



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

<b>Declaration of Authorship</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>Abbreviations</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Research objectives . . . . .	2
1.3 Academic publication . . . . .	2
1.4 Dissertation structure . . . . .	2
<b>2 Literature Review</b>	<b>4</b>
2.1 Optimisation in Air Travel . . . . .	4
2.1.1 Fleet Assignment Problem . . . . .	4
2.1.2 Crew Scheduling Problem . . . . .	5
2.1.3 Disruption Management . . . . .	5
2.1.4 Airline adaptation to new demand . . . . .	6
2.2 Traveling Salesman problem and its adapation . . . . .	8
2.3 The Monte Carlo Tree Search algorithm . . . . .	11
2.3.1 Overview . . . . .	11
2.3.2 Example . . . . .	12
2.3.3 The different parameters in the MCTS . . . . .	18
2.4 Previous study on this challenge . . . . .	19
<b>3 Problem Description</b>	<b>21</b>
3.1 Overview . . . . .	21

3.2	Instances . . . . .	24
3.2.1	Description . . . . .	24
3.2.2	General formulation . . . . .	26
3.2.3	Kiwi's rules . . . . .	27
<b>4</b>	<b>Methodology</b>	<b>29</b>
4.1	Monte Carlo Tree Search implementation . . . . .	30
4.1.1	General flow . . . . .	30
4.1.1.1	Data Preprocessing . . . . .	32
4.1.1.2	Node . . . . .	34
4.2	The different policies . . . . .	35
4.2.1	Simulation policies . . . . .	35
4.2.2	Expansion policies . . . . .	36
4.2.3	Pseudo-code . . . . .	37
<b>5</b>	<b>Results and performance</b>	<b>40</b>
5.1	Hypothesis . . . . .	40
5.2	Results analysis . . . . .	41
5.2.1	Overview . . . . .	41
5.2.2	Analysis . . . . .	42
5.2.2.1	I1, I2 and I3 . . . . .	42
5.2.2.2	I5 and I6 . . . . .	44
5.2.2.3	I4, I7 and I8 . . . . .	45
<b>6</b>	<b>Conclusion</b>	<b>46</b>
6.1	Summary of Work . . . . .	46
6.2	Critics . . . . .	46
6.3	Future Work . . . . .	46
<b>7</b>	<b>Draf</b>	<b>47</b>
<b>A</b>	<b>Code Listings</b>	<b>48</b>
A.1	Data preprocessing . . . . .	48
A.2	Node . . . . .	54
A.3	MCTS . . . . .	61
<b>B</b>	<b>Test Instances</b>	<b>78</b>
<b>C</b>	<b>Simulations results</b>	<b>79</b>
C.1	Instance 1 . . . . .	79
C.1.1	Solution found . . . . .	79
C.1.2	Solution not found . . . . .	92
C.2	Instance 2 . . . . .	93
C.2.1	Solution found . . . . .	93

---

C.2.2	Solution no found . . . . .	100
C.3	Instance 3 . . . . .	107
<b>D</b>	<b>Best solutions</b>	<b>111</b>

# List of Figures

2.1	European demand seasonality [1]	6
2.2	Time complexity of different functions	8
2.3	Assymetrical growth of MCTS - Simulation and Expansion - [2]	12
2.4	Selection - $I1$	12
2.5	Simulation - $I1$	13
2.6	Backpropagation - $I1$	15
2.7	Selection - $I2$	15
2.8	Simulation and Backpropagation - $I2$	16
2.9	Selection - $I3$	16
2.10	Selection and Expansion - $I3$	17
2.11	Simulation and Backpropagation - $I3$	17
2.12	Selection - Simulation - Backpropagation - $I4$	18
4.1	Flow MCTS	30
4.2	Explanation of the data preprocessing class	32
4.3	Explanation of the Node class	34
5.1	$C_p$ vs Number of selection	42
5.2	$C_p$ vs Total cost	43
5.3	Ratio expansion vs Time to find the solution	44
5.4	Expansion ratio vs Total cost	45



# List of Tables

2.1	Kiwi TSP 2.0 - Chosen algorithm of the state of the art solutions . . . . .	20
2.2	Kiwi TSP 2.0 - State of the art solution . . . . .	20
3.1	Flight connections sample I6 . . . . .	25
3.2	Time limits based on the number of areas and airports . . . . .	27
3.3	Instances and their respective parameters . . . . .	28
5.1	Grid search . . . . .	41
5.2	Best results vs State of the art . . . . .	41

# 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 [3], 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 [4], 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 characterised 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

The goals of this dissertation are:

- The implementation of a Python innovative solution to solve the Kiwi.com Traveling Salesman problem 2.0 with no focus on the time limit.
- Try to find better solutions than the state of the art for the considered instances.

## 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 [5] 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 [6], 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) [7], 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. [8]

### 2.1.3 Disruption Management

Disruptions in airline operations, as noted in [9], 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 [10], effective disruption management is crucial to minimise the impact on passengers and overall airline operations.

The two main 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 [11] and [12], 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.

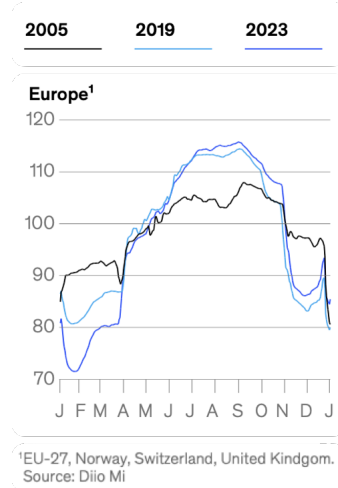


FIGURE 2.1: European demand seasonality [1]

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 operations 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 [13]) 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 adaptation

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 salesman that has to travel around a given number of cities while minimizing the overall journey's distance. This problem is characterised as  $\mathcal{NP}$ -Hard [14]. 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.

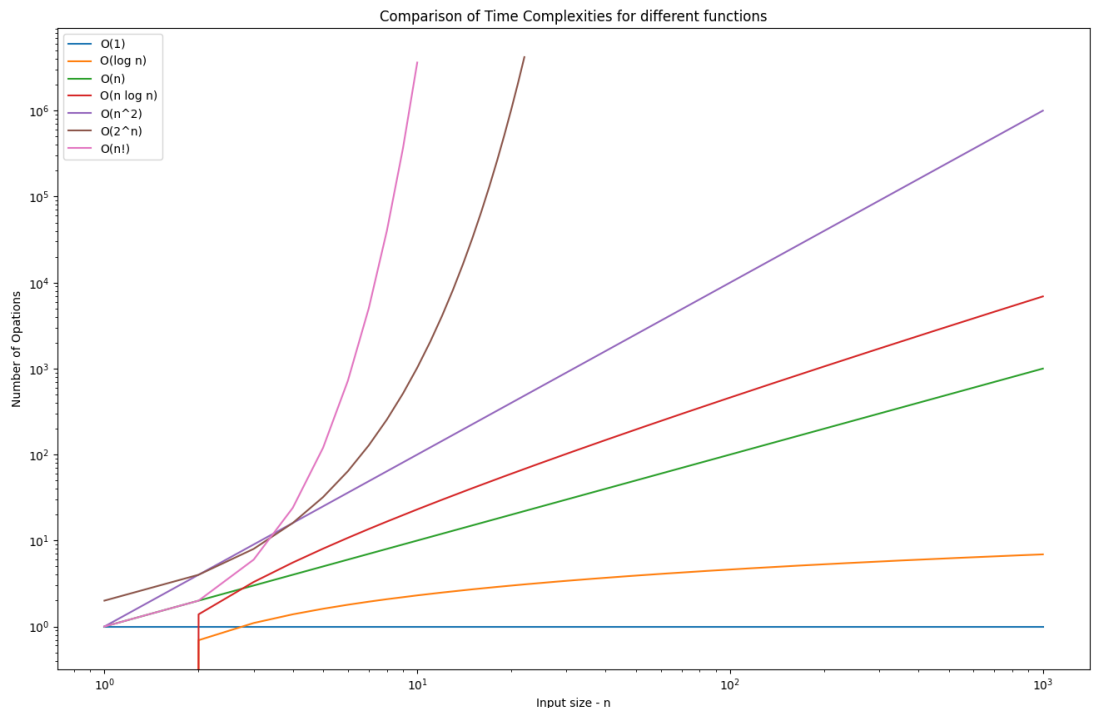


FIGURE 2.2: Time complexity of different functions

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 reasonable 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.
- Assymmetric TSP (ATSP): The distance between cities are assymmetric, 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. [15]
- 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. [16]
- Time Window TSP (TWTSP): Each city has to be visited in a defined time slot. [17]
- 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. [18]
- Stochastic TSP (STSP): The distances between the cities or the cost of travels are stochastic (i.e random variables) rather than deterministic. [19]
- 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. [20]
- Generalised TSP (GTSP): The cities are grouped into clusters, the goal is to visit exactly one city from each cluster. [21]
- Open TSP (OTSP): The traveler does not have to end his journey at the starting city. [22]

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. [23]

- **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. [24]
- **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. [25]
- **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. [26]
  - **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. [27]
  - **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. [28]
  - **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 are gradually refined through updates based on pheromone trails. [29]

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 traditional 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 [30], Amazons [31], Lines of Actions [32], Go, Chess, and Shogi [33], establishing it as the state-of-the-art algorithm [34], [35], [36]. 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 [37]. In 2016, Lee Sedol [38] - the best Go's player in the world was been beaten by AlphaGo 4-1 [39].

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. [40]

### 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. [41]

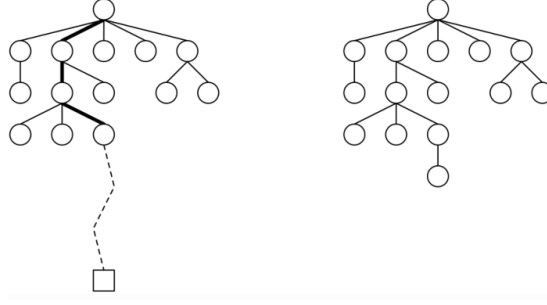


FIGURE 2.3: Assymetrical growth of MCTS - Simulation and Expansion - [2]

To ensure a clearer understanding of MCTS algorithm's stages, we will start by exploring a detailed example [42]. 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

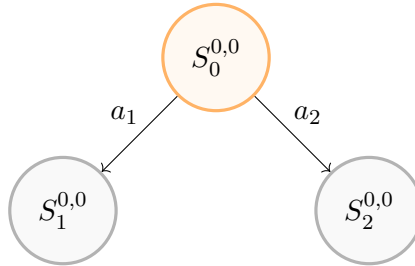


FIGURE 2.4: Selection - I1

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:

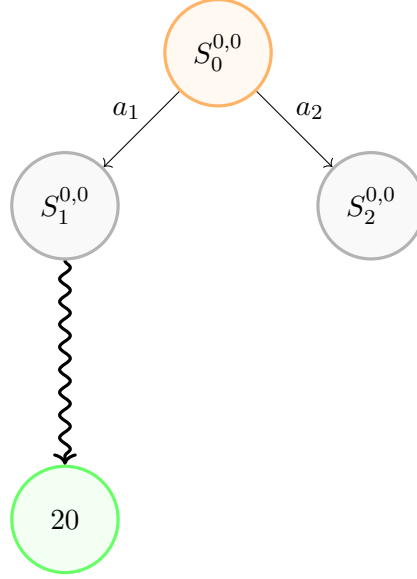


FIGURE 2.5: Simulation - I1



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:

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

- After backpropagation:

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

We can then define a backpropagation function:

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

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

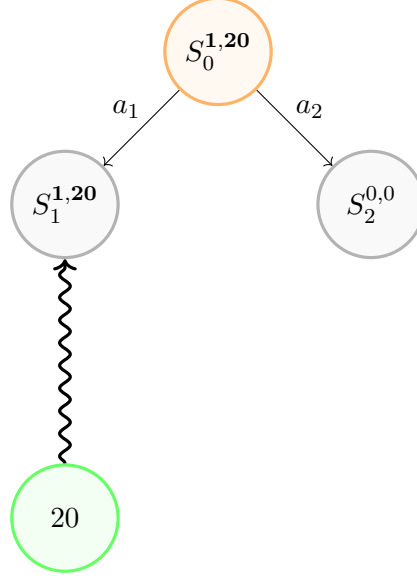


FIGURE 2.6: Backpropagation - I1

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

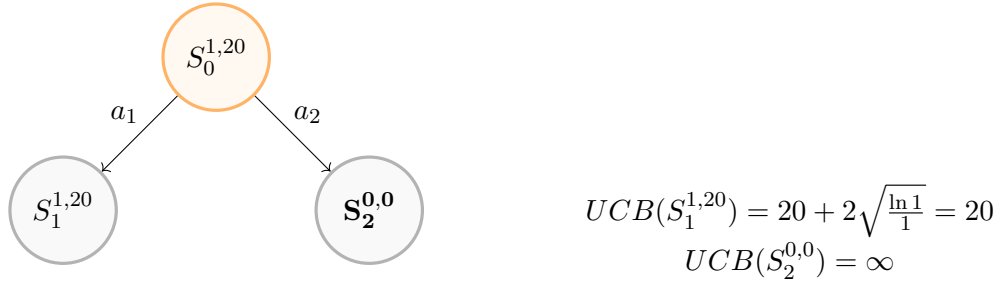


FIGURE 2.7: Selection - I2

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.

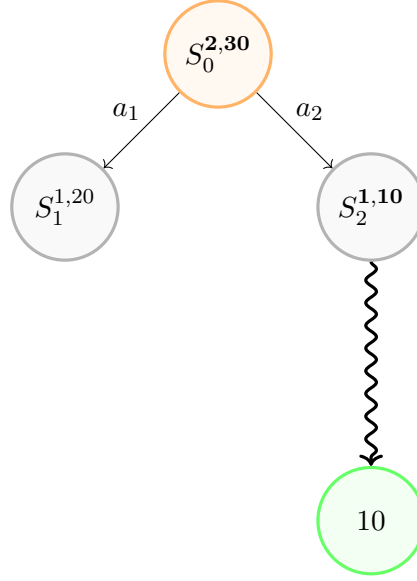


FIGURE 2.8: Simulation and Backpropagation - I2

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  $\mathcal{B}(S_1^{1,20}) = S_0^{2,30}$ . Next, we start the 3<sup>rd</sup> iteration, based on the *UCB* score we decide to choose  $a_1$ .

