	Best known	Best found	Mean	Std
I1	1396	1396	1396	**1396**	1396	0
I2	1498	1498	1498	**1498**		
I3	7672	7672	7672	**7672**		
I4	14024	13952	13952	15101	16208.8	520.8
I5	698	690	690	-	-	-
I6	2159	2610	2159	-	-	-
I7	31681	30937	30937	-		
I8	4052	4081	4052	**4037**	-	
I9	76372	75604	75604	-	-	-
I10	21667	58304	21667	-	-	-
I11	44153	59361	44153	-	-	-
I12	65447	86074	65447	-	-	-
I13	97859	166543	97859	-	-	-
I14	118811	198787	118811	-	-	-

A solution was found for $I_1, \dots, I_4$ and I_7, I_8. The results shown in table 5.2 are the best found costs' solution within the grid search. The results of the simulations for $I_1, \dots I_4$ are displayed in Section 9 and the detailed path-solution for these instances can be found in Section 10.

5.2.2 Analysis

5.2.2.1 I1, I2 and I3

For these three instances, the best known solutions were found and the various simulations were carried out successfully. Therefore, the influence of the parameters on the final solution was investigated. For I_3, the analysis focuses on the C_p parameter, the influence of the expansion ratio and finally the study will investigate the overall correlation matrix.

Analysis on C_p

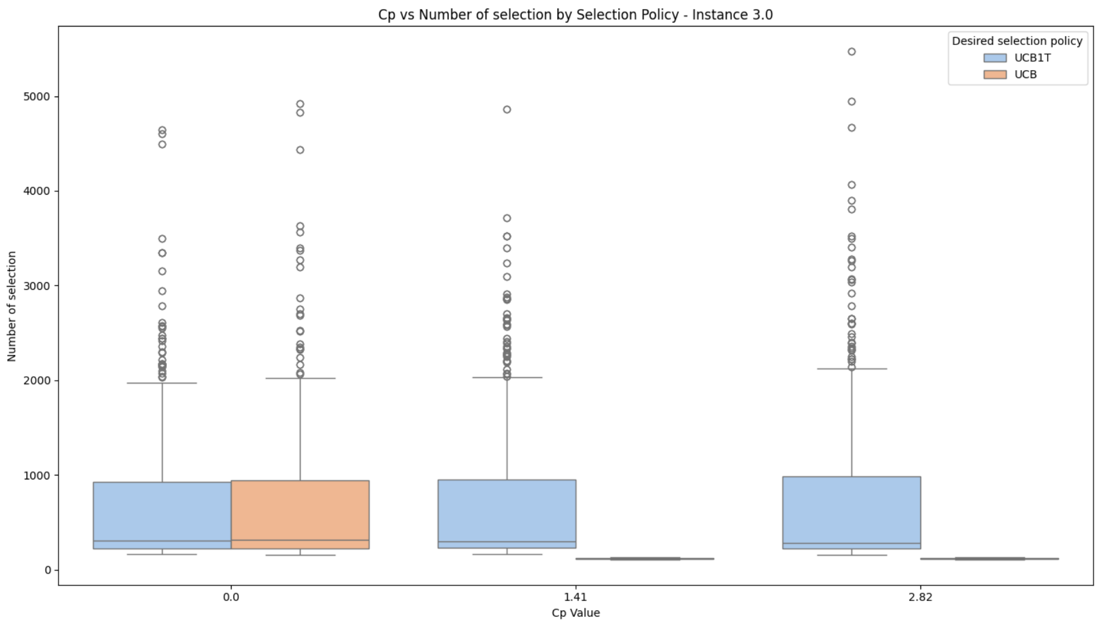


FIGURE 5.1: C_p vs Number of selection

On Figure 5.2 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 the same (cf equation 2.3 and 2.4).
- **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 UCB policy decreases. A higher number of selection for the UCB policy could be expected, as C_p increases but for small instances it convergences faster.

- **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.
- **Balance Between Exploration and Exploitation:** The choice of C_p should be based on the specific problem, balancing the need for exploration (higher C_p) with the desire for quicker convergence (lower C_p).

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, as shown on Figure 5.2. While UCB1T may require more time to converge, it

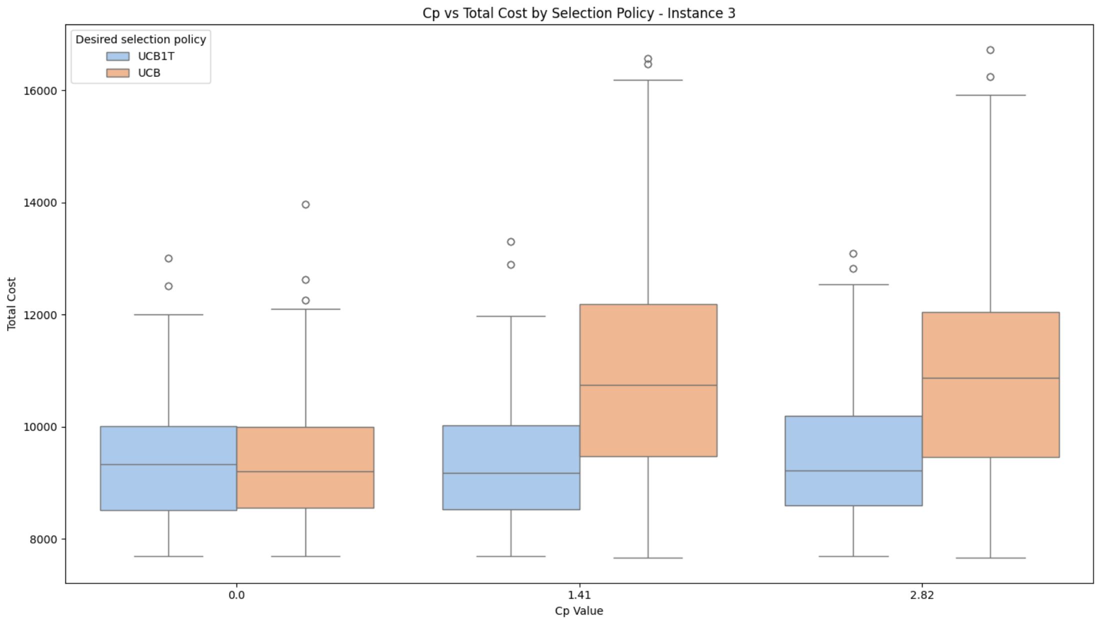


FIGURE 5.2: C_p vs Total cost

generally explores the search tree more effectively, leading to better overall performance.

Analysis of Expansion ratio

The box plots show 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:

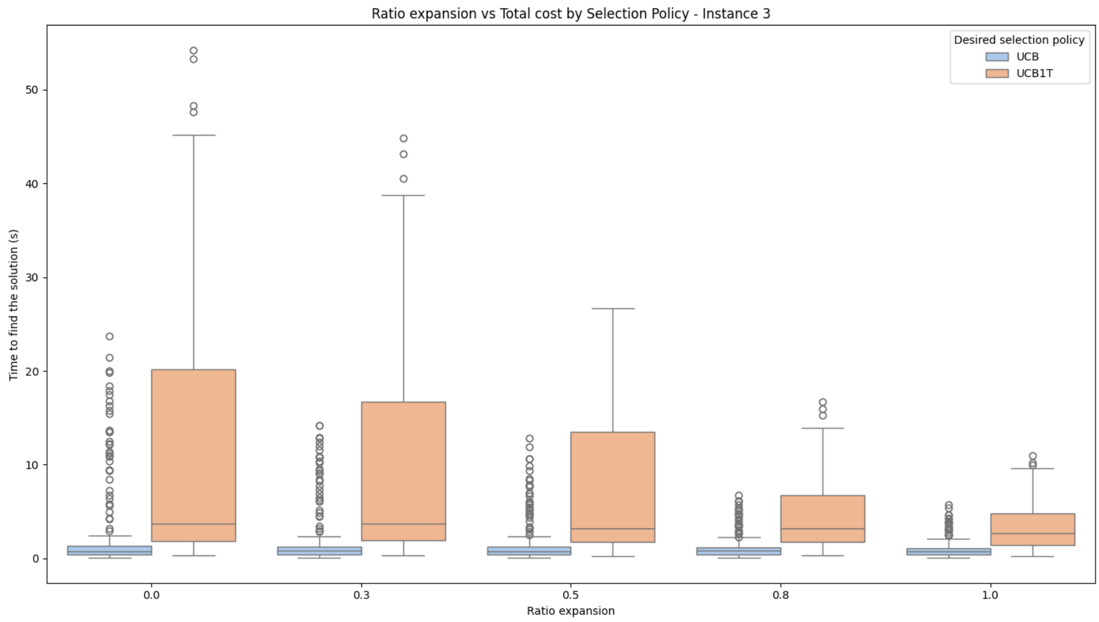


FIGURE 5.3: Ratio expansion vs Time to find the solution

- **UCB is More Time-Efficient:** Across all ratio expansion values, the UCB policy consistently finds solutions more quickly than UCB1T. This suggests that UCB, being less aggressive in exploration, converges on solutions faster.
- **Higher Ratio Expansions Lead to Quicker Solutions:** For both policies, the time to find a solution generally decreases as the ratio expansion increases, indicating a more efficient search process when more expanded nodes lead to solutions. 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 on Figure 5.4.

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 result.

5.2.2.2 I5 and I6

The challenge faced with these two instances is that the defined selection, simulation and expansion policy were not robust enough to explore the tree effectively.

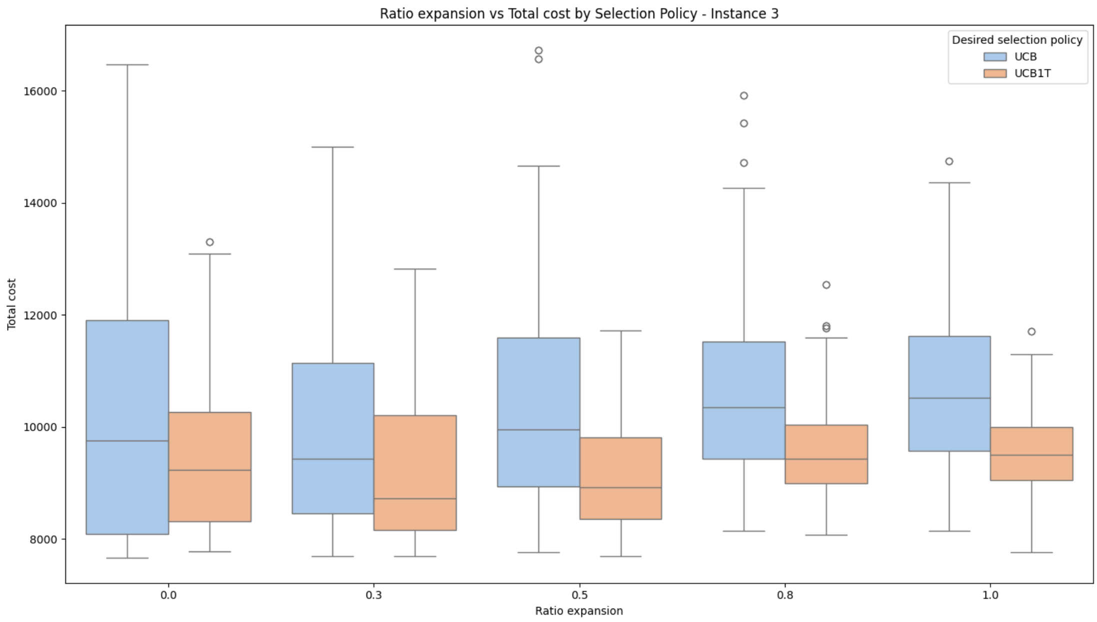


FIGURE 5.4: Expansion ratio vs Total cost

After all the simulations, some nodes in the tree—under the random policy, were able to simulate until reaching a final state. However, due to the randomness of the policy, the tree was pruned at each further iteration where a solution could not be found.

5.2.2.3 I4, I7 and I8

For these instances we have found

Chapter 6

Conclusion

(TODO) Explain what conclusions you have come to as a result of doing this work. Lessons learnt and what would you do different next time. Please summarise the key recommendations at the end of this section, in no more than 5 bullet points.

6.1 Summary of Work

6.2 Critics

6.3 Future Work

(TODO) The References section should include a full list of references. Avoid having a list of web sites. Examiners may mark you down very heavily if your references are mainly web sites.

Chapter 7

Progress and next steps

Selec policy	Exp policy	Simu policy	N° childrens	Ratio	Cp	Best cost	Mean	Std	T(s)
UCB	ratio k	tolerance	10.0	0.0	1.4	1396.0	1396.0	0.0	0.0

Chapter 8

Test Instances

The instances can be found on the following website: https://code.kiwi.com/articles/travelling-salesman-challenge-2-0-wrap-up/

Chapter 9

Simulations results

9.1 Instance 1

Selec policy	Exp policy	Simu policy	N° childrens	Ratio	Cp	Best cost	Mean	Std	T(s)
UCB	ratio k	tolerance	10.0	0.0	1.4	1396.0	1396.0	0.0	0.0
UCB	top k	tolerance	10.0	0.0	2.8	1396.0	1396.0	0.0	0.1
UCB	top k	tolerance	5.0	0.0	1.4	1396.0	1396.0	0.0	0.1
UCB	ratio k	tolerance	5.0	0.0	2.8	1396.0	1531.0	133.8	0.1
UCB	top k	tolerance	15.0	0.8	1.4	1396.0	1577.8	133.0	0.2
UCB	ratio k	tolerance	15.0	0.0	1.4	1396.0	1396.0	0.0	0.2
UCB	top k	tolerance	5.0	1.0	2.8	1396.0	1572.6	117.8	0.2
UCB	top k	tolerance	15.0	0.0	1.4	1396.0	1396.0	0.0	0.2
UCB	top k	tolerance	5.0	1.0	1.4	1396.0	1666.6	148.8	0.2
UCB	ratio k	tolerance	5.0	1.0	1.4	1396.0	1521.7	114.1	0.2
UCB	ratio k	tolerance	15.0	1.0	1.4	1396.0	1643.6	132.6	0.2
UCB	top k	tolerance	5.0	0.0	2.8	1396.0	1396.0	0.0	0.2
UCB	ratio k	tolerance	15.0	0.0	2.8	1396.0	1396.0	0.0	0.2
UCB	ratio k	tolerance	10.0	0.0	2.8	1396.0	1396.0	0.0	0.2
UCB	ratio k	tolerance	10.0	0.8	1.4	1396.0	1596.4	92.2	0.2
UCB	top k	tolerance	10.0	0.0	1.4	1396.0	1396.0	0.0	0.3
UCB	top k	tolerance	15.0	0.0	2.8	1396.0	1396.0	0.0	0.3
UCB	top k	tolerance	10.0	0.5	2.8	1396.0	1555.9	87.7	0.4
UCB	top k	tolerance	5.0	0.3	1.4	1431.0	1518.0	43.2	0.0
UCB	ratio k	tolerance	10.0	0.8	2.8	1431.0	1622.4	159.4	0.1
UCB	top k	tolerance	15.0	0.5	2.8	1431.0	1561.9	95.8	0.1