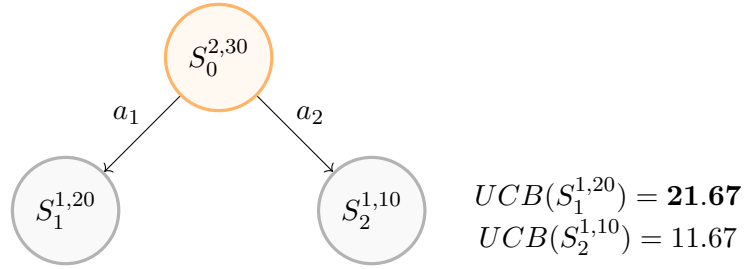


FIGURE 2.9: Selection - I3

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

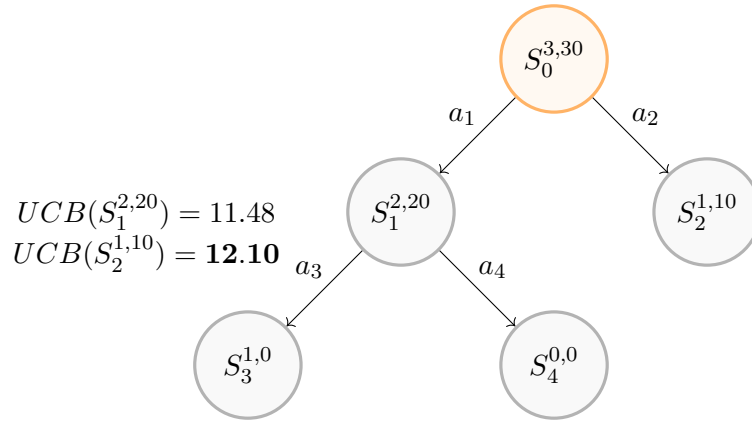


FIGURE 2.10: Selection and Expansion - I3

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

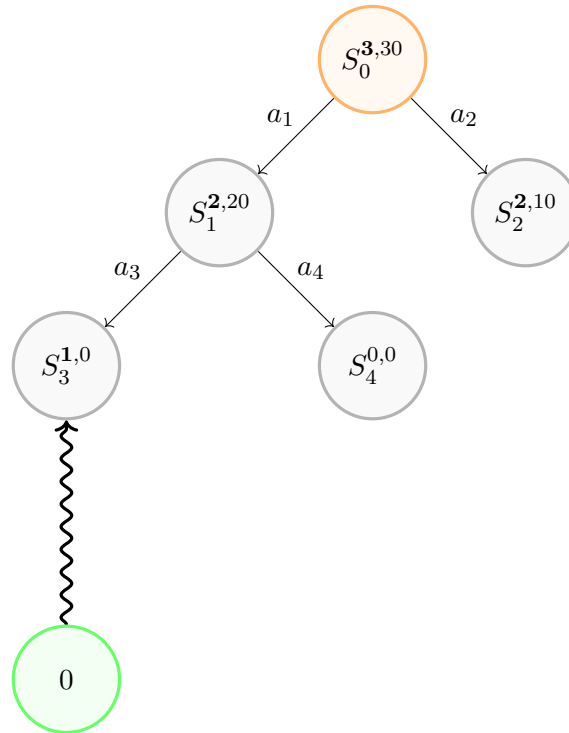


FIGURE 2.11: Simulation and Backpropagation - I3

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

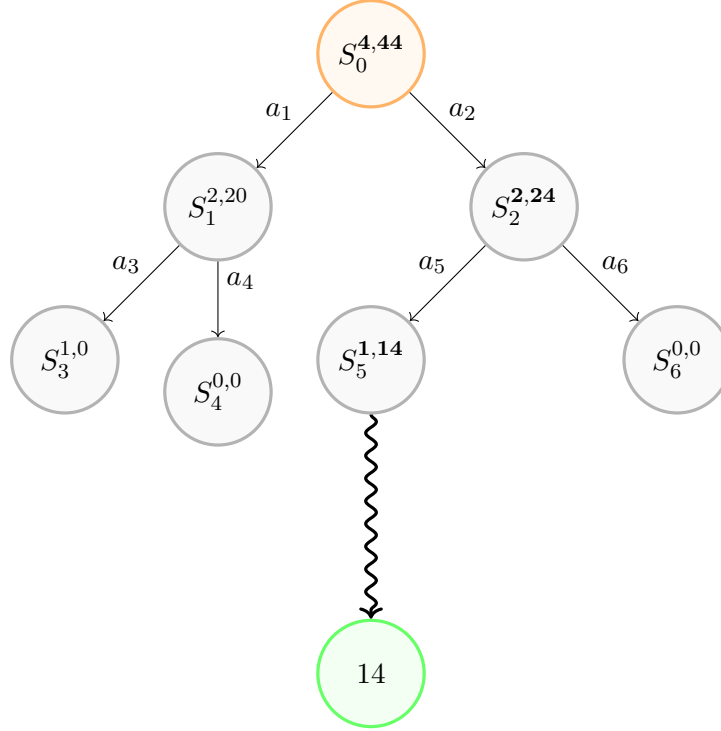


FIGURE 2.12: Selection - Simulation - Backpropagation - I4

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 traditionally used is the Upper Confidence Bound 1 (UCB). However, there are a lot of different MCTS' selection functions as mentioned in this survey [43]. 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 its variants, the UCB1-Tuned are defined as follow:

$$UCB = \bar{X}_i + C_p \sqrt{\frac{2 \ln N}{n_i}} \quad (2.3)$$

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

Where:

- $\bar{X}_i$ : 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
- $\text{Var}(X_i)$ : 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  $\min \left( \frac{1}{4}, \text{Var}(X_i) + \sqrt{\frac{2 \ln N}{n_i}} \right)$ , 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.

## 2.4 Previous study on this challenge

More than 500 teams registered for Kiwi.com TSP 2.0 challenge, and only 100 teams developed algorithms that were robust to go to the second phase of the challenge. The

literature on the methods used by the participants is relatively sparse compared to the number of competitors. On the Kiwi.com website, the best instances are showcased. The winners explained their approach during the award ceremony [44], they implemented a Breadth-first search (BFS) algorithm in C++. Other participants employed well-known heuristic such as modified Simulated Annealing, Genetic Algorithms and Reinforcement Learning hyper-heuristics. Two papers were published on this challenge, [4] and [45] where a Local Search and a Reinforcement Learning (RL) hyper-heuristics algorithms are implemented. The algorithms used by the known participants to solve this challenge are summarised on Table 2.1.

TABLE 2.1: Kiwi TSP 2.0 - Chosen algorithm of the state of the art solutions

References	Reinforcement Learning	Local search	BFS	SA
Paper 1 [4]	x			
Paper 2 [45]		x		
Kiwi's official winner			x	
Other participants	x	x	x	x

Furthermore, Table 2.1 summarises the state of the art solutions.

TABLE 2.2: Kiwi TSP 2.0 - State of the art solution

Instance	Kiwi's	Local Search	Reinforcement learning	Best known
I1	1396	1396	1396	1396
I2	1498	1498	1498	1498
I3	7672	7672	7672	7672
I4	14024	14045	13952	13952
I5	698	837	690	690
I6	2159	3021	2610	2159
I7	31681	32354	30937	30937
I8	4052	4041	4081	4041
I9	76372	82242	75604	75604
I10	21667	87462	58304	21667
I11	44153	49453	59361	44153
I12	65447	70082	86074	65447
I13	97859	-	166543	97859
I14	118811	-	198787	118811

These points motivate our selection of Monte Carlo Tree Search as a suitable approach for this challenge.

## 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 necessarily 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 necessarily 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:

- $\mathcal{A} = \{1, 2, \dots, N\}$ : Set of areas.
- $A_j = \{a_{j,1}, a_{j,2}, \dots, a_{j,k_j}\}$ : Set of 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

- $c_{ij}^d$ : 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  $d$ :

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

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

- Binary variable constraints:

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

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

## 3.2 Instances

### 3.2.1 Description

We are given a set of 14 Instances  $I_n = \{I_1, I_2, \dots, I_{13}, I_{14}\}$  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, starting at 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  $k$ :

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

– **Columns:**

- \* Departure Airport:  $a_{i,k,j}^d$  (Departure airport for the  $j$ -th flight on day  $k$ )
- \* Arrival Airport:  $a_{i,k,j}^a$  (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	$\leq 20$	$< 50$	3
Medium	$\leq 100$	$< 200$	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	atrdrwkbz - AEW	200	1 - 4	300	15
I9	fcjsqtmccq - GVT	250	1 - 1	250	15
I10	eqlfrvhlwu - ECB	300	1 - 1	300	15
I11	pbggaejrjv - LIJ	150	1 - 4	200	15
I12	unnwaxhnoq - PJE	200	1 - 4	250	15
I13	hpkogdfpf - 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:

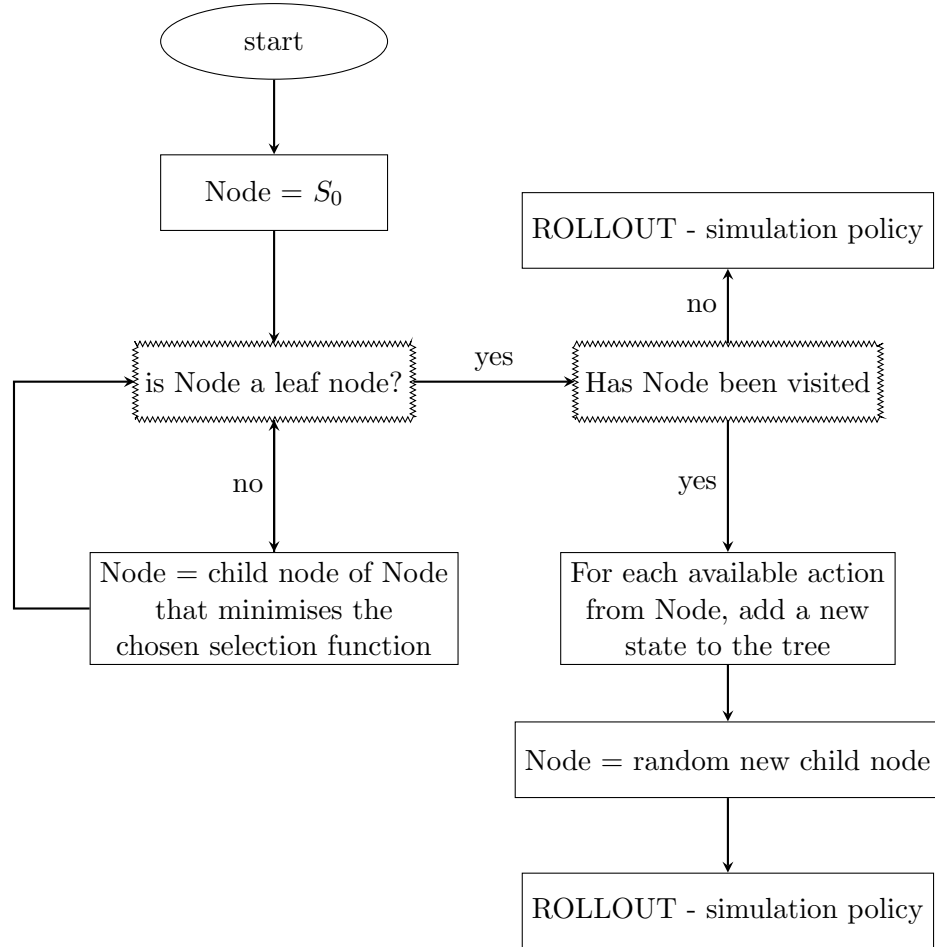


FIGURE 4.1: Flow MCTS

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 necessarily 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.

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

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 [46]. This `data_preprocessing` class is represented on Figure 4.2. The input is an instance  $I_i$ , as defined in Chapter 3:

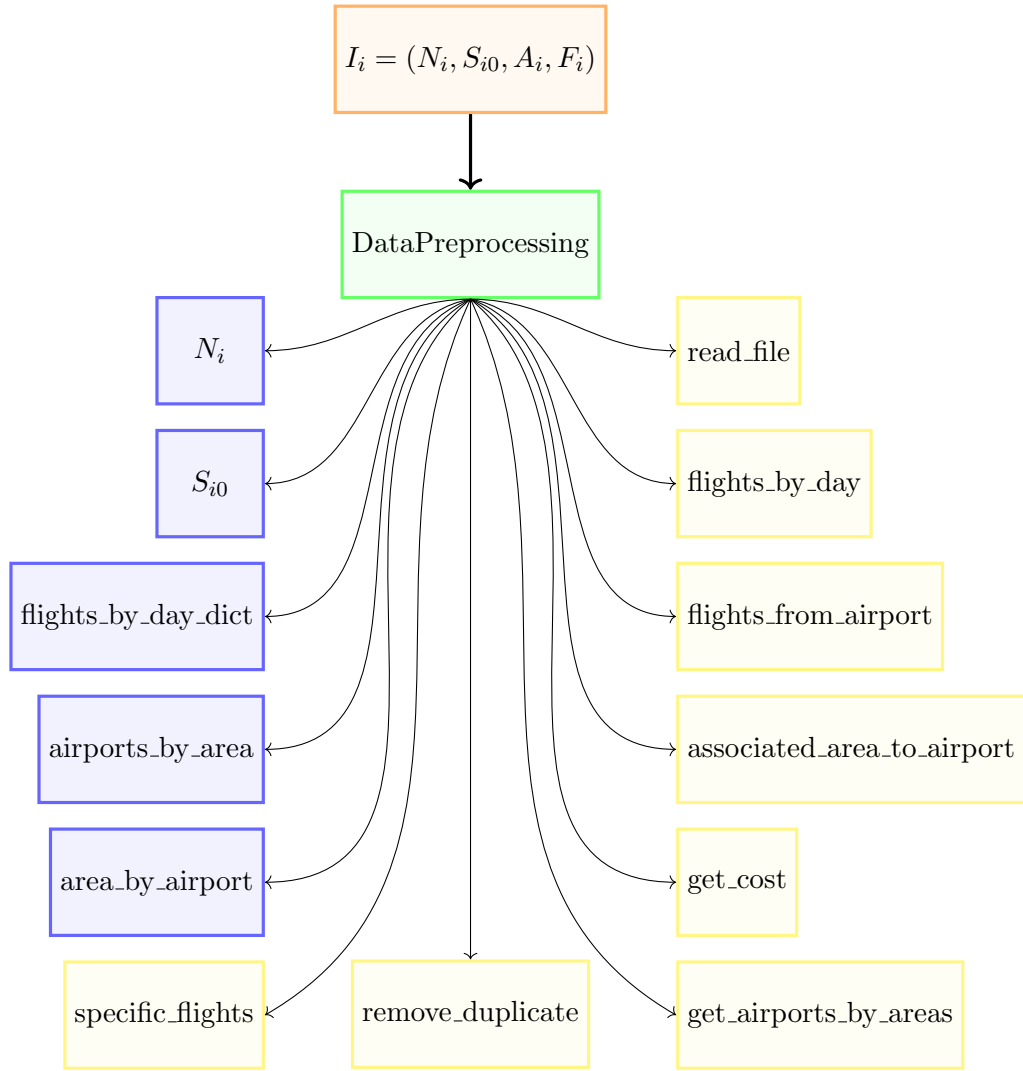


FIGURE 4.2: Explanation of the data preprocessing class

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, dictionaries are used where possible. Dictionaries 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

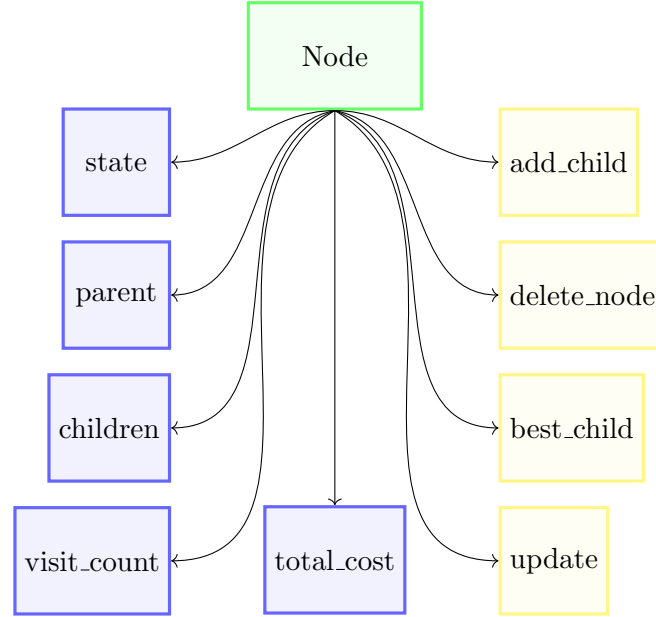


FIGURE 4.3: Explanation of the Node class

As mentioned 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 dictionary 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 core 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 literature 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 combination 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  $(1 - c) * \text{number\_of\_children}$  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
 

---

```

1: Initialise Root_Node with Initial_State
2: while Tree is not fully explored do
3:    $Node \leftarrow \text{Select}(Root\_Node)$ 
4:   if Node is not fully expanded then
5:      $Node \leftarrow \text{Expand}(Node)$ 
6:   end if
7:    $Cost \leftarrow \text{Simulate}(Node)$ 
8:    $\text{Backpropagate}(Node, Cost)$ 
9: end while
10: 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

---

```

1: Input: Node
2: Current  $\leftarrow$  Node
3: while Current.Children is not empty do
4:   if Current is not fully expanded then
5:     UnvisitedChildren  $\leftarrow$  Children with VisitCount = 0
6:     if UnvisitedChildren is not empty then
7:       SelectedChild  $\leftarrow$  Randomly select from UnvisitedChildren
8:       return True, SelectedChild
9:     end if
10:  else
11:    Current  $\leftarrow$  BestChild(Current)
12:  end if
13: end while
14: if Current.Children is empty and Current.State["current_day"] ==  $N_{Areas}$  then
15:  return False, Current
16: else if Current.Children is empty and Current.State["current_day"]  $\neq N_{Areas}$ 
    then
17:  return False, Current
18: else if Current.State["current_day"] ==  $N_{Areas} + 1$  then
19:  return True, Current
20: 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

---

```

1: while Node is not None do
2:   Node.Update(Cost)
3:   Node  $\leftarrow$  Node.Parent
4: 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

---

```

1:  $New\_State \leftarrow \text{Copy of } State$ 
2:  $New\_State.Current\_Day \leftarrow State.Current\_Day + 1$ 
3:  $New\_State.Current\_Airport \leftarrow Action[0]$ 
4:  $New\_State.Total\_Cost \leftarrow State.Total\_Cost + Action[1]$ 
5:  $Update(New\_State.Path, New\_State.Current\_Airport)$ 
6:  $Remove\_Visited(New\_State.Remaining\_Zones, New\_State.Current\_Airport)$ 
7:  $Add\_Visited(New\_State.Visited\_Zones, New\_State.Current\_Airport)$ 
8: 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*

```

1:  $Visited\_Children \leftarrow \text{Children with } visitCount > 0$ 
2:  $Choices\_Weights \leftarrow [Selection\_Function(child) \text{ for } child \text{ in } Visited\_Children]$ 
3:  $Best\_Child\_Node \leftarrow \text{Child with minimum } Choices\_Weights$ 
4: 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 mentioned in Section 1.2, the primary objective was to implement a new algorithm in order to find solutions without imposing time constraints. Only instances  $(I_1 \dots I_8)$  are studied because they represent realistic scenarios.

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

	$(I_1 \dots I_6)$	$(I_7 \dots I_8)$	$(I_9 \dots I_{14})$
<i>selection_policies</i>	top_k, ratio_k	top_k, ratio_k	-
<i>simulation_policies</i>	random, greedy, tolerance	greedy	-
<i>selection_policies</i>	UCB, UCB1T	UCB, UCB1T	-
<i>c_p</i>	0, 1.4, 2.8	1.4	-
<i>N_children</i>	5, 10, 15	10	-
<i>Ratio</i>	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) [4] - 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	<b>1396</b>	1396	0
I2	1498	1498	1498	<b>1498</b>		
I3	7672	7672	7672	<b>7672</b>		
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	<b>4037</b>	-	
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 C and the detailed path-solution for these instances can be found in Section D.

## 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 $C_p$

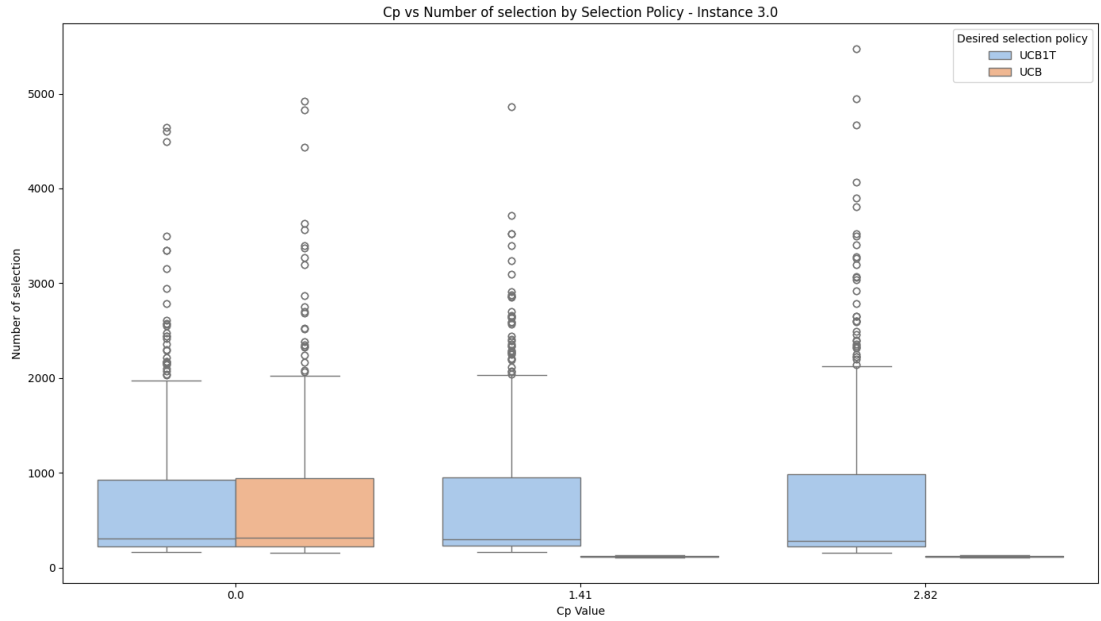


FIGURE 5.1:  $C_p$  vs Number of selection

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

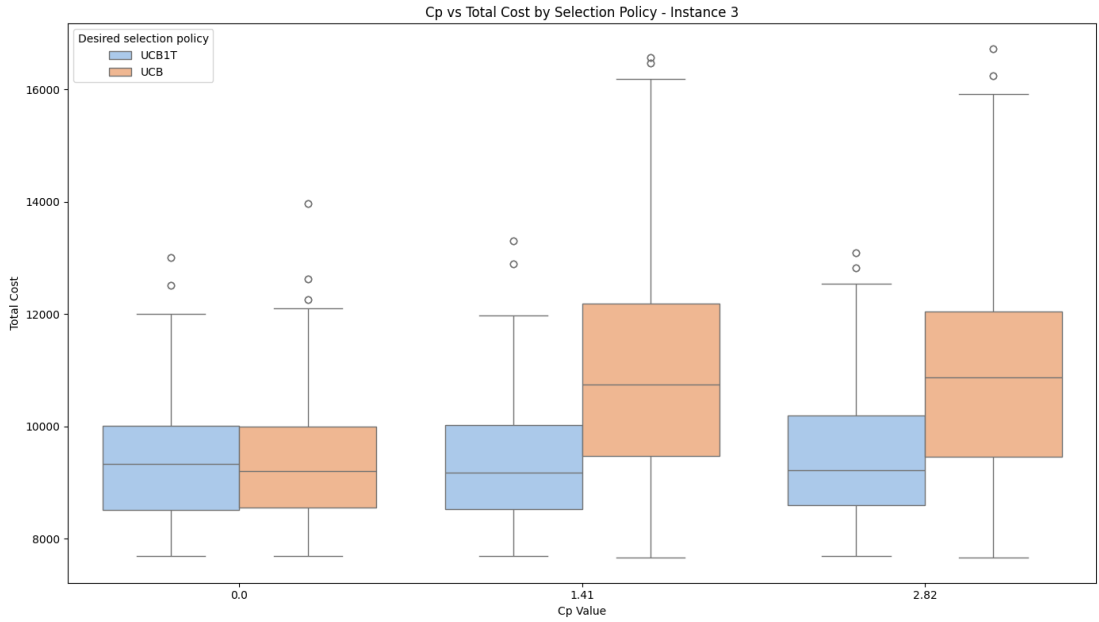


FIGURE 5.2:  $C_p$  vs Total cost

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:

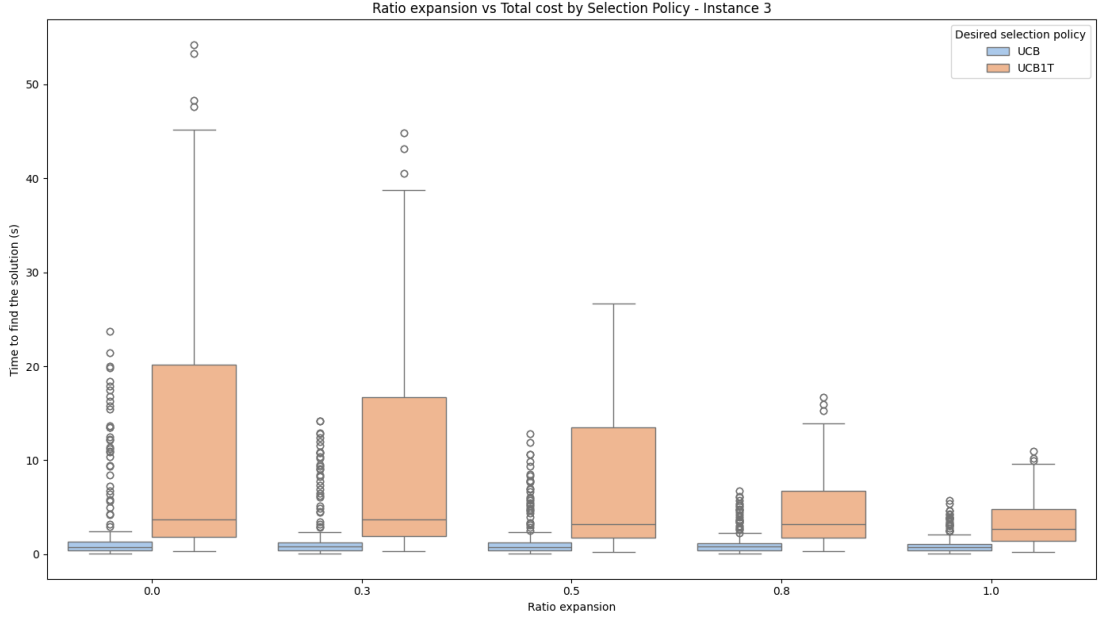


FIGURE 5.3: Ratio expansion vs Time to find the solution

- **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.

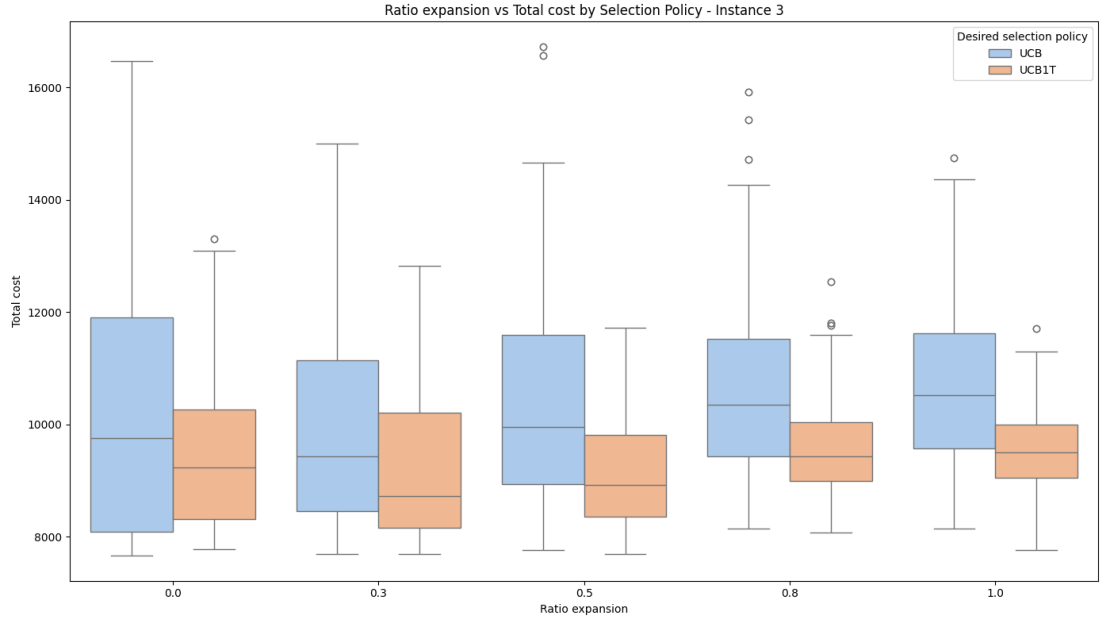


FIGURE 5.4: Expansion ratio vs Total cost

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

## Draf

Selec policy	Exp policy	Simu policy	N° chil- drens	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

- Background

Methodology Previous studies - Litterature gaps Just expand on the motivation I tried my mathematical formulation

Careful between I1 and I.1

Delete instances that are not studied

Computational time - good to report Try to do significant tests Add diminution of objective function plots

Submitted

Instutional for Electric

# Appendix A

## Code Listings

### A.1 Data preprocessing

---