UCB	ratio k	tolerance	15.0	0.8	1.4	1431.0	1582.4	130.4	0.1
UCB	top k	tolerance	15.0	1.0	1.4	1431.0	1597.3	124.3	0.1
UCB	top k	tolerance	15.0	0.3	0.0	1431.0	1815.8	189.8	0.3
UCB	top k	tolerance	10.0	0.3	2.8	1457.0	1614.4	173.1	0.0
UCB	ratio k	tolerance	5.0	0.3	2.8	1457.0	1566.0	106.9	0.0
UCB	ratio k	tolerance	5.0	0.5	2.8	1457.0	1600.6	113.8	0.1
UCB	top k	tolerance	5.0	0.3	2.8	1457.0	1533.8	86.8	0.2
UCB	ratio k	tolerance	5.0	0.0	1.4	1458.0	1586.0	97.9	0.0
UCB	top k	tolerance	10.0	1.0	1.4	1458.0	1581.1	137.5	0.0
UCB	ratio k	tolerance	10.0	0.5	2.8	1458.0	1570.0	81.5	0.0
UCB	ratio k	tolerance	5.0	1.0	2.8	1458.0	1596.3	100.2	0.1
UCB	ratio k	tolerance	15.0	0.5	2.8	1458.0	1586.6	91.9	0.1
UCB	top k	tolerance	15.0	0.3	1.4	1458.0	1516.2	49.7	0.1
UCB	top k	tolerance	10.0	1.0	2.8	1458.0	1589.1	95.9	0.1
UCB	top k	tolerance	15.0	1.0	2.8	1458.0	1618.7	137.1	0.1
UCB	ratio k	tolerance	15.0	1.0	2.8	1458.0	1624.0	133.5	0.1
UCB	top k	tolerance	15.0	0.5	1.4	1458.0	1654.1	146.8	0.1
UCB	ratio k	tolerance	15.0	0.8	2.8	1458.0	1666.0	108.3	0.1
UCB	top k	tolerance	5.0	0.8	1.4	1458.0	1625.7	105.6	0.2
UCB	top k	tolerance	10.0	0.5	1.4	1458.0	1547.4	78.2	0.2
UCB	top k	tolerance	10.0	0.8	2.8	1458.0	1626.9	129.6	0.2
UCB	top k	tolerance	5.0	0.8	2.8	1458.0	1588.2	79.4	0.2
UCB	top k	tolerance	15.0	0.3	2.8	1472.0	1523.3	56.3	0.2
UCB	top k	tolerance	15.0	0.8	2.8	1472.0	1615.0	99.0	0.2
UCB	top k	tolerance	10.0	0.3	1.4	1472.0	1507.8	38.9	0.2
UCB	ratio k	tolerance	5.0	0.3	0.0	1472.0	1913.6	199.6	0.3
UCB	top k	tolerance	5.0	1.0	0.0	1472.0	1775.0	165.7	0.4
UCB	ratio k	tolerance	15.0	1.0	0.0	1472.0	1786.7	200.4	0.6
UCB	top k	tolerance	15.0	0.8	0.0	1472.0	1787.3	179.2	0.7
UCB1T	top k	tolerance	15.0	0.5	1.4	1472.0	1827.4	227.2	1.0
UCB1T	ratio k	tolerance	10.0	0.8	1.4	1472.0	1819.2	166.9	1.1
UCB	top k	tolerance	10.0	0.8	0.0	1472.0	1798.6	177.2	1.4
UCB1T	top k	tolerance	15.0	0.3	0.0	1472.0	1828.1	215.2	1.8
UCB1T	top k	tolerance	15.0	0.0	2.8	1472.0	1819.1	204.2	2.0
UCB1T	top k	tolerance	5.0	0.5	1.4	1472.0	1785.3	162.4	5.5
UCB1T	top k	tolerance	5.0	0.5	2.8	1472.0	1757.0	223.9	5.9
UCB	ratio k	tolerance	10.0	0.3	1.4	1479.0	1522.3	35.4	0.1
UCB	ratio k	tolerance	5.0	0.3	1.4	1479.0	1528.1	92.9	0.1
UCB	ratio k	tolerance	15.0	0.3	1.4	1479.0	1551.9	60.1	0.1
UCB	ratio k	tolerance	10.0	0.3	2.8	1479.0	1590.3	112.1	0.1
UCB	ratio k	tolerance	15.0	0.3	2.8	1479.0	1624.9	134.5	0.1

UCB1T	ratio k	tolerance	5.0	0.0	0.0	1490.0	1798.1	164.7	0.1
UCB	ratio k	tolerance	10.0	0.5	1.4	1490.0	1628.4	87.8	0.2
UCB	ratio k	tolerance	5.0	0.5	1.4	1490.0	1596.1	72.6	0.2
UCB	top k	tolerance	10.0	0.8	1.4	1493.0	1618.6	91.7	0.1
UCB	top k	tolerance	5.0	0.5	1.4	1493.0	1573.0	68.2	0.1
UCB	ratio k	tolerance	5.0	0.8	2.8	1506.0	1610.3	81.3	0.1
UCB	ratio k	tolerance	10.0	1.0	1.4	1521.0	1629.3	72.7	0.1
UCB1T	top k	tolerance	15.0	0.8	1.4	1521.0	1811.4	188.3	2.0
UCB	ratio k	tolerance	10.0	1.0	2.8	1522.0	1627.1	90.9	0.2
UCB	ratio k	tolerance	5.0	0.8	1.4	1522.0	1641.8	73.5	0.2
UCB	top k	tolerance	5.0	0.5	2.8	1526.0	1577.7	43.3	0.2
UCB	ratio k	tolerance	5.0	0.5	0.0	1529.0	1877.8	199.1	0.2
UCB	ratio k	tolerance	10.0	0.5	0.0	1529.0	1847.4	160.9	0.3
UCB1T	top k	tolerance	15.0	0.8	2.8	1529.0	1811.1	140.9	0.5
UCB1T	top k	tolerance	10.0	0.3	1.4	1529.0	1970.6	254.1	0.8
UCB1T	top k	tolerance	15.0	0.3	1.4	1529.0	1732.0	178.4	1.1
UCB1T	top k	tolerance	5.0	0.3	0.0	1529.0	1780.9	164.4	1.3
UCB1T	ratio k	tolerance	5.0	0.8	2.8	1529.0	1809.0	137.8	1.4
UCB	ratio k	tolerance	15.0	0.3	0.0	1529.0	1956.8	208.3	1.6
UCB1T	ratio k	tolerance	10.0	0.3	0.0	1529.0	1864.2	213.2	1.7
UCB1T	top k	tolerance	5.0	0.0	2.8	1529.0	1824.0	147.9	6.8
UCB	ratio k	tolerance	15.0	0.5	1.4	1530.0	1585.4	55.7	0.1
UCB1T	top k	tolerance	15.0	1.0	0.0	1533.0	1834.1	199.4	2.6
UCB1T	ratio k	tolerance	10.0	1.0	1.4	1533.0	1869.4	181.3	2.9
UCB1T	ratio k	tolerance	5.0	1.0	1.4	1533.0	1850.6	172.3	3.8
UCB1T	top k	tolerance	15.0	0.0	0.0	1540.0	1795.3	202.5	0.2
UCB1T	ratio k	tolerance	10.0	1.0	2.8	1540.0	1851.2	221.6	0.3
UCB	ratio k	tolerance	10.0	0.0	0.0	1540.0	1791.9	160.8	0.5
UCB1T	ratio k	tolerance	10.0	0.5	2.8	1540.0	1800.6	203.9	0.5
UCB	ratio k	tolerance	5.0	1.0	0.0	1540.0	1874.2	157.6	0.6
UCB1T	ratio k	tolerance	15.0	0.0	1.4	1540.0	1815.7	127.5	0.7
UCB	top k	tolerance	15.0	0.0	0.0	1540.0	1885.4	190.7	0.8
UCB	top k	tolerance	10.0	1.0	0.0	1540.0	1880.1	308.8	1.0
UCB	top k	tolerance	10.0	0.3	0.0	1540.0	1876.0	172.9	1.1
UCB	top k	tolerance	10.0	0.5	0.0	1540.0	1938.9	187.7	1.1
UCB1T	ratio k	tolerance	15.0	0.5	1.4	1540.0	1760.2	132.8	1.3
UCB1T	top k	tolerance	15.0	1.0	1.4	1540.0	1887.1	187.8	1.4
UCB1T	ratio k	tolerance	15.0	0.0	2.8	1540.0	1813.4	195.5	1.5
UCB1T	top k	tolerance	10.0	0.5	0.0	1540.0	1893.8	221.5	1.6
UCB1T	top k	tolerance	10.0	0.3	2.8	1540.0	1843.9	155.5	1.6
UCB	top k	tolerance	15.0	0.5	0.0	1540.0	1881.9	220.3	1.8