```
import numpy as np
from copy import deepcopy

class data_preprocessing:
    def __init__(self, instance_path):
        self.instance_path = instance_path

        self.info, self.flights = self.read_file(f_name=self.instance_path)
        self.number_of_areas, self.starting_airport = (
            int(self.info[0][0]),
            self.info[0][1],
        )

        self.flights_by_day_dict =
self.flights_by_day(flight_list=self.flights)

        self.flights_by_day_dict = self.remove_duplicate(
            flights_by_day=self.flights_by_day_dict
        )

        self.list_days = [k for k in range(1, self.number_of_areas)]

        self.airports_by_area = self.get_airports_by_areas()
```

```

        self.area_to_explore = self.which_area_to_explore(
            airports_by_area=self.airports_by_area
        )
        self.area_by_airport =
self.invert_dict(original_dict=self.airports_by_area)

        self.starting_area = self.associated_area_to_airport(
            airport=self.starting_airport
        )
        self.list_airports = self.get_list_of_airports()
        self.list_areas = list(self.airports_by_area.keys())
        self.areas_connections_by_day = (
            self.possible_flights_from_zone_to_zone_specific_day()
        )

def read_file(self, f_name):
    dist = []
    line_nu = -1
    with open(f_name) as infile:
        for line in infile:
            line_nu += 1
            if line_nu == 0:
                index = int(line.split()[0]) * 2 + 1
            if line_nu >= index:
                temp = line.split()
                temp[2] = int(temp[2])
                temp[3] = int(temp[3])
                dist.append(temp)
            else:
                dist.append(line.split())
    info = dist[: int(dist[0][0]) * 2 + 1]
    flights = dist[int(dist[0][0]) * 2 + 1 :]
    return info, flights

def flights_by_day(self, flight_list):
    # Create an empty dictionary to hold flights organized by day
    flights_by_day = {}

    # Iterate over each flight in the input list
    for flight in flight_list:
        # Extract the day from the flight entry

```

```
        day = flight[2]

        # Create a flight entry without the day
        flight_without_day = flight[:2] + flight[3:]

        # Add the flight to the corresponding day in the dictionary
        if day not in flights_by_day:
            flights_by_day[day] = []
        flights_by_day[day].append(flight_without_day)

    return flights_by_day

def flights_from_airport(self, flights_by_day, from_airport,
    considered_day):
    flights_from_airport = []
    for day, flights in flights_by_day.items():
        if day == considered_day:
            for flight in flights:
                if flight[0] == from_airport:
                    flights_from_airport.append(flight)
            return flights_from_airport
        else:
            return None

def invert_dict(self, original_dict):
    inverted_dict = {}
    for key, value_list in original_dict.items():
        for value in value_list:
            if value in inverted_dict:
                inverted_dict[value].append(key)
            else:
                inverted_dict[value] = key
    return inverted_dict

def get_cost(self, day, from_airport, to_airport):
    # Retrieve flights for the specified day and day 0
    flights_day = self.flights_by_day_dict.get(day, [])
    flights_day_0 = self.flights_by_day_dict.get(0, [])

    # Find the cost for the specified day
    cost_day = next(
```

```

        (
            flight[2]
            for flight in flights_day
            if flight[0] == from_airport and flight[1] == to_airport
        ),
        float("inf"),
    )

    # Find the cost for day 0
    cost_day_0 = next(
        (
            flight[2]
            for flight in flights_day_0
            if flight[0] == from_airport and flight[1] == to_airport
        ),
        float("inf"),
    )

    # Return the minimum cost if either exists, otherwise inf
    if cost_day == float("inf") and cost_day_0 == float("inf"):
        return float("inf")

    return min(cost_day, cost_day_0)

def possible_flights_from_zone_to_zone_specific_day(self):
    areas_connections_by_day = {}

    for day, flights in self.flights_by_day_dict.items():
        areas_connections_list = []

        for flight in flights:
            connection = f"{self.area_by_airport.get(flight[0])} to {self.area_by_airport.get(flight[1])}"
            if connection not in areas_connections_list:
                areas_connections_list.append(connection)

        areas_connections_by_day[day] = areas_connections_list

    return areas_connections_by_day

def get_airports_by_areas(self):

```

---

```

        area_num = int(self.info[0][0])
        return {f"{i}": self.info[2 + i * 2] for i in range(0, area_num)}

def get_list_of_airports(self):
    unique_airports = set()

    # Iterate through each sublist and add elements to the set
    for sublist in self.airports_by_area.values():
        for airport in sublist:
            unique_airports.add(airport)

    return list(unique_airports)

def associated_area_to_airport(self, airport):
    return next(
        (
            area
            for area, airports in self.airports_by_area.items()
            if airport in airports
        ),
        "Airport not found",
    )

def remove_duplicate(self, flights_by_day):
    for day, flights in flights_by_day.items():
        unique_flights = {}
        for flight in flights:
            flight_key = (flight[0], flight[1])
            if flight_key not in unique_flights:
                unique_flights[flight_key] = flight
            else:
                if flight[2] < unique_flights[flight_key][2]:
                    #
print(flight[0],flight[1],flight[2],flight_key,unique_flights[flight_key][2])
                    unique_flights[flight_key] = flight
        flights_by_day[day] = list(unique_flights.values())
    return flights_by_day

def possible_flights_from_an_airport_at_a_specific_day(self, day,
from_airport):
    daily_flights = self.flights_by_day_dict.get(day, [])

```

---

```
        flights_from_airport = []
        for flight in daily_flights:
            if flight[0] == from_airport:

                flights_from_airport.append([flight[1], flight[2]])

        return flights_from_airport

def
possible_flights_from_an_airport_at_a_specific_day_with_previous_areas(
    self, day, from_airport, visited_areas
):
    daily_flights = self.flights_by_day_dict.get(
        day, []
    ) + self.flights_by_day_dict.get(0, [])
    flights_from_airport = []
    for flight in daily_flights:
        # print(self.associated_area_to_airport(airport=flight[0]))
        if (flight[0] == from_airport) and (
            self.associated_area_to_airport(airport=flight[1]) not in
visited_areas
        ):

            flights_from_airport.append([flight[1], flight[2]])

    return flights_from_airport

def which_area_to_explore(self, airports_by_area):
    return list(
        {
            key: len(value)
            for key, value in airports_by_area.items()
            if len(value) > 1
        }
    )
```

---



## A.2 Node

---

```
import numpy as np
import random
from scipy.stats import (
    kstest,
    norm,
    beta,
    expon,
    gamma,
    lognorm,
    weibull_min,
    uniform,
    pareto,
    t,
    chi2,
)

class Node:
    def __init__(self, state, desired_selection_policy, cp, parent=None):
        self.cp = cp
        self.desired_selection_policy = desired_selection_policy
        self.state = state # State is a dictionary representing the current situation
        self.parent = parent # Parent node
        self.children = [] # List of child nodes
        self.visit_count = 0 # Number of times this node has been visited
        self.total_cost = 0 # Total cost accumulated in simulations from this node
        self.scores = []

    def add_child(self, child_state):
        child_node = Node(
            state=child_state,
            desired_selection_policy=self.desired_selection_policy,
            cp=self.cp,
            parent=self,
        )
        self.children.append(child_node)
    return child_node
```

---

```

def is_fully_expanded(self):
    # if self.parent is None:
    #     return False
    return len(self.children) > 0 and all(
        child.visit_count > 0 for child in self.children
    )

def update(self, result):
    self.visit_count += 1
    self.total_cost += result
    self.scores.append(result)

def UCB(self, c_param):
    epsilon = 0

    visited_children = [child for child in self.children if (child.visit_count >
        0)]

    sorted_children = sorted(
        visited_children,
        key=lambda child: child.total_cost / (child.visit_count + epsilon),
    )
    scores = {child: rank + 1 for rank, child in enumerate(sorted_children)}
    total_scores = sum(scores.values())

    def normalized_score(child):
        return scores[child] / total_scores

    choices_weights = [
        normalized_score(child)
        + c_param
        * (2 * np.log(self.visit_count) / (child.visit_count + epsilon)) ** 0.5
        for child in visited_children
    ]

    best_child_node = self.children[np.argmin(choices_weights)]

    return best_child_node

def SP(self):

```

```
visited_children = [child for child in self.children if child.visit_count > 0]
D = 1

def sp_mcts_score(child):
    mean_cost = np.mean(child.scores) if len(child.scores) > 0 else 0
    variance = np.var(child.scores) if len(child.scores) > 0 else 0
    possible_deviation = np.sqrt(variance + (D / child.visit_count))
    return mean_cost - self.cp * possible_deviation

choices_weights = [sp_mcts_score(child) for child in visited_children]

best_child_node = self.children[np.argmin(choices_weights)]
return best_child_node

def Bayesian(self):
    visited_children = [child for child in self.children if child.visit_count > 0]
    N = self.visit_count

    def bayesian_uct_score(child, use_variance=False):
        mean_cost = np.mean(child.scores) if len(child.scores) > 0 else 0
        exploration_term = np.sqrt(2 * np.log(N) / child.visit_count)

        if use_variance:
            variance = np.sqrt(np.var(child.scores)) if len(child.scores) > 0 else 0
            exploration_term *= variance

        return mean_cost + exploration_term

    # Select which Bayesian UCT formula to use
    use_variance = True # Change this to 'False' to use the first formula
    choices_weights = [
        bayesian_uct_score(child, use_variance=use_variance)
        for child in visited_children
    ]

    best_child_node = self.children[np.argmin(choices_weights)]
    return best_child_node

def UCB1_tuned(self, c_param):
    visited_children = [child for child in self.children if child.visit_count > 0]
```

```
def ucb1_tuned_score(child):
    mean_cost = np.mean(child.scores) if len(self.scores) > 1 else 0
    variance = np.var(self.scores) if len(self.scores) > 1 else 0
    # UCB1-Tuned formula
    exploration_term = np.sqrt(
        (np.log(self.visit_count) / child.visit_count)
        * min(
            0.25,
            variance
            + np.sqrt(2 * np.log(self.visit_count) / child.visit_count),
        )
    )
    return mean_cost + c_param * exploration_term

choices_weights = [ucb1_tuned_score(child) for child in visited_children]

best_child_node = self.children[np.argmin(choices_weights)]
return best_child_node

def thompson_sampling(self, c_param):
    visited_children = [child for child in self.children if child.visit_count > 0]

def best_fit_distribution(scores):
    distributions = {
        "normal": norm,
        "beta": beta,
        "exponential": expon,
        "gamma": gamma,
        "lognormal": lognorm,
        "weibull_min": weibull_min,
        "uniform": uniform,
        "pareto": pareto,
        "t": t,
        "chi2": chi2,
    }
    p_values = {}
    for dist_name, dist in distributions.items():
        try:
            params = dist.fit(scores)
            d_statistic, p_value = kstest(scores, dist_name, args=params)
            p_values[dist_name] = p_value
```

---

```

except Exception as e:
    p_values[dist_name] = (
        0 # Handle the error and skip this distribution
    )
    print(f"Skipping {dist_name} due to fitting issues: {e}")

best_dist_name = max(p_values, key=p_values.get)
best_p_value = p_values[best_dist_name]

if best_p_value < 0.05:
    return None, None

best_dist = distributions[best_dist_name]
best_params = best_dist.fit(scores)

return best_dist, best_params

sampled_values = []
for child in visited_children:
    if len(child.scores) > 1:
        best_dist, best_params = best_fit_distribution(child.scores)
        if best_dist is not None:
            sampled_value = best_dist.rvs(*best_params)
            sampled_values.append(sampled_value)
        else:
            return self.UCB(
                c_param
            ) # Fallback to UCB if no good distribution is found
    else:
        sampled_values.append(np.mean(child.scores))

best_child_node = visited_children[np.argmin(sampled_values)]
return best_child_node

def randomized_ucb(self, c_param, random_factor=0.1):
    visited_children = [child for child in self.children if child.visit_count > 0]

    def randomized_ucb_score(child):
        mean_cost = np.mean(child.scores) if len(child.scores) > 0 else 0
        exploration_term = np.sqrt(
            (2 * np.log(self.visit_count) / (child.visit_count))

```

```

)
random_term = random_factor * np.random.rand()
return mean_cost + c_param * exploration_term + random_term

choices_weights = [randomized_ucb_score(child) for child in visited_children]

best_child_node = visited_children[np.argmin(choices_weights)]
return best_child_node

def epsilon_greedy(self, epsilon):
    visited_children = [child for child in self.children if child.visit_count > 0]

    if np.random.rand() < epsilon:
        # Explore: randomly select a child
        best_child_node = np.random.choice(visited_children)
    else:
        # Exploit: select the child with the best average cost
        best_child_node = min(
            visited_children,
            key=lambda child: (
                np.mean(child.scores) if len(child.scores) > 0 else float("inf")
            ),
        )

    return best_child_node

def best_child(self):
    if self.desired_selection_policy == "UCB":
        return self.UCB(c_param=self.cp)
    if self.desired_selection_policy == "UCB1T":
        return self.UCB1_tuned(c_param=self.cp)
    if self.desired_selection_policy == "SP":
        return self.epsilon_greedy(self.cp)
    if self.desired_selection_policy == "Bayesian":
        return self.Bayesian()

    else:
        raise ValueError(
            f"Unknown Selection policy: {self.desired_selection_policy}"
        )

```

---

```
def delete_node(self):  
    self.parent.children = [  
        child for child in self.parent.children if child != self  
    ]
```

---

### A.3 MCTS

---

```
import numpy as np
import random
from copy import deepcopy
import logging
import time
import os
import shutil
import glob

from Data_Preprocessing import data_preprocessing
from Node import Node

class MCTS(data_preprocessing):
    def __init__(
        self,
        instance,
        instance_number,
        number_childrens,
        desired_expansion_policy,
        ratio_expansion,
        desired_simulation_policy,
        desired_selection_policy,
        cp,
        number_simulation,
    ):
        self.instance_number = instance_number
        self.number_childrens = number_childrens
        self.desired_simulation_policy = desired_simulation_policy
        self.desired_expansion_policy = desired_expansion_policy
        self.ratio_expansion = ratio_expansion
        self.number_simulation = number_simulation
        self.desired_selection_policy = desired_selection_policy
        self.cp = cp

        self.expanded_nodes = []
        self.simulations_dict = {}

        self.start_time = time.time()
```



```

        super().__init__(instance_path=instance)
        self.end_time_data_preprocessing = time.time() - self.start_time
        self.simulation()
        # self.organise_log_files_in_folder(
        #     folder_path=os.path.dirname(self.instance_path)
        # )
        # self.collect_all_nodes()

def configure_logging(self):
    log_file =
f"{self.instance_path}_{self.number_childrens}_{self.desired_simulation_policy}_{self.desi
    log_file = self.get_unique_log_file(log_file)

    # Clear any existing handlers
    for handler in logging.root.handlers[:]:
        logging.root.removeHandler(handler)

    # Configure the logger
    logging.basicConfig(
        level=logging.DEBUG, # Set the log level to DEBUG to capture all
types of logs
        format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
        handlers=[
            logging.FileHandler(
                log_file, mode="w"
            ), # 'w' to overwrite the log file each run, 'a' to append
            # logging.StreamHandler(), # Optional: to also print logs to
the console
        ],
    )
    logger = logging.getLogger(__name__)
    return logger

def get_unique_log_file(self, base_log_file):
    """
    Check if the log file exists and if so, create a new file with a
unique suffix.
    """
    base_name, extension = os.path.splitext(base_log_file)
    counter = 0 # Start with 0 to have the first file as _0
    while True:

```

```
new_log_file = f"{base_name}_{counter}{extension}"
if not os.path.exists(new_log_file):
    return new_log_file
counter += 1

def organise_log_files_in_folder(self, folder_path):
    """
    Organize log files in the specified folder by moving files with the
    same base name into a dedicated directory.

    :param folder_path: Path to the folder containing the log files.
    """
    # Change to the target directory
    os.chdir(folder_path)

    # Find all log files in the directory
    log_files = glob.glob("*.log")

    # Track which files have already been moved to avoid duplication
    processed_bases = set()

    for log_file in log_files:
        # Extract the base name (up to the first '_')
        base_name = (
            log_file.rsplit("_", 1)[0]
            if "_" in log_file
            else log_file.rsplit(".", 1)[0]
        )

        if base_name not in processed_bases:
            # Mark this base as processed
            processed_bases.add(base_name)

            # Create a pattern to match all similar files
            pattern = f"{base_name}*.log"

            # Find all files matching this pattern
            matching_files = glob.glob(pattern)

            if matching_files:
                # Create a directory for these files
```

---

```

        folder_name = os.path.join(folder_path, base_name)
        os.makedirs(folder_name, exist_ok=True)

        # Move each matching file into the directory
        for file in matching_files:
            shutil.move(file, folder_name)

        # print(
        #     f"Moved files with base '{base_name}' into folder:
{folder_name}"
        # )

def initialise_root_node(self):
    return {
        "current_day": 1,
        "current_airport": self.starting_airport,
        "remaining_zones": [
            x for x in self.list_areas if x != self.starting_area
        ], # Exclude the starting area
        "visited_zones": [self.starting_area], # Exclude the starting area
        "total_cost": 0,
        "path": [self.starting_airport],
    }

def transition_function(self, state, action):
    new_state = deepcopy(state)
    new_state["current_day"] += 1
    new_state["current_airport"] = action[0]
    new_state["total_cost"] += action[1]
    new_state["path"].append(action[0])
    # self.logger.info(
    #     f"Airport {action[0]},
{self.associated_area_to_airport(airport=action[0])} to remove in
{new_state['remaining_zones']}"
    # )
    new_state["remaining_zones"].remove(
        self.associated_area_to_airport(airport=action[0])
    )
    new_state["visited_zones"].append(
        self.associated_area_to_airport(airport=action[0])
    )

```

```
        return new_state

def random_policy(self, actions):

    if not actions:
        return None
    return random.choice(actions)

def greedy_policy(self, actions):

    # self.logger.info(f"Actions: {actions}")
    if not actions:
        return None
    # Select the action with the lowest cost
    best_action = min(actions, key=lambda x: x[1])
    # self.logger.info(f"Chosen action based on heuristic policy:
{best_action}")
    return best_action

def tolerance_heuristic_policy(self, actions):
    # self.logger.info(f"Actions: {actions}")

    if not actions:
        return None

    # Find the minimum cost
    min_cost = min(actions, key=lambda x: x[1])[1]

    # Filter actions within the tolerance level
    best_actions = [
        action
        for action in actions
        if action[1] <= min_cost * (1 + self.ratio_expansion)
    ]

    # Select a random action from the best actions
    best_action = random.choice(best_actions)

    # self.logger.info(f"Chosen action based on tolerance policy:
{best_action}")
```

```
        return best_action

def get_unvisited_children(self, node):
    queue = [node]
    unvisited_children = []
    while queue:
        current_node = queue.pop(0)
        for child in current_node.children:
            if child.visit_count == 0:
                unvisited_children.append(child)
            else:
                queue.append(child)

    return unvisited_children

def backpropagate(self, node, cost):
    while node is not None:

        node.update(cost)

        # self.logger.info(
        #     f"Backpropagating Node: {node.state}, Visit Count:
{node.visit_count}, Total Cost: {node.total_cost}, Scores: {node.scores}"
        # )

        node = node.parent

def collect_all_nodes(self):
    nodes = []
    queue = [self.root]
    while queue:
        node = queue.pop(0)
        nodes.append(node)
        queue.extend(node.children)
    return nodes

def get_final_nodes(self):
    day = self.number_of_areas + 1
    nodes = [
        node
        for node in self.collect_all_nodes()
```

```

        if node.state.get("current_day") == day
    ]

    # Initialize variables to track the best nodes for this day
    min_cost_child = None
    robust_child = None
    min_cost_robust_child = None
    secure_child = None

    # Values to compare against
    min_cost = float("inf")
    max_visit_count = -float("inf")
    max_secure_value = -float("inf")

    for node in nodes:
        # Min-Cost Child: Select the root child with the lowest total_cost
        if node.total_cost < min_cost:
            min_cost = node.total_cost
            min_cost_child = node

        # Robust Child: Select the most visited root child (visit_count)
        if node.visit_count > max_visit_count:
            max_visit_count = node.visit_count
            robust_child = node

        # Min-Cost-Robust Child: Among the nodes with the lowest
        total_cost, select the one with the highest visit_count
        if node.total_cost == min_cost and node.visit_count >=
max_visit_count:
            min_cost_robust_child = node

        # Secure Child: Select the child that minimizes a lower confidence
bound
        if (
            node.visit_count > 0
            and node.parent is not None
            and node.parent.visit_count > 0
        ):
            secure_value = (node.total_cost / node.visit_count) - self.cp
* (
            (node.parent.visit_count / node.visit_count) ** 0.5

```

```

        )
        if secure_value > max_secure_value:
            max_secure_value = secure_value
            secure_child = node

    # Logging the results for the current day
    if min_cost_child:
        self.logger.info("\n\n")
        self.logger.info(f"Best Node: {min_cost_child.state}")
    if robust_child:
        self.logger.info(
            f"Robust Child (Day {day}): State={robust_child.state}, Visit
Count={max_visit_count}"
        )
    if min_cost_robust_child:
        self.logger.info(
            f"Min-Cost-Robust Child (Day {day}):
State={min_cost_robust_child.state}, Cost={min_cost}, Visit
Count={min_cost_robust_child.visit_count}"
        )
    if secure_child:
        self.logger.info(
            f"Secure Child (Day {day}): State={secure_child.state}, Secure
Value={max_secure_value}"
        )

def display_all_nodes(self, nodes):
    for node in nodes:
        print(
            f"State: {node.state}, Visit Count: {node.visit_count}, Total
Cost: {node.total_cost}"
        )
        self.logger.info(
            f"State: {node.state}, Visit Count: {node.visit_count}, Total
Cost: {node.total_cost}"
        )

def print_execution_times(self):
    self.logger.info(
        f"\n\n\n Time to preprocess the data:
{self.end_time_data_preprocessing:.4f} seconds"
    )

```

```

    )
    self.logger.info(
        f"\n\n\n Time to find the solution: {self.end_search_time:.4f}
seconds"
    )
    self.logger.info(
        f"\n\n\n Total time:
{self.end_time_data_preprocessing+self.end_search_time:.4f} seconds \n\n"
    )

def get_simulation_policy(self):
    if self.desired_simulation_policy == "greedy_policy":
        return self.greedy_policy
    elif self.desired_simulation_policy == "random_policy":
        return self.random_policy
    elif self.desired_simulation_policy == "tolerance_policy":
        return self.tolerance_heuristic_policy
    else:
        raise ValueError(
            f"Unknown simulation policy: {self.desired_simulation_policy}"
        )

def get_expansion_policy(self):
    if self.desired_expansion_policy == "top_k":
        return self.top_k_actions

    if self.desired_expansion_policy == "ratio_k":
        return self.ratio_best_random

    else:
        raise ValueError(
            f"Unknown expansion policy: {self.desired_expansion_policy}"
        )

def top_k_actions(self, actions):
    sorted_actions = sorted(actions, key=lambda x: x[1])
    return sorted_actions[: self.number_childrens]

def ratio_best_random(self, actions):
    # Determine the number of best actions to take based on the ratio
    ratio = self.ratio_expansion

```



---

```

num_best = int(self.number_childrens * ratio)
num_random = self.number_childrens - num_best

# Sort actions to get the best ones
sorted_actions = sorted(actions, key=lambda x: x[1])
best_actions = sorted_actions[:num_best]

# Select the remaining random actions from the remaining pool
remaining_actions = sorted_actions[num_best:]

# Ensure we don't try to sample more than available actions
num_random = min(num_random, len(remaining_actions))

# If num_random is zero or there are no remaining actions, we skip the
sampling
if num_random > 0 and remaining_actions:
    random_actions = random.sample(remaining_actions, num_random)
else:
    random_actions = []

# Combine the best actions and the random actions
final_actions = best_actions + random_actions
random.shuffle(final_actions)

return final_actions

def delete_node(self, node):
    if node.parent:
        for _ in node.parent.children:
            pass
            # self.logger.info(
            #     f"before deletion: {len(node.parent.children)}, {_.state}"
            # )
        node.parent.children.remove(node)
        for _ in node.parent.children:
            pass
            # self.logger.info(
            #     f"after deletion: {len(node.parent.children)}, {_.state}"
            # )

def print_characteristics_simulation(self):

```

```

        self.logger.info(f"\n\nSimulation dictionary:
{self.simulations_dict}")
        self.logger.info(f"Number of childrens: {self.number_childrens}")
        self.logger.info(f"Desired expansion policy:
{self.desired_expansion_policy}")
        self.logger.info(f"Ratio expansion: {self.ratio_expansion}")
        self.logger.info(f"Desired simulation policy:
{self.desired_simulation_policy}")
        self.logger.info(f"Desired selection policy:
{self.desired_selection_policy}")
        self.logger.info(f"Cp: {self.cp}")
        self.logger.info(f"Instance: {self.instance_number}")

def simulation(self):
    for _ in range(self.number_simulation):
        self.logger = None
        self.logger = self.configure_logging()
        self.root = Node(
            self.initialise_root_node(),
            desired_selection_policy=self.desired_selection_policy,
            cp=self.cp,
        )
        self.best_leaf = None
        self.best_leaf_cost = float("inf")
        self.search()
        self.end_search_time = time.time() - self.start_time
        self.print_execution_times()
        self.get_final_nodes()
        self.print_characteristics_simulation()

def select(self, node):
    self.logger.info("\nSELECTION\n")
    current_node = node
    self.logger.info(f"Starting selection at node: {current_node.state}")

    while current_node.children:
        self.logger.info(f"Current node: {current_node.state}")
        self.logger.info(f"Childrens: {current_node.children}")

        if not current_node.is_fully_expanded():
            # Select a random unvisited child if there are any

```

---