UCB	top k	tolerance	10.0	0.0	0.0	1540.0	1886.0	166.7	1.8
UCB	ratio k	tolerance	5.0	0.8	0.0	1540.0	1862.7	184.8	2.0
UCB1T	top k	tolerance	5.0	0.8	1.4	1540.0	1813.8	145.1	2.4
UCB1T	top k	tolerance	15.0	1.0	2.8	1540.0	1855.0	159.3	2.6
UCB1T	ratio k	tolerance	15.0	1.0	2.8	1540.0	1799.9	176.5	2.9
UCB1T	ratio k	tolerance	10.0	0.3	2.8	1540.0	1915.6	218.8	3.1
UCB1T	top k	tolerance	10.0	1.0	1.4	1540.0	1938.8	188.1	3.1
UCB1T	top k	tolerance	5.0	0.0	1.4	1540.0	1806.1	151.8	4.2
UCB1T	ratio k	tolerance	5.0	1.0	0.0	1540.0	1830.2	200.5	5.4
UCB	ratio k	tolerance	10.0	0.8	0.0	1544.0	1864.8	240.7	1.1
UCB1T	top k	tolerance	10.0	0.3	0.0	1546.0	1857.7	188.7	2.3
UCB	top k	tolerance	5.0	0.3	0.0	1551.0	1772.9	88.5	0.3
UCB	ratio k	tolerance	10.0	0.3	0.0	1551.0	1826.8	142.9	0.6
UCB1T	ratio k	tolerance	5.0	1.0	2.8	1551.0	1839.2	206.1	1.6
UCB1T	ratio k	tolerance	15.0	0.5	0.0	1551.0	1882.0	170.6	1.7
UCB1T	top k	tolerance	5.0	1.0	1.4	1551.0	1868.1	259.4	2.4
UCB	ratio k	tolerance	5.0	0.0	0.0	1553.0	1936.8	207.6	0.6
UCB	ratio k	tolerance	15.0	0.8	0.0	1553.0	1881.0	203.4	0.9
UCB1T	top k	tolerance	10.0	0.0	0.0	1564.0	1877.2	181.4	0.3
UCB1T	ratio k	tolerance	15.0	0.5	2.8	1564.0	1956.2	224.0	0.3
UCB1T	ratio k	tolerance	10.0	0.8	2.8	1564.0	1814.7	190.9	1.0
UCB1T	top k	tolerance	5.0	0.8	2.8	1564.0	1872.6	147.0	1.9
UCB1T	top k	tolerance	15.0	0.5	2.8	1564.0	1799.9	218.4	2.2
UCB1T	ratio k	tolerance	15.0	0.8	2.8	1564.0	1780.3	168.9	2.3
UCB1T	ratio k	tolerance	5.0	0.8	0.0	1564.0	1849.1	139.3	2.7
UCB1T	ratio k	tolerance	10.0	0.5	0.0	1564.0	1828.4	197.4	2.8
UCB1T	top k	tolerance	5.0	0.3	2.8	1564.0	1812.8	148.3	4.4
UCB1T	top k	tolerance	5.0	1.0	0.0	1564.0	1806.7	129.6	5.7
UCB1T	top k	tolerance	5.0	0.3	1.4	1564.0	1856.6	158.2	6.5
UCB1T	top k	tolerance	5.0	1.0	2.8	1564.0	1861.3	142.9	6.7
UCB1T	top k	tolerance	15.0	0.0	1.4	1573.0	1879.4	206.1	1.8
UCB	top k	tolerance	5.0	0.8	0.0	1578.0	1843.2	130.6	0.5
UCB1T	top k	tolerance	15.0	0.8	0.0	1578.0	1813.1	164.8	1.4
UCB	top k	tolerance	5.0	0.5	0.0	1578.0	1875.3	178.6	1.9
UCB1T	ratio k	tolerance	5.0	0.5	0.0	1615.0	1832.4	183.6	1.5
UCB	ratio k	tolerance	15.0	0.5	0.0	1620.0	1867.1	158.9	0.2
UCB1T	ratio k	tolerance	15.0	0.3	0.0	1620.0	1828.3	161.8	0.7
UCB1T	ratio k	tolerance	5.0	0.3	1.4	1622.0	1847.3	153.3	1.5
UCB1T	ratio k	tolerance	5.0	0.3	2.8	1658.0	1871.2	111.7	0.3
UCB1T	ratio k	tolerance	15.0	0.8	1.4	1662.0	1929.2	225.2	1.3
UCB1T	top k	tolerance	10.0	0.0	2.8	1663.0	1880.1	183.9	1.3

UCB1T	top k	tolerance	10.0	0.5	1.4	1663.0	1865.8	119.5	1.4
UCB1T	top k	tolerance	5.0	0.0	0.0	1674.0	1808.0	85.7	7.4
UCB1T	ratio k	tolerance	10.0	0.8	0.0	1678.0	1862.6	163.0	1.1
UCB	ratio k	tolerance	10.0	1.0	0.0	1678.0	1915.4	132.6	1.5
UCB1T	top k	tolerance	10.0	0.8	1.4	1690.0	1909.8	166.0	1.0
UCB1T	ratio k	tolerance	15.0	0.8	0.0	1690.0	1897.4	127.5	1.8
UCB1T	ratio k	tolerance	5.0	0.3	0.0	1695.0	1941.2	167.6	1.0
UCB1T	top k	tolerance	5.0	0.5	0.0	1695.0	1946.6	148.7	4.7
UCB	ratio k	tolerance	15.0	0.0	0.0	1697.0	1921.8	177.6	1.0
UCB	top k	tolerance	5.0	0.0	0.0	1698.0	1866.7	118.9	3.6
UCB1T	top k	tolerance	15.0	0.3	2.8	1699.0	1920.2	146.3	0.4
UCB1T	ratio k	tolerance	5.0	0.5	1.4	1705.0	1856.6	114.4	1.0
UCB1T	top k	tolerance	10.0	0.8	0.0	1705.0	1851.3	112.4	1.8
UCB1T	top k	tolerance	10.0	1.0	2.8	1708.0	1920.3	181.4	2.6
UCB1T	top k	tolerance	10.0	1.0	0.0	1710.0	1948.2	164.5	0.3
UCB1T	top k	tolerance	10.0	0.5	2.8	1715.0	1934.2	144.1	2.4
UCB1T	ratio k	tolerance	5.0	0.5	2.8	1720.0	1934.9	110.9	2.0
UCB1T	ratio k	tolerance	15.0	1.0	1.4	1720.0	1882.9	123.4	2.6
UCB1T	ratio k	tolerance	5.0	0.8	1.4	1729.0	1884.0	116.8	2.8
UCB1T	top k	tolerance	10.0	0.0	1.4	1739.0	1959.0	148.3	0.6
UCB1T	top k	tolerance	5.0	0.8	0.0	1741.0	1880.4	96.7	7.7
UCB1T	ratio k	tolerance	5.0	0.0	1.4	1745.0	1973.6	173.2	0.6
UCB1T	ratio k	tolerance	10.0	0.0	2.8	1745.0	1987.0	165.1	0.9
UCB1T	top k	tolerance	10.0	0.8	2.8	1751.0	1954.7	190.1	1.1
UCB1T	ratio k	tolerance	15.0	0.3	1.4	1773.0	1920.4	126.9	0.3
UCB1T	ratio k	tolerance	10.0	1.0	0.0	1778.0	1916.1	74.0	0.3
UCB1T	ratio k	tolerance	10.0	0.5	1.4	1778.0	1973.1	125.8	0.6
UCB1T	ratio k	tolerance	5.0	0.0	2.8	1783.0	1972.2	184.5	0.3
UCB1T	ratio k	tolerance	10.0	0.0	0.0	1784.0	1963.9	148.1	1.1
UCB1T	ratio k	tolerance	15.0	0.0	0.0	1786.0	1934.2	125.5	2.5
UCB1T	top k	tolerance	15.0	0.5	0.0	1793.0	1935.1	120.6	2.2
UCB	top k	tolerance	15.0	1.0	0.0	1795.0	1954.8	117.0	0.8
UCB1T	ratio k	tolerance	10.0	0.3	1.4	1798.0	1969.7	198.4	0.6
UCB1T	ratio k	tolerance	15.0	0.3	2.8	1798.0	1944.9	87.6	0.6
UCB1T	ratio k	tolerance	15.0	1.0	0.0	1833.0	2005.4	124.3	2.1
UCB1T	ratio k	tolerance	10.0	0.0	1.4	1856.0	2001.7	118.2	1.6

9.2 Instance 2

9.3 Instance 3

Selec policy	Exp policy	Simu policy	N° childrens	Ratio	Cp	Best cost	Mean	Std	T(s)
UCB	top k	tolerance	10.0	0.0	2.8	7672	7672.0	0.0	0.3
UCB	top k	tolerance	10.0	0.0	1.4	7672	7672.0	0.0	1.2
UCB	ratio k	tolerance	10.0	0.3	2.8	7698	8928.5	988.2	0.2
UCB1T	top k	tolerance	10.0	0.5	2.8	7698	8276.3	291.6	1.6
UCB	ratio k	tolerance	10.0	0.3	0.0	7698	8246.8	370.1	8.2
UCB	top k	tolerance	10.0	0.3	1.4	7703	9027.7	855.5	1.1
UCB1T	ratio k	tolerance	10.0	0.3	1.4	7703	8364.8	323.2	13.1
UCB1T	top k	tolerance	10.0	0.3	0.0	7703	7984.2	220.5	26.1
UCB	top k	tolerance	10.0	0.5	0.0	7768	8435.2	419.3	10.6
UCB1T	ratio k	tolerance	10.0	0.3	2.8	7768	8266.1	263.6	11.1
UCB1T	top k	tolerance	10.0	0.3	2.8	7773	8187.1	296.8	5.3
UCB1T	ratio k	tolerance	10.0	1.0	0.0	7773	9300.7	684.7	7.1
UCB1T	top k	tolerance	10.0	0.0	1.4	7787	7901.3	198.2	10.9
UCB1T	top k	tolerance	10.0	0.3	1.4	7787	8160.6	337.3	16.1
UCB1T	top k	tolerance	10.0	0.0	2.8	7787	7857.4	106.2	35.2
UCB	top k	tolerance	10.0	0.3	0.0	7792	8097.5	285.0	14.2
UCB1T	top k	tolerance	10.0	0.0	0.0	7792	7854.1	81.6	21.2
UCB1T	ratio k	tolerance	10.0	0.3	0.0	7795	8288.7	300.8	14.8
UCB1T	top k	tolerance	10.0	0.5	0.0	7795	8297.6	292.4	22.4
UCB	top k	tolerance	10.0	0.0	0.0	7806	7897.2	124.7	17.5
UCB1T	top k	tolerance	10.0	0.5	1.4	7807	8411.9	432.1	13.1
UCB1T	ratio k	tolerance	10.0	0.5	0.0	7821	8624.7	564.2	15.9
UCB	ratio k	tolerance	10.0	0.5	0.0	7829	8548.9	463.1	3.9
UCB	ratio k	tolerance	10.0	0.3	1.4	7934	8992.4	653.7	0.9
UCB1T	top k	tolerance	10.0	1.0	1.4	7969	9300.1	696.9	8.5
UCB	top k	tolerance	10.0	0.3	2.8	7981	8673.0	487.2	0.1
UCB1T	ratio k	tolerance	10.0	0.5	1.4	8030	8633.6	349.7	8.3
UCB	ratio k	tolerance	10.0	0.0	1.4	8068	9370.3	1063.5	0.5
UCB1T	ratio k	random	10.0	0.5	1.4	8077	9646.8	1175.5	0.3
UCB	top k	random	10.0	0.0	1.4	8083	11851.6	1850.5	0.2
UCB1T	top k	tolerance	10.0	1.0	0.0	8083	9373.6	727.4	0.5
UCB1T	ratio k	tolerance	10.0	1.0	2.8	8083	9460.8	713.7	1.2
UCB1T	ratio k	random	10.0	0.8	0.0	8083	9774.6	713.9	1.4