```

        unvisited_children = [
            child for child in current_node.children if
child.visit_count == 0
        ]
        self.logger.info(f"Unvisited children:
{len(unvisited_children)}")
        if unvisited_children:
            selected_child = random.choice(unvisited_children)
            self.logger.info(
                f"Randomly selected unvisited child: {selected_child}"
            )
            return True, selected_child

    else:
        current_node = current_node.best_child()
        self.logger.info(f"Moving to best child: {current_node.state}")
        # return True, current_node

    if (not current_node.children) and (
        current_node.state["current_day"] == self.number_of_areas
    ):
        self.logger.info("Final day selected")
        return False, current_node

    elif (not current_node.children) and (
        current_node.state["current_day"] != self.number_of_areas
    ):
        self.logger.info(f"The node {current_node.state} has no children")
        return False, current_node

    elif current_node.state["current_day"] == self.number_of_areas + 1:
        return True, current_node

def expand_node(self, node):
    if node not in self.expanded_nodes:
        self.expanded_nodes.append(node)

    actions =
self.possible_flights_from_an_airport_at_a_specific_day_with_previous_areas(
    node.state["current_day"],
    node.state["current_airport"],

```

```

        node.state["visited_zones"],
    )

    if node.state["current_day"] == self.number_of_areas:
        node.state["visited_zones"] = node.state["visited_zones"][1:]
        node.state["remaining_zones"].append(
            self.associated_area_to_airport(self.starting_airport)
        )

        actions =
self.possible_flights_from_an_airport_at_a_specific_day_with_previous_areas(
            node.state["current_day"],
            node.state["current_airport"],
            node.state["visited_zones"],
        )

    expansion_policy = self.get_expansion_policy()
    actions = expansion_policy(actions)

    if actions:
        self.logger.info("Start expansion")
        for action in actions:
            self.logger.info(f"{action}")
            new_state = self.transition_function(node.state, action)
            node.add_child(new_state)
        self.logger.info("End expansion")
    else:
        self.logger.info(f"No actions possible")
        return None

    return node

else:
    self.logger.info("INFINITE LOOP")
    return None

def search(self):
    while True:
        node_to_explore = self.select(self.root)

        self.logger.info(f"Node to explore: {node_to_explore[1].state}")

```

```

        if node_to_explore[1].state["current_day"] == self.number_of_areas
+ 1:
            while not node_to_explore[1].parent.is_fully_expanded():
                # self.logger.info(
                #     "Node to explore is last day but all siblings have
not been visited yet"
                # )
                node_to_explore = self.select(self.root)
                self.logger.info(f"Node to explore:
{node_to_explore[1].state}")
                result = node_to_explore[1].state["total_cost"]
                self.backpropagate(node_to_explore[1], result)

            node_to_explore[1].state["visited_zones"].append(
                self.associated_area_to_airport(
                    airport=node_to_explore[1].state["path"][-1]
                )
            )
        return

    if not node_to_explore[0]:
        expanded_node = self.expand_node(node=node_to_explore[1])
        if not expanded_node:
            self.logger.info("Not unexpandable so deleted")
            node_to_explore[1].delete_node()
            # self.logger.info(f"Nodes in tree:
{len(self.collect_all_nodes())}")
            if len(self.collect_all_nodes()) == 1:
                self.logger.info("Everything has been deleted to the
root node")

                self.end_time_data_preprocessing = 0
                self.end_search_time = 0
                self.print_characteristics_simulation()
                self.print_execution_times()
                break
            continue
        else:
            self.logger.info(
                f"{node_to_explore[1].state} has been successfully
expanded"

```

```

        )
        continue

    else:
        simulation = self.simulate(node_to_explore[1])

        if simulation[0]:
            self.logger.info(f"Result from simulation:
{simulation[0]}")

            key = str(node_to_explore[1].state["current_day"])
            value_to_add = simulation[0]
            if key in self.simulations_dict:
                self.simulations_dict[key].append(value_to_add)
            else:
                self.simulations_dict[key] = [value_to_add]

            self.backpropagate(node_to_explore[1], simulation[0])

        else:
            self.logger.info(
                "Simulation failed to reach a valuable state - node
deleted"
            )
            self.delete_node(node_to_explore[1])
            if len(self.collect_all_nodes()) == 1:
                self.logger.info("Everything has been deleted to the
root node")

                self.end_time_data_preprocessing = 0
                self.end_search_time = 0
                self.print_characteristics_simulation()
                self.print_execution_times()
                break

    def simulate(self, node):
        self.logger.info("\n\nSIMULATION")
        simulation_policy = self.get_simulation_policy()
        current_simulation_state = deepcopy(node.state)
        self.logger.info(f"Selected node for simulation
{current_simulation_state}")

```

---

```

        while current_simulation_state["current_day"] != self.number_of_areas:
            actions =
self.possible_flights_from_an_airport_at_a_specific_day_with_previous_areas(
                day=current_simulation_state["current_day"],
                from_airport=current_simulation_state["current_airport"],
                visited_areas=current_simulation_state["visited_zones"],
            )

            action = simulation_policy(actions=actions)
            # self.logger.info(f"Action: {action}")
            if action is None:
                self.logger.info("Action is None")
                return False, False

            current_simulation_state = self.transition_function(
                current_simulation_state, action
            )
            # self.logger.info(f"Current simulation state
{current_simulation_state}")

            if current_simulation_state["current_day"] == self.number_of_areas:
                current_simulation_state["visited_zones"] =
current_simulation_state[
                    "visited_zones"
                ][1:]
                current_simulation_state["remaining_zones"].append(
                    self.associated_area_to_airport(self.starting_airport)
                )

                actions =
self.possible_flights_from_an_airport_at_a_specific_day_with_previous_areas(
                    day=current_simulation_state["current_day"],
                    from_airport=current_simulation_state["current_airport"],
                    visited_areas=current_simulation_state["visited_zones"],
                )

            if not actions:
                self.logger.info("No flight available to go back to the
initial area")
                return False, False
            else:

```

---

```
        action = simulation_policy(actions=actions)

        current_simulation_state = self.transition_function(
            current_simulation_state, action
        )
        self.logger.info(f"Current simulation state
{current_simulation_state}")