UCB1T	ratio k	random	10.0	0.8	2.8	8094	9823.5	1208.4	0.3
UCB1T	ratio k	tolerance	10.0	0.8	0.0	8094	9347.3	550.0	7.4
UCB1T	ratio k	tolerance	10.0	0.5	2.8	8101	8682.9	421.1	14.0
UCB1T	ratio k	tolerance	10.0	1.0	1.4	8104	9074.4	482.8	6.8
UCB1T	ratio k	random	10.0	0.3	2.8	8123	10386.4	1130.7	1.3
UCB	top k	random	10.0	0.0	0.0	8131	9774.6	1245.2	1.7
UCB1T	top k	random	10.0	0.0	0.0	8139	9779.3	924.0	1.6
UCB	ratio k	random	10.0	1.0	0.0	8147	9862.4	819.7	1.9
UCB	top k	tolerance	10.0	0.8	0.0	8155	9145.9	603.1	3.6
UCB	top k	random	10.0	0.3	0.0	8164	9478.3	650.0	1.1
UCB1T	top k	random	10.0	0.3	1.4	8193	10077.9	903.4	0.4
UCB	ratio k	tolerance	10.0	0.5	2.8	8219	9410.7	727.1	0.5
UCB1T	ratio k	random	10.0	0.8	1.4	8222	9628.9	859.2	2.3
UCB1T	top k	tolerance	10.0	0.8	0.0	8227	9241.9	557.0	8.9
UCB	top k	tolerance	10.0	0.5	2.8	8248	9611.8	913.2	0.1
UCB	top k	random	10.0	0.5	0.0	8264	9756.5	980.8	1.0
UCB1T	top k	random	10.0	1.0	1.4	8266	9614.8	796.0	1.7
UCB	ratio k	tolerance	10.0	0.0	2.8	8268	9861.9	1242.3	0.6
UCB1T	ratio k	tolerance	10.0	0.8	1.4	8289	9232.9	469.6	6.8
UCB	ratio k	tolerance	10.0	0.5	1.4	8293	9503.2	764.9	0.2
UCB	ratio k	random	10.0	0.8	0.0	8302	9794.7	821.4	1.4
UCB	top k	tolerance	10.0	1.0	0.0	8318	9374.6	550.0	2.5
UCB1T	ratio k	random	10.0	0.5	2.8	8324	9655.1	925.1	0.3
UCB	top k	tolerance	10.0	1.0	2.8	8331	10393.4	1175.3	0.6
UCB	top k	tolerance	10.0	0.5	1.4	8332	9607.5	863.6	1.2
UCB1T	ratio k	tolerance	10.0	0.0	0.0	8346	8942.4	438.9	8.4
UCB1T	top k	random	10.0	0.8	0.0	8352	9765.2	768.4	2.6
UCB	top k	random	10.0	1.0	0.0	8357	9704.9	764.0	1.2
UCB1T	top k	random	10.0	0.8	1.4	8374	9970.7	1074.0	3.0
UCB1T	ratio k	tolerance	10.0	0.0	1.4	8384	9052.0	358.6	4.2
UCB1T	top k	tolerance	10.0	0.8	1.4	8389	9156.9	522.9	3.0
UCB	ratio k	tolerance	10.0	0.8	0.0	8393	9185.4	457.5	1.1
UCB1T	top k	random	10.0	0.8	2.8	8400	10254.1	911.3	0.4
UCB	ratio k	tolerance	10.0	0.0	0.0	8400	8967.8	363.3	11.4
UCB1T	ratio k	random	10.0	0.3	0.0	8410	10341.3	909.2	2.5
UCB1T	ratio k	tolerance	10.0	0.0	2.8	8416	9198.6	509.0	5.0
UCB1T	top k	random	10.0	0.0	2.8	8419	9819.1	746.9	3.9
UCB1T	ratio k	random	10.0	0.5	0.0	8443	9900.9	839.4	2.8
UCB	top k	tolerance	10.0	0.8	2.8	8455	10359.9	1057.4	0.6
UCB1T	top k	random	10.0	1.0	0.0	8461	9999.3	748.9	2.5
UCB1T	ratio k	random	10.0	1.0	0.0	8464	9694.9	829.7	2.0

UCB1T	top k	random	10.0	0.0	1.4	8466	9712.1	813.1	2.6
UCB	ratio k	random	10.0	0.5	0.0	8497	10283.6	989.2	1.3
UCB1T	top k	random	10.0	0.5	0.0	8499	10056.1	899.5	1.3
UCB1T	top k	random	10.0	0.5	1.4	8506	9615.5	669.2	3.4
UCB	ratio k	tolerance	10.0	1.0	0.0	8507	9315.7	495.5	2.7
UCB1T	top k	tolerance	10.0	1.0	2.8	8514	9292.7	502.4	3.4
UCB	top k	random	10.0	0.8	0.0	8536	10240.1	926.6	1.7
UCB1T	ratio k	tolerance	10.0	0.8	2.8	8552	9182.7	336.8	0.7
UCB	ratio k	tolerance	10.0	0.8	2.8	8557	10301.7	841.5	0.4
UCB1T	top k	random	10.0	0.3	2.8	8578	10113.4	810.2	0.3
UCB1T	ratio k	random	10.0	0.3	1.4	8631	10177.4	984.9	3.5
UCB1T	top k	random	10.0	1.0	2.8	8637	9692.7	689.6	0.3
UCB1T	ratio k	random	10.0	1.0	1.4	8672	10043.0	839.8	1.0
UCB	ratio k	random	10.0	0.3	0.0	8740	9895.7	760.9	0.4
UCB1T	top k	tolerance	10.0	0.8	2.8	8748	9196.2	272.7	11.8
UCB	ratio k	random	10.0	0.8	2.8	8769	11392.2	1265.6	0.5
UCB	ratio k	tolerance	10.0	0.8	1.4	8769	10389.8	1255.4	1.1
UCB1T	ratio k	random	10.0	1.0	2.8	8769	9758.3	604.5	1.8
UCB	ratio k	tolerance	10.0	1.0	1.4	8792	10840.1	1221.2	0.8
UCB1T	top k	random	10.0	0.3	0.0	8810	10091.7	874.4	0.3
UCB	ratio k	random	10.0	0.8	1.4	8848	11327.9	1485.9	0.9
UCB	top k	random	10.0	0.8	2.8	8861	11805.1	1534.0	0.9
UCB	top k	tolerance	10.0	0.8	1.4	8874	10947.5	1618.9	0.4
UCB	top k	tolerance	10.0	1.0	1.4	8878	11056.1	1560.8	0.6
UCB1T	ratio k	random	10.0	0.0	1.4	8894	10840.1	1165.3	4.2
UCB1T	ratio k	random	10.0	0.0	0.0	8991	10776.1	1145.5	0.9
UCB1T	top k	random	10.0	0.5	2.8	9145	10006.3	515.8	0.4
UCB	ratio k	tolerance	10.0	1.0	2.8	9153	10667.7	1168.7	0.5
UCB	ratio k	random	10.0	0.0	0.0	9239	11182.3	1200.5	1.4
UCB	top k	random	10.0	0.3	1.4	9310	11662.6	1328.0	0.1
UCB	top k	random	10.0	0.5	2.8	9417	12207.4	1105.5	0.9
UCB	ratio k	random	10.0	1.0	2.8	9475	11612.8	1285.2	0.5
UCB1T	ratio k	random	10.0	0.0	2.8	9519	11286.2	912.8	1.0
UCB	ratio k	random	10.0	0.5	1.4	9594	12251.7	1929.4	1.0
UCB	top k	random	10.0	0.3	2.8	9626	11712.6	1103.5	1.1
UCB	ratio k	random	10.0	0.5	2.8	9943	12714.8	1563.9	0.3
UCB	top k	random	10.0	0.8	1.4	10047	12433.9	1655.6	0.9
UCB	top k	random	10.0	1.0	1.4	10055	11606.0	1101.4	0.3
UCB	top k	random	10.0	1.0	2.8	10088	11859.3	984.4	0.5
UCB	ratio k	random	10.0	0.0	2.8	10117	13609.3	1623.8	0.9
UCB	ratio k	random	10.0	1.0	1.4	10203	11761.7	1145.0	0.2