        return current_simulation_state["total_cost"],
current_simulation_state
```

---



## Appendix B

### Test Instances

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

# Appendix C

## Simulations results

### C.1 Instance 1

#### C.1.1 Solution found

Selec policy	Exp policy	Simu policy	N° chil- drens	Ratio	Cp	Best cost	Mean	Std	T(s)
UCB	ratio k	greedy	5	.3	2.8	1396	1396.00		.084
UCB	top k	greedy	5	.5	1.4	1396	1396.00		.085
UCB	top k	greedy	5	.3	1.4	1396	1396.00		.085
UCB	top k	greedy	10	.8	1.4	1396	1396.00		.096
UCB	top k	greedy	10	.3	1.4	1396	1396.00		.097
UCB	top k	greedy	5	.3	2.8	1396	1396.00		.097
UCB	top k	greedy	5	1	1.4	1396	1396.00		.097
UCB	top k	greedy	5	.8	2.8	1396	1396.00		.098
UCB	ratio k	greedy	10	1	2.8	1396	1396.00		.098
UCB	top k	greedy	5	0	2.8	1396	1396.00		.099
UCB	ratio k	greedy	5	0	2.8	1396	1396.00		.100
UCB	ratio k	greedy	5	1	1.4	1396	1396.00		.101
UCB	top k	greedy	5	.5	2.8	1396	1396.00		.101
UCB	ratio k	greedy	10	.3	2.8	1396	1396.00		.102
UCB	top k	greedy	10	0	1.4	1396	1396.00		.103
UCB	top k	greedy	15	.3	1.4	1396	1396.00		.107
UCB	top k	greedy	5	0	1.4	1396	1396.00		.107
UCB	ratio k	greedy	10	.5	2.8	1396	1396.00		.112
UCB	ratio k	greedy	15	.8	1.4	1396	1396.00		.112

UCB	top k	greedy	15	.8	1.4	1396	1396.00		.115
UCB	top k	greedy	15	1	1.4	1396	1396.00		.115
UCB	ratio k	greedy	10	0	2.8	1396	1396.00		.116
UCB	top k	greedy	10	1	1.4	1396	1396.00		.116
UCB	ratio k	greedy	10	.3	1.4	1396	1396.00		.117
UCB	top k	greedy	5	1	2.8	1396	1396.00		.117
UCB	top k	greedy	15	.3	2.8	1396	1396.00		.118
UCB	top k	greedy	10	.5	1.4	1396	1396.00		.118
UCB	top k	greedy	5	.8	1.4	1396	1396.00		.118
UCB	ratio k	greedy	15	.3	2.8	1396	1396.00		.119
UCB	ratio k	greedy	15	.8	2.8	1396	1396.00		.119
UCB	top k	greedy	15	.8	2.8	1396	1396.00		.120
UCB	ratio k	greedy	10	.5	1.4	1396	1396.00		.120
UCB	ratio k	tolerance	10	0	2.8	1396	1396.00	0.00	.120
UCB	ratio k	greedy	10	.8	2.8	1396	1396.00		.122
UCB	top k	greedy	15	0	2.8	1396	1396.00		.122
UCB	top k	tolerance	5	0	2.8	1396	1396.00	0.00	.126
UCB	top k	greedy	15	.5	1.4	1396	1396.00		.126
UCB	ratio k	greedy	10	0	1.4	1396	1396.00		.126
UCB	top k	greedy	10	.8	2.8	1396	1396.00		.127
UCB	ratio k	tolerance	15	0	2.8	1396	1396.00	0.00	.128
UCB	ratio k	greedy	15	1	2.8	1396	1396.00		.129
UCB	top k	greedy	10	.3	2.8	1396	1396.00		.131
UCB	ratio k	greedy	5	1	2.8	1396	1396.00		.131
UCB	ratio k	greedy	15	0	2.8	1396	1396.00		.132
UCB	top k	greedy	10	0	2.8	1396	1396.00		.132
UCB	ratio k	greedy	15	.3	1.4	1396	1396.00		.133
UCB	ratio k	greedy	15	.5	1.4	1396	1396.00		.133
UCB	top k	greedy	15	.5	2.8	1396	1396.00		.134
UCB	ratio k	greedy	10	1	1.4	1396	1396.00		.136
UCB	ratio k	greedy	15	1	1.4	1396	1396.00		.137
UCB	ratio k	tolerance	5	0	1.4	1396	1518.60	99.08	.139
UCB	top k	greedy	15	1	2.8	1396	1396.00		.142
UCB	top k	greedy	10	1	2.8	1396	1396.00		.143
UCB	ratio k	tolerance	15	0	1.4	1396	1396.00	0.00	.143
UCB	top k	greedy	15	0	1.4	1396	1396.00		.143
UCB	ratio k	greedy	15	.5	2.8	1396	1396.00		.147
UCB	ratio k	greedy	15	0	1.4	1396	1396.00		.148
UCB	ratio k	tolerance	10	0	1.4	1396	1396.00	0.00	.152
UCB	top k	tolerance	15	0	1.4	1396	1396.00	0.00	.153
UCB	top k	tolerance	10	0	1.4	1396	1396.00	0.00	.155

UCB	top k	greedy	10	.5	2.8	1396	1396.00		.157
UCB	top k	tolerance	15	.3	2.8	1396	1524.20	72.14	.157
UCB	top k	tolerance	5	0	1.4	1396	1396.00	0.00	.158
UCB	top k	tolerance	15	.8	1.4	1396	1654.70	185.65	.161
UCB	top k	tolerance	10	0	2.8	1396	1396.00	0.00	.174
UCB	top k	tolerance	15	0	2.8	1396	1396.00	0.00	.177
UCB	ratio k	greedy	10	.8	1.4	1396	1396.00		.178
UCB	ratio k	tolerance	15	.5	1.4	1396	1599.20	89.85	.385
UCB	top k	tolerance	5	.5	2.8	1396	1588.50	109.62	.394
UCB	ratio k	tolerance	10	1	1.4	1396	1572.20	148.00	.488
UCB	top k	tolerance	15	.8	2.8	1396	1647.80	209.03	.645
UCB	ratio k	tolerance	5	0	2.8	1396	1509.00	81.71	.659
UCB	top k	tolerance	10	1	1.4	1396	1617.70	183.61	.794
UCB	top k	tolerance	15	1	2.8	1396	1589.50	130.85	.809
UCB	top k	tolerance	15	.5	2.8	1396	1528.40	109.46	.837
UCB	ratio k	tolerance	15	1	1.4	1396	1606.10	125.35	.864
UCB	top k	tolerance	5	.3	2.8	1396	1528.60	76.94	.961
UCB	top k	tolerance	10	.3	1.4	1396	1528.90	109.87	1.060
UCB	ratio k	tolerance	5	.8	1.4	1396	1574.70	123.89	1.208
UCB	top k	tolerance	15	1	1.4	1396	1592.50	143.08	1.613
UCB	ratio k	random	10	.8	0	1407	3549.90	1959.51	2.745
UCB	top k	tolerance	5	.3	1.4	1431	1532.50	112.31	.514
UCB	top k	tolerance	5	.8	1.4	1431	1618.70	97.18	.806
UCB	ratio k	tolerance	10	.8	1.4	1431	1583.10	123.40	.830
UCB	ratio k	tolerance	10	.5	2.8	1431	1549.10	96.27	1.021
UCB	ratio k	tolerance	15	1	2.8	1431	1615.40	179.82	1.432
UCB	ratio k	tolerance	10	.3	2.8	1457	1508.70	40.06	.138
UCB	top k	tolerance	5	1	1.4	1457	1543.60	84.48	1.857
UCB	ratio k	greedy	5	.5	2.8	1458	1458.00		.113
UCB	ratio k	greedy	5	0	1.4	1458	1458.00		.115
UCB	top k	tolerance	5	1	2.8	1458	1563.00	88.51	.126
UCB	top k	tolerance	10	.8	2.8	1458	1640.50	101.40	.348
UCB	ratio k	tolerance	15	.8	2.8	1458	1575.60	102.64	.381
UCB	top k	tolerance	10	.8	1.4	1458	1571.40	123.30	.382
UCB	ratio k	random	15	.8	2.8	1458	4879.30	2587.48	.591
UCB	ratio k	tolerance	5	.8	2.8	1458	1586.00	106.15	.806
UCB	top k	tolerance	5	.5	1.4	1458	1541.30	45.20	.901
UCB	ratio k	tolerance	15	.3	2.8	1458	1502.60	63.95	1.081
UCB	ratio k	tolerance	10	.5	1.4	1458	1523.70	46.63	1.161
UCB	ratio k	random	10	1	1.4	1458	5975.30	4237.38	1.756
UCB1T	ratio k	greedy	10	.5	1.4	1472	1472.00		.893

UCB	top k	tolerance	10	.8	0	1472	1903.50	169.28	1.057
UCB	ratio k	tolerance	10	1	2.8	1472	1661.30	160.90	1.267
UCB	ratio k	greedy	5	1	0	1472	1472.00		1.801
UCB1T	ratio k	tolerance	15	.3	1.4	1472	1818.00	150.91	5.009
UCB	top k	tolerance	15	.5	0	1472	1808.00	146.75	5.484
UCB1T	top k	tolerance	5	0	2.8	1472	1803.50	208.94	6.320
UCB	ratio k	tolerance	5	1	0	1472	1799.70	161.43	6.925
UCB1T	ratio k	tolerance	15	1	0	1472	1870.00	220.25	19.040
UCB1T	ratio k	tolerance	5	1	2.8	1472	1895.10	211.12	28.132
UCB	top k	tolerance	10	.3	2.8	1479	1520.70	70.89	.160
UCB	ratio k	tolerance	5	.3	2.8	1479	1523.20	81.22	.216
UCB	ratio k	tolerance	5	.3	1.4	1479	1550.20	92.04	.440
UCB	top k	tolerance	5	.8	2.8	1479	1643.70	125.43	.500
UCB	ratio k	tolerance	10	.3	1.4	1479	1560.20	65.68	.870
UCB	top k	tolerance	15	.3	1.4	1479	1561.00	73.15	1.526
UCB	ratio k	greedy	5	.3	1.4	1481	1481.00		.095
UCB	ratio k	greedy	5	.8	2.8	1481	1481.00		.104
UCB	ratio k	greedy	5	.8	1.4	1481	1481.00		.115
UCB	ratio k	greedy	5	.5	1.4	1481	1481.00		.117
UCB	top k	tolerance	15	.5	1.4	1481	1566.80	83.76	.738
UCB	ratio k	tolerance	15	.3	1.4	1481	1607.00	95.21	1.236
UCB1T	ratio k	tolerance	5	.8	0	1481	1847.50	208.87	13.143
UCB	ratio k	tolerance	5	.5	2.8	1485	1559.70	90.12	.644
UCB	ratio k	tolerance	5	1	1.4	1489	1649.10	60.98	.126
UCB	ratio k	tolerance	5	.5	1.4	1490	1555.70	56.12	.106
UCB1T	top k	tolerance	15	.8	0	1490	1865.60	158.48	5.096
UCB1T	top k	random	5	.5	1.4	1493	2407.10	1045.81	3.136
UCB	ratio k	tolerance	15	.5	2.8	1495	1551.60	37.38	.316
UCB	top k	tolerance	10	.5	2.8	1495	1608.60	78.12	1.129
UCB	ratio k	random	5	1	1.4	1506	3187.40	1785.08	.179
UCB	ratio k	random	5	1	2.8	1506	4330.10	2775.69	.492
UCB	ratio k	tolerance	15	.3	0	1506	1745.30	193.93	1.829
UCB1T	top k	random	15	.3	0	1506	2634.80	1495.29	2.654
UCB	ratio k	tolerance	15	.8	1.4	1521	1664.60	140.50	.160
UCB	top k	random	5	0	1.4	1522	3817.40	2271.90	1.650
UCB1T	top k	tolerance	5	.3	1.4	1522	1803.10	135.11	44.419
UCB	ratio k	tolerance	5	1	2.8	1526	1636.10	95.27	.322
UCB	ratio k	tolerance	10	.8	2.8	1526	1658.90	95.49	.654
UCB1T	ratio k	greedy	5	0	1.4	1529	1529.00		.512
UCB1T	ratio k	random	5	.5	1.4	1529	2613.00	1381.25	.883
UCB	top k	tolerance	15	.3	0	1529	1816.10	204.92	1.285

UCB1T	ratio k	tolerance	5	0	0	1529	1889.20	191.58	2.025
UCB1T	ratio k	random	10	.3	1.4	1529	2922.40	1754.56	2.695
UCB1T	top k	tolerance	15	0	0	1529	1884.20	166.62	2.890
UCB1T	ratio k	tolerance	10	1	0	1529	1827.00	215.19	3.312
UCB1T	top k	tolerance	15	.3	2.8	1529	1890.20	168.87	4.693
UCB1T	ratio k	random	15	.8	1.4	1529	3993.00	2298.81	5.162
UCB1T	top k	tolerance	10	.5	0	1529	1823.10	211.45	6.695
UCB1T	ratio k	tolerance	10	.5	1.4	1529	1850.50	224.25	8.508
UCB1T	ratio k	tolerance	10	.3	1.4	1529	1796.60	167.63	9.516
UCB1T	top k	tolerance	15	1	0	1529	1831.70	133.56	11.001
UCB1T	ratio k	tolerance	5	.8	1.4	1529	1798.40	216.53	16.114
UCB	top k	tolerance	10	.5	1.4	1530	1609.90	76.29	.924
UCB1T	ratio k	random	5	0	2.8	1533	3012.00	1836.74	3.879
UCB1T	top k	tolerance	5	1	0	1533	1882.40	178.26	8.601
UCB1T	ratio k	tolerance	5	1	1.4	1533	1809.70	202.15	9.562
UCB1T	ratio k	tolerance	10	.8	0	1533	1838.20	145.60	9.573
UCB1T	top k	tolerance	10	1	0	1533	1834.90	172.42	17.707
UCB1T	ratio k	greedy	10	.8	0	1540	1540.00		.666
UCB1T	ratio k	greedy	10	1	0	1540	1540.00		.879
UCB1T	top k	random	15	.5	2.8	1540	3122.70	1753.55	1.088
UCB1T	top k	greedy	15	.5	2.8	1540	1540.00		1.181
UCB1T	ratio k	tolerance	5	.3	0	1540	1864.60	175.12	1.319
UCB1T	top k	greedy	15	.8	2.8	1540	1540.00		1.664
UCB	top k	greedy	5	.5	0	1540	1540.00		1.694
UCB	top k	greedy	5	0	0	1540	1540.00		1.702
UCB1T	ratio k	tolerance	10	.8	1.4	1540	1845.80	162.15	2.461
UCB1T	top k	greedy	5	.3	1.4	1540	1540.00		2.500
UCB	top k	tolerance	5	.5	0	1540	1800.30	156.39	2.706
UCB1T	ratio k	tolerance	10	.3	0	1540	1831.40	174.19	2.958
UCB1T	top k	tolerance	15	.3	0	1540	1896.80	214.92	3.424
UCB	ratio k	tolerance	10	0	0	1540	1850.20	183.61	4.229
UCB1T	ratio k	tolerance	5	.3	2.8	1540	1919.00	194.58	5.013
UCB1T	top k	tolerance	15	0	2.8	1540	1865.30	202.86	5.138
UCB1T	top k	tolerance	10	.5	2.8	1540	1776.30	180.87	6.638
UCB	ratio k	tolerance	15	0	0	1540	1913.30	207.39	7.511
UCB1T	top k	tolerance	10	.5	1.4	1540	1885.00	235.90	7.624
UCB	top k	tolerance	15	.8	0	1540	1810.70	162.48	7.951
UCB	top k	tolerance	10	0	0	1540	1882.00	169.58	8.397
UCB1T	ratio k	tolerance	10	.3	2.8	1540	1786.20	149.51	9.080
UCB1T	ratio k	tolerance	15	.8	2.8	1540	1758.30	183.71	9.664
UCB1T	top k	tolerance	15	1	2.8	1540	1814.90	116.59	9.710

UCB1T	ratio k	tolerance	15	1	2.8	1540	1862.50	184.05	9.835
UCB1T	ratio k	tolerance	15	.3	0	1540	1809.80	164.19	10.175
UCB1T	ratio k	tolerance	10	1	2.8	1540	1897.00	281.85	10.565
UCB1T	ratio k	tolerance	15	.8	1.4	1540	1908.50	196.92	13.413
UCB1T	ratio k	tolerance	15	.5	1.4	1540	1868.40	160.08	15.130
UCB1T	top k	tolerance	5	.8	0	1540	1824.20	165.59	16.175
UCB1T	top k	tolerance	5	0	1.4	1540	1806.50	198.51	17.011
UCB1T	top k	tolerance	5	.3	0	1540	1870.70	196.15	22.181
UCB1T	top k	tolerance	5	0	0	1540	1732.90	156.80	35.029
UCB1T	top k	tolerance	5	.5	2.8	1540	1828.10	128.77	53.816
UCB	ratio k	random	15	.8	0	1544	3538.60	1864.08	1.304
UCB1T	top k	random	5	0	2.8	1544	2510.60	969.95	1.983
UCB1T	top k	tolerance	15	.3	1.4	1546	1832.00	184.84	3.920
UCB	top k	tolerance	10	1	2.8	1547	1639.00	101.50	.350
UCB	top k	random	15	.3	2.8	1548	7304.10	5361.45	1.066
UCB1T	top k	tolerance	15	.5	0	1548	1838.50	129.02	5.476
UCB1T	top k	tolerance	10	.3	0	1548	1959.70	210.51	20.794
UCB1T	ratio k	random	15	0	0	1551	2592.00	1259.81	2.241
UCB1T	top k	tolerance	15	.8	2.8	1551	1862.00	139.22	12.043
UCB1T	ratio k	tolerance	5	1	0	1551	1884.40	202.74	22.815
UCB	ratio k	tolerance	5	.5	0	1552	1861.00	177.52	1.938
UCB	ratio k	tolerance	5	.8	0	1552	1818.80	158.49	2.109
UCB1T	top k	tolerance	10	1	2.8	1552	1826.90	170.80	7.951
UCB1T	ratio k	tolerance	5	.5	0	1553	1842.90	145.37	1.425