UCB	ratio k	random	10.0	0.3	1.4	10236	12376.5	1281.9	0.7
UCB	top k	random	10.0	0.5	1.4	10243	12075.0	1077.4	0.5
UCB	top k	random	10.0	0.0	2.8	10369	11852.0	1073.4	0.5
UCB	ratio k	random	10.0	0.3	2.8	10661	12410.7	1256.0	0.3
UCB	ratio k	random	10.0	0.0	1.4	10757	13697.3	1356.7	0.2

Chapter 10

Best solutions

Instance 1

- Starting airport:'ABO'
- Solution = ['AB0', 'AB7', 'AB4', 'AB9', 'AB1', 'AB6', 'AB2', 'AB8', 'AB3', 'AB5', 'AB0']
- Associated cost = 1396

Instance 2

- Starting airport:'EBJ'
- Solution = ['EBJ', 'NBP', 'OMG', 'NCA', 'NUJ', 'OHT', 'GSM', 'EFZ', 'QKK', 'SSC', 'TKT']
- Associated cost = 1498

Instance 3

- Starting airport:'GDN'
- Solution = ['GDN', 'SZY', 'WMI', 'LD3', 'LB1', 'PD1', 'KRK', 'SA1', 'WRO', 'IEG', 'POZ', 'BZG', 'OSZ', 'OSP']
- Associated cost = 7672

Instance 4

- Starting airport:
- Solution: ['GDN', 'SXF', 'CPH', 'OSL', 'BLE', 'TLL', 'HEL', 'LED', 'RIX', 'VNO', 'BQT', 'LWO', 'KIV', 'IAS', 'VAR', 'IST', 'AKT', 'PVK', 'SKP', 'TGD', 'TIA', 'MLA', 'DBV', 'SJJ', 'BEG', 'BUD', 'BTS', 'LJU', 'INN', 'VCE', 'GVA', 'LUX', 'BRU', 'AMS', 'LTN', 'ORK', 'OPO', 'MAD', 'MRS', 'PRG', 'POZ']
- Associated cost: 15101

Instance 5-6

Not found

Instance 7

Instance 8

- Starting airport: 'AEW'
- Solution: ['AEW', 'AUO', 'ZMT', 'TRH', 'IDB', 'LVN', 'FCJ', 'OAE', 'FMC', 'VCO', 'AOY', 'KCY', 'RIS', 'IHK', 'OTQ', 'JBS', 'SXJ', 'ILI', 'JQL', 'MZO', 'TGY', 'PCD', 'CJM', 'DVQ', 'EBC', 'JKB', 'ULO', 'BNL', 'OOM', 'CKW', 'JLS', 'CJT', 'OBE', 'PDI', 'ZZP', 'OVD', 'HRX', 'AZF', 'OLQ', 'WCD', 'XMD', 'IHD', 'FWA', 'NPF', 'FCP', 'RLT', 'NPT', 'BPY', 'YED', 'KIL', 'RGK', 'IYZ', 'ECS', 'CHK', 'IID', 'VRF', 'EBY', 'VDQ', 'ALA', 'CZJ', 'MYR', 'FKP', 'UYS', 'RAA', 'UPZ', 'VFT', 'JEL', 'AKF', 'URK', 'WCU', 'RWZ', 'MVV', 'FGF', 'XSF', 'PRO', 'FYA', 'ZCX', 'VXE', 'KFD', 'CQP', 'JSR', 'EBK', 'RZG', 'LII', 'KIW', 'UEW', 'IXO', 'GHI', 'USB', 'JZU', 'JRX', 'LKE', 'QHR', 'RHQ', 'XSY', 'ASF', 'HPZ', 'CIL', 'EOG', 'JQI', 'QBR', 'PUW', 'PFI', 'WUL', 'PNH', 'TBS', 'LTP', 'RAR', 'DDZ', 'FIG', 'EGV', 'SRY', 'NVV', 'NZN', 'UJW', 'JCY', 'ZNG', 'RWM', 'IUN', 'OPC', 'JRT', 'MHW', 'LTF', 'DRO', 'SVZ', 'QRL', 'BJG', 'BFZ', 'EXV', 'IVF', 'LRU', 'HMM', 'DCY', 'PUG', 'CGR', 'JBJ', 'PEP', 'GSC', 'EHZ', 'CUU', 'BMD', 'PJS', 'GPI', 'BLJ', 'QMS', 'FAO', 'JIM', 'CAA', 'MYZ', 'GRH', 'KBN', 'IPE', 'MMN', 'AUJ', 'LNC', 'ROM', 'JAH', 'DSR', 'HTD', 'EQV', 'NOR', 'RUP', 'OXH',

'BYB', 'BQL', 'EOW', 'PEU', 'JFU', 'MSW', 'DNZ', 'AME', 'JHO', 'HNP', 'LTI', 'PFU', 'QZU', 'RWO', 'LRL', 'KIC', 'MFT', 'EOB', 'QXU', 'QQT', 'BKB', 'AFH', 'MRE', 'MAE', 'BCU', 'PDY', 'ZXD', 'BIN', 'DWQ', 'NRS', 'JJY', 'DSN', 'HIX', 'BAB', 'DCB', 'OVC', 'HIN', 'AEW']

- Associated cost: 4037

Instance 9

Bibliography

[1] Jaap Bouwer Ludwig Hausmann Nina Lind Christophe Verstreken and Stavros Xanthopoulos. Air travel is becoming more seasonal. what steps can airlines take to adapt to the new shape of demand. *McKinsey*, January 8, 2024. URL https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/how-airlines-can-handle-busier-summers-and-comparatively-quiet-winters#/.

[2] Hennie de Harder. Np-what? complexity types of optimization problems explained. *Towards Data Science*, August 17, 2023.

[3] Hendrik Baier and Peter D. Drake. The power of forgetting: Improving the last-good-reply policy in monte carlo go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:303–309, 2010. URL https://api.semanticscholar.org/CorpusID:13578069.

[4] Not mentionned. 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.

[5] Yaroslav Pylyavskyy, Ahmed Kheiri, and Leena Ahmed. A reinforcement learning hyper-heuristic for the optimisation of flight connections. pages 1–8, 07 2020. doi: 10.1109/CEC48606.2020.9185803.

[6] Hanif D. Sherali, Ebru K. Bish, and Xiaomei Zhu. Airline fleet assignment concepts, models, and algorithms. *European Journal of Operational Research*, 172(1): 1–30, 2006. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2005.01.056. URL https://www.sciencedirect.com/science/article/pii/S0377221705002109.

[7] J.E. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers and Operations Research*, 25(7):567–582,

1998. ISSN 0305-0548. doi: https://doi.org/10.1016/S0305-0548(98)00019-7. URL https://www.sciencedirect.com/science/article/pii/S0305054898000197.

[8] Deirdre Fulton. Unstoppable lccs - growth indicates a new norm. https://www.oag.com/blog/unstoppable-lccs-growth-indicates-new-norm, 2023.

[9] FranceTV Slash / Enquêtes. Ryanair: Y-a-t-il un rh dans l'avion? enquête sur les conditions de travail du géant du low-cost, 2024. URL https://www.youtube.com/watch?v=4TOsoX6aPiA. Accessed: 2024-07-05.

[10] Jens Clausen, Allan Larsen, Jesper Larsen, and Natalia J. Rezanova. Disruption management in the airline industry—concepts, models and methods. *Computers and Operations Research*, 37(5):809–821, 2010. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2009.03.027. URL https://www.sciencedirect.com/science/article/pii/S0305054809000914. Disruption Management.

[11] Allison Hope. The complex process behind your flight's schedule. *CNTraveler*, 2017. URL https://www.cntraveler.com/story/the-complex-process-behind-your-flights-schedule#:~:text=Flight%20schedules%20are%20mapped%20out,affect%20departure%20and%20arrival%20times.

[12] Not mentionned. Advanced decision support for aviation disruption management. https://www.inform-software.com/en/lp/aviation-disruption-management#:~:text=Proper%20aviation%20disruption%20management%20means,the%20schedule%2C%20while%20minimizing%20costs., 2024.

[13] Not mentionned. A modern cloud platform to optimize end-to-end airline operations and crew management. iflight drives unmatched efficiencies, cost-savings, and productivity for the world's top airlines. https://www.ibsplc.com/product/airline-operations-solutions/iflight, 2024.

[14] Not mentionned. What is acmi leasing? ACC Aviation, 2024.

[15] Lark Editorial Team. Np hard definition of np hardness. *Lark*, 26 December, 2023.

[16] Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983. ISSN 0167-6377. doi: https://doi.org/10.1016/0167-6377(83)90048-2. URL https://www.sciencedirect.com/science/article/pii/0167637783900482.

[17] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006. ISSN 0305-0483. doi: https://doi.org/10.1016/j.omega.2004.10.004. URL `https://www.sciencedirect.com/science/article/pii/S0305048304001550`.

[18] Snežana Mitrović-Minić and Ramesh Krishnamurti. The multiple tsp with time windows: vehicle bounds based on precedence graphs. *Operations Research Letters*, 34(1):111–120, 2006. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2005.01.009. URL `https://www.sciencedirect.com/science/article/pii/S0167637705000295`.

[19] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2010.03.045. URL `https://www.sciencedirect.com/science/article/pii/S0377221710002973`.

[20] Roberto Tadei, Guido Perboli, and Francesca Perfetti. The multi-path traveling salesman problem with stochastic travel costs. *EURO Journal on Transportation and Logistics*, 6(1):3–23, 2017. ISSN 2192-4376. doi: https://doi.org/10.1007/s13676-014-0056-2. URL `https://www.sciencedirect.com/science/article/pii/S219243762030087X`.

[21] Aviv Adler. The traveling salesman problem under dynamic constraints. *Massachusetts Institute of Technology*, Feb 2023.

[22] Petrică C. Pop, Ovidiu Cosma, Cosmin Sabo, and Corina Pop Sitar. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research*, 314(3):819–835, 2024. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2023.07.022. URL `https://www.sciencedirect.com/science/article/pii/S0377221723005581`.

[23] Hung Chieng and Noorhaniza Wahid. *A Performance Comparison of Genetic Algorithm's Mutation Operators in n-Cities Open Loop Travelling Salesman Problem*, volume 287, pages 89–97. 01 2014. ISBN 978-3-319-07691-1. doi: 10.1007/978-3-319-07692-8_9.

[24] Malik Muneeb Abid and Muhammad Iqbal. Heuristic approaches to solve traveling salesman problem. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 15:390–396, 09 2015. doi: 10.11591/telkomnika.v15i2.8301.

[25] Bernhard Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, 21(3):307–317, 1985. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(85)90151-1. URL `https://www.sciencedirect.com/science/article/pii/0377221785901511`.

[26] Not specified. Travelling salesman problem using dynamic programming. *Geeksforgeeks*, 19 April, 2023.

[27] Daniel Rosenkrantz, Richard Stearns, and Philip II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 09 1977. doi: 10.1137/0206041.

[28] Zakir Ahmed. Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *International Journal of Biometric and Bioinformatics*, 3, 03 2010. doi: 10.14569/IJACSA.2020.0110275.

[29] Lei Yang, Xin Hu, Kangshun Li, Weijia Ji, Qiongdan Hu, Rui Xu, and Dongya Wang. *Nested Simulated Annealing Algorithm to Solve Large-Scale TSP Problem*, pages 473–487. 05 2020. ISBN 978-981-15-5576-3. doi: 10.1007/978-981-15-5577-0_37.

[30] Yong Wang and Zunpu Han. Ant colony optimization for traveling salesman problem based on parameters optimization. *Applied Soft Computing*, 107:107439, 2021. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2021.107439. URL `https://www.sciencedirect.com/science/article/pii/S1568494621003628`.

[31] Wikipedia. Havannah (board game) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Havannah%20(board%20game)&oldid=1240631485`, 2024. [Online; accessed 18-August-2024].

[32] Wikipedia. Game of the Amazons — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Game%20of%20the%20Amazons&oldid=1235225698`, 2024. [Online; accessed 18-August-2024].

[33] Wikipedia. Lines of Action — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Lines%20of%20Action&oldid=1198717858`, 2024. [Online; accessed 18-August-2024].

[34] Wikipedia. Shogi — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Shogi&oldid=1240175752`, 2024. [Online; accessed 18-August-2024].

[35] Joris Duguépéroux, Ahmad Mazyad, Fabien Teytaud, and Julien Dehos. Pruning playouts in monte-carlo tree search for the game of havannah. volume 10068, pages 47–57, 06 2016. ISBN 978-3-319-50934-1. doi: 10.1007/978-3-319-50935-8_5.

[36] Richard J. Lorentz. Amazons discover monte-carlo. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Computers and Games*, pages 13–24, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87608-3.

[37] Mark Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:239 – 250, 12 2010. doi: 10.1109/TCIAIG.2010.2061050.

[38] Wikipedia. Go (game) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Go%20(game)&oldid=1239511822`, 2024. [Online; accessed 18-July-2024].

[39] Wikipedia. Lee Sedol — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Lee%20Sedol&oldid=1234296689`, 2024. [Online; accessed 11-August-2024].

[40] Google DeepMind. Alphago - the movie / full award-winning documentary. Youtube, 2020.

[41] Not mentionned. Explain the role of monte carlo tree search (mcts) in alphago and how it integrates with policy and value networks. EITCA, 2024.

[42] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1:1–43, 03 2012. doi: 10.1109/TCIAIG.2012.2186810.

[43] at the University of Strathclyde John Levine for his class CS310: Foundations of Artificial Intelligence. Monte carlo tree search, 2017. URL `https://www.youtube.com/watch?v=UXW2yZndl7U&t=385s`. Accessed: June, 2024.

[44] Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, S. Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions*

on Computational Intelligence and AI in Games, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

[45] Rowaina Abdelnasser. Python naming conventions: 10 essential guidelines for clean and readable code. `https://medium.com/@rowainaabdelnasser/python-naming-conventions-10-essential-guidelines-for-clean-and-readable-code-fe8 ~:text=For%20class%20names%20in%20Python,or%20behavior%20of%20the%20class.`, June 6, 2023.