UCB1T	ratio k	tolerance	5	.5	2.8	1553	1820.10	155.51	3.897
UCB1T	top k	random	10	0	2.8	1553	3300.10	1765.23	3.970
UCB	ratio k	tolerance	10	.8	0	1553	1865.80	179.46	5.543
UCB1T	ratio k	tolerance	15	0	2.8	1553	1842.80	230.98	5.783
UCB1T	ratio k	tolerance	10	.5	0	1553	1832.90	113.14	9.797
UCB1T	ratio k	tolerance	15	.5	0	1553	1853.50	165.94	12.827
UCB1T	top k	random	5	.5	0	1555	2709.90	1386.28	.908
UCB1T	top k	random	15	.3	1.4	1555	2758.60	1549.45	5.773
UCB1T	top k	tolerance	10	.8	0	1561	1842.40	185.52	4.651
UCB1T	ratio k	tolerance	15	.5	2.8	1561	1886.10	209.51	8.565
UCB	ratio k	tolerance	10	1	0	1564	1792.40	163.67	.729
UCB	top k	greedy	10	.5	0	1564	1564.00		.746
UCB1T	ratio k	greedy	10	1	1.4	1564	1564.00		.967
UCB1T	ratio k	greedy	15	.3	1.4	1564	1564.00		1.123
UCB1T	top k	tolerance	10	.3	2.8	1564	1848.00	154.42	1.583
UCB1T	top k	tolerance	10	0	2.8	1564	1876.20	146.84	2.413
UCB	ratio k	tolerance	10	.3	0	1564	1894.60	168.37	3.180

UCB1T	ratio k	tolerance	15	0	1.4	1564	1926.10	169.41	3.914
UCB1T	ratio k	tolerance	10	0	0	1564	1894.10	209.84	5.046
UCB	top k	tolerance	5	.3	0	1564	1802.20	110.06	5.248
UCB1T	ratio k	tolerance	10	.5	2.8	1564	1903.40	202.94	5.620
UCB	top k	tolerance	15	0	0	1564	1996.80	188.44	7.431
UCB1T	top k	tolerance	15	.5	1.4	1564	1801.10	181.31	8.398
UCB	top k	tolerance	5	.8	0	1564	1821.10	171.24	8.987
UCB1T	top k	tolerance	15	1	1.4	1564	1857.30	200.71	10.041
UCB1T	ratio k	tolerance	15	1	1.4	1564	1931.50	183.48	12.772
UCB1T	top k	tolerance	5	.5	1.4	1564	1891.70	149.04	31.446
UCB	ratio k	random	10	.3	2.8	1565	5063.80	4094.92	.375
UCB1T	top k	random	10	.3	0	1565	3329.80	2124.79	2.699
UCB1T	top k	random	15	1	0	1565	3236.20	2047.21	5.953
UCB1T	top k	random	10	1	2.8	1569	2779.10	1889.48	1.492
UCB	ratio k	random	15	.3	2.8	1577	6779.70	3457.07	1.545
UCB1T	top k	tolerance	10	0	0	1577	1873.80	178.43	2.373
UCB1T	ratio k	random	15	1	0	1577	3337.20	1588.71	5.721
UCB1T	ratio k	random	10	1	0	1577	2901.20	1262.22	5.992
UCB1T	top k	tolerance	5	.3	2.8	1578	1838.70	131.18	30.039
UCB1T	top k	tolerance	10	.8	2.8	1580	1939.40	235.60	4.992
UCB	top k	random	15	.8	2.8	1583	3255.00	1757.31	.794
UCB	top k	random	5	.3	0	1583	3648.60	2136.03	1.634
UCB	top k	random	10	1	0	1583	3451.60	2094.03	3.372
UCB	ratio k	random	5	.5	2.8	1588	5819.70	3215.99	.441
UCB1T	ratio k	random	10	.8	1.4	1591	3953.30	2378.61	2.344
UCB	ratio k	random	5	.8	1.4	1602	3413.60	1617.10	.687
UCB1T	top k	random	15	.8	0	1606	3215.40	1457.51	2.597
UCB1T	ratio k	random	10	.5	0	1615	2917.70	1738.06	1.223
UCB1T	top k	random	5	0	1.4	1615	2243.60	798.90	2.994
UCB1T	top k	random	15	.8	1.4	1615	3428.80	2035.35	3.857
UCB1T	top k	random	10	0	0	1623	4175.70	2223.66	3.049
UCB1T	ratio k	greedy	5	.3	0	1624	1624.00		.749
UCB	top k	random	5	1	1.4	1627	3999.30	2670.48	.513
UCB	top k	random	5	0	0	1629	2414.10	1280.07	.261
UCB1T	top k	random	5	1	2.8	1633	2838.30	1236.17	3.507
UCB1T	ratio k	random	5	1	2.8	1644	3006.80	1803.40	.479
UCB	top k	random	15	1	0	1647	2351.30	1096.41	1.609
UCB	top k	random	15	.3	1.4	1651	3986.00	2543.37	1.927
UCB1T	ratio k	random	5	.8	1.4	1651	2863.80	1129.95	3.658
UCB	top k	random	10	0	0	1658	2117.20	621.46	3.543
UCB1T	ratio k	random	5	.5	0	1659	4723.70	1707.27	3.918



UCB	top k	random	10	1	2.8	1660	4102.50	2659.39	.610
UCB1T	top k	tolerance	5	.8	1.4	1660	1874.60	188.31	47.533
UCB1T	ratio k	random	10	.8	0	1661	3054.30	1542.70	1.292
UCB	top k	random	10	.8	2.8	1661	4508.90	3139.63	1.591
UCB1T	ratio k	random	5	.8	0	1662	1940.40	231.25	4.169
UCB1T	ratio k	tolerance	5	.8	2.8	1662	1837.90	117.26	6.998
UCB1T	ratio k	greedy	5	.3	1.4	1663	1663.00		.480
UCB	ratio k	tolerance	5	.3	0	1663	1865.90	178.68	1.407
UCB	ratio k	random	10	1	2.8	1663	4552.30	3487.52	1.795
UCB1T	ratio k	tolerance	5	0	2.8	1663	1927.80	140.83	3.367
UCB1T	top k	tolerance	15	.5	2.8	1663	1829.20	105.51	4.329
UCB1T	ratio k	random	10	.3	2.8	1663	3892.50	2094.32	4.499
UCB	top k	tolerance	5	0	0	1663	1838.30	154.11	9.751
UCB1T	ratio k	random	10	0	0	1666	2367.30	814.70	1.318
UCB	ratio k	random	5	.3	2.8	1666	5948.00	4768.95	1.669
UCB1T	ratio k	tolerance	5	.3	1.4	1666	1941.90	195.48	7.231
UCB1T	top k	tolerance	15	0	1.4	1666	1904.90	142.41	9.858
UCB1T	top k	random	15	.8	2.8	1668	2303.00	1087.21	4.729
UCB1T	ratio k	random	10	.3	0	1673	3451.40	2083.35	4.420
UCB	top k	tolerance	10	1	0	1674	1853.70	131.34	2.065
UCB1T	top k	tolerance	10	1	1.4	1674	1878.20	129.77	5.518
UCB	top k	random	5	1	0	1678	2416.90	959.20	1.502
UCB1T	ratio k	tolerance	10	0	1.4	1678	1914.00	147.42	3.342
UCB	top k	tolerance	5	1	0	1678	1867.30	151.01	3.844
UCB	ratio k	random	5	.5	1.4	1681	5446.30	3691.78	.327
UCB	ratio k	tolerance	10	.5	0	1689	1904.90	154.68	2.938
UCB1T	top k	random	10	.5	1.4	1689	2506.30	1778.12	4.497
UCB1T	ratio k	greedy	15	.3	0	1690	1690.00		1.255
UCB	top k	random	15	.5	2.8	1691	7503.60	5126.19	1.821
UCB	top k	random	5	.3	1.4	1695	4332.60	2620.35	.510
UCB	ratio k	tolerance	5	0	0	1695	1905.40	153.02	3.821
UCB	ratio k	tolerance	15	1	0	1695	1890.90	135.28	6.258
UCB1T	ratio k	tolerance	10	1	1.4	1695	1859.40	127.66	6.288
UCB1T	ratio k	random	5	.5	2.8	1696	2727.90	1650.15	2.736
UCB1T	ratio k	random	15	0	2.8	1698	3103.20	2377.88	1.629
UCB1T	top k	greedy	5	.8	2.8	1698	1698.00		3.695
UCB1T	top k	random	10	.8	0	1698	2707.90	1578.17	4.028
UCB1T	top k	tolerance	5	1	2.8	1698	1864.90	122.14	35.399
UCB	ratio k	random	10	.5	1.4	1703	3470.90	2151.66	.579
UCB	ratio k	random	10	.3	1.4	1704	4665.60	2019.66	.354
UCB	top k	random	10	.5	1.4	1704	5167.30	2683.04	.807

UCB	top k	random	5	.8	1.4	1706	3614.60	1951.29	.381
UCB	top k	random	5	.5	1.4	1706	4569.00	2297.57	1.213
UCB1T	top k	tolerance	10	.8	1.4	1708	1906.30	132.86	2.060
UCB	top k	random	5	.5	0	1709	2336.70	1046.92	1.765
UCB1T	top k	random	5	.3	0	1709	3106.80	1567.96	2.854
UCB1T	ratio k	tolerance	15	.8	0	1710	1881.00	143.86	15.660
UCB	top k	greedy	10	.3	0	1711	1711.00		.738
UCB	ratio k	random	5	.5	0	1715	3376.90	2127.71	1.448
UCB1T	ratio k	random	15	.5	1.4	1717	3660.50	2148.20	4.281
UCB1T	top k	random	5	1	0	1718	3008.10	1546.53	1.967
UCB1T	top k	greedy	5	.5	2.8	1720	1720.00		3.694
UCB1T	top k	tolerance	5	.5	0	1720	1858.30	108.43	5.149
UCB1T	ratio k	random	15	.8	0	1720	3732.40	1699.79	6.089
UCB1T	top k	random	10	.3	2.8	1724	2674.40	1285.84	3.201
UCB1T	top k	random	5	.8	2.8	1726	2636.60	1126.54	1.093
UCB	ratio k	random	5	0	1.4	1728	4667.50	2998.11	.755
UCB	ratio k	random	10	0	2.8	1729	5947.60	3119.40	1.541
UCB1T	top k	tolerance	5	1	1.4	1729	1885.90	169.41	34.003
UCB1T	top k	random	10	1	0	1730	3578.90	2090.43	.633
UCB	ratio k	random	15	1	0	1730	2956.50	1754.33	2.322
UCB1T	ratio k	greedy	15	1	0	1734	1734.00		1.119
UCB	top k	random	10	.3	1.4	1734	6041.60	3811.51	1.387
UCB	ratio k	tolerance	15	.8	0	1734	1937.30	160.39	2.265
UCB1T	top k	tolerance	10	.3	1.4	1740	1937.60	141.06	5.471
UCB1T	top k	greedy	10	.8	0	1741	1741.00		.881
UCB1T	ratio k	random	10	.5	2.8	1741	3925.10	2795.34	5.267
UCB	top k	tolerance	10	.5	0	1741	1849.90	89.06	6.946
UCB1T	ratio k	tolerance	15	.3	2.8	1741	1966.30	171.06	15.788
UCB	top k	random	5	.3	2.8	1742	5442.90	2963.68	.342
UCB	top k	greedy	5	.3	0	1742	1742.00		1.340
UCB1T	ratio k	random	15	0	1.4	1743	3496.30	1585.15	2.786
UCB1T	top k	random	10	.3	1.4	1744	2799.80	1512.75	3.386
UCB1T	ratio k	tolerance	10	.8	2.8	1744	1942.20	114.58	7.479
UCB	top k	tolerance	15	1	0	1745	1888.90	110.64	2.671
UCB1T	top k	random	15	.3	2.8	1746	3621.40	1663.32	.667
UCB1T	top k	random	15	.5	0	1746	3835.00	1661.25	5.242
UCB1T	top k	random	5	.3	1.4	1748	2388.70	1026.37	.811
UCB	ratio k	random	5	.8	0	1752	3143.70	1727.99	.330
UCB	top k	random	10	0	1.4	1752	6513.10	2763.61	.388
UCB1T	ratio k	greedy	15	.3	2.8	1752	1752.00		.822
UCB	top k	random	15	.3	0	1752	2640.80	1776.83	.905

UCB1T	ratio k	random	5	.8	2.8	1752	2631.00	1651.86	1.930
UCB1T	top k	random	15	0	0	1754	3047.90	1422.93	4.130
UCB1T	top k	random	15	0	1.4	1755	3598.20	1915.88	.549
UCB1T	ratio k	random	10	.8	2.8	1755	3665.50	1700.67	4.641
UCB	ratio k	greedy	5	.3	0	1757	1757.00		.308
UCB1T	ratio k	random	15	.3	0	1758	3812.60	1938.27	1.372
UCB1T	top k	greedy	15	1	1.4	1759	1759.00		1.236
UCB1T	top k	random	10	.5	0	1767	3736.40	2107.52	3.212
UCB	ratio k	random	15	.5	0	1771	2491.90	1254.11	.934
UCB1T	ratio k	random	15	.5	2.8	1771	3457.70	1985.88	5.624
UCB	ratio k	greedy	15	0	0	1773	1773.00		.384
UCB1T	ratio k	random	15	.3	1.4	1773	3853.40	2207.96	.599
UCB1T	top k	greedy	15	.3	0	1773	1773.00		1.033
UCB1T	top k	greedy	15	1	2.8	1774	1774.00		.618
UCB	ratio k	greedy	15	.5	0	1778	1778.00		.669
UCB1T	ratio k	greedy	15	.8	0	1778	1778.00		.732
UCB1T	top k	greedy	10	.5	2.8	1778	1778.00		.835
UCB	ratio k	greedy	10	.8	0	1778	1778.00		.852
UCB1T	ratio k	tolerance	15	0	0	1778	1959.60	159.75	1.432
UCB1T	top k	greedy	5	0	0	1778	1778.00		2.859
UCB1T	ratio k	greedy	5	.8	1.4	1778	1778.00		3.065
UCB1T	top k	tolerance	10	0	1.4	1778	1907.20	101.60	8.331
UCB1T	ratio k	random	15	.8	2.8	1779	3335.30	2191.39	5.037
UCB1T	top k	tolerance	15	.8	1.4	1780	1896.70	105.56	6.394
UCB	top k	random	10	.8	1.4	1782	6276.60	2458.38	.204
UCB1T	ratio k	random	15	1	2.8	1782	3349.50	1615.37	.677
UCB	ratio k	random	15	.3	0	1782	3142.90	2500.95	2.405
UCB1T	top k	greedy	10	.3	2.8	1783	1783.00		1.084
UCB	ratio k	random	10	.5	2.8	1783	6292.70	2661.63	1.304
UCB	top k	random	10	1	1.4	1783	5389.80	2177.70	1.721
UCB	top k	greedy	5	.8	0	1783	1783.00		2.146
UCB1T	top k	random	5	.3	2.8	1783	3757.60	1922.41	2.584
UCB1T	top k	greedy	5	.8	1.4	1783	1783.00		2.623
UCB1T	ratio k	tolerance	10	0	2.8	1783	1951.40	99.87	5.562
UCB	top k	random	5	1	2.8	1791	4959.50	2252.57	.836
UCB1T	top k	random	5	.8	1.4	1792	3362.30	1680.69	1.490
UCB	top k	random	15	0	2.8	1793	6246.20	3845.94	1.006
UCB1T	top k	random	10	.5	2.8	1795	2935.90	1756.79	.671
UCB1T	ratio k	random	10	0	1.4	1796	3402.30	1591.21	4.567
UCB1T	ratio k	random	15	.5	0	1796	2785.10	1424.75	5.271
UCB	ratio k	random	15	0	2.8	1797	4958.80	3035.26	.991

UCB1T	ratio k	tolerance	5	.5	1.4	1797	1952.50	142.06	6.487
UCB1T	top k	tolerance	5	.8	2.8	1797	1906.40	90.79	11.151
UCB	top k	random	10	.8	0	1798	3569.70	2296.99	.390
UCB	top k	greedy	10	.8	0	1798	1798.00		.466
UCB1T	ratio k	greedy	10	.3	0	1798	1798.00		.909
UCB1T	ratio k	greedy	10	.8	1.4	1798	1798.00		1.255
UCB1T	ratio k	tolerance	5	0	1.4	1798	1927.00	109.64	2.407
UCB1T	top k	greedy	5	1	0	1798	1798.00		2.471
UCB1T	top k	greedy	5	0	1.4	1798	1798.00		2.742
UCB	top k	random	15	0	0	1800	3375.00	1522.78	1.985
UCB1T	top k	random	15	.5	1.4	1801	3564.00	2310.60	1.718
UCB	top k	random	5	0	2.8	1802	5150.30	3980.89	.696
UCB1T	top k	random	10	.8	2.8	1804	3337.90	1579.14	3.917
UCB1T	ratio k	greedy	15	.5	0	1805	1805.00		1.020
UCB1T	top k	greedy	10	1	1.4	1805	1805.00		1.252
UCB	ratio k	random	15	1	2.8	1805	3020.50	1454.09	1.851
UCB	top k	random	15	.5	0	1810	2685.60	1291.87	2.002
UCB	ratio k	random	5	0	2.8	1811	5222.20	2594.44	1.431
UCB1T	ratio k	random	5	.3	0	1811	3121.80	1195.49	2.154
UCB1T	top k	random	15	1	1.4	1811	2746.10	1029.79	4.165
UCB	top k	random	15	1	1.4	1812	3813.70	2108.22	1.580
UCB	ratio k	tolerance	15	.5	0	1815	1933.60	109.14	7.289
UCB1T	top k	random	5	.8	0	1817	2863.50	1598.52	4.821
UCB	top k	greedy	15	.8	0	1819	1819.00		.819
UCB	top k	random	5	.8	0	1819	2653.10	1378.29	2.417
UCB	top k	random	15	.8	0	1821	2959.50	1379.33	.548
UCB	top k	greedy	5	1	0	1822	1822.00		1.778
UCB1T	top k	greedy	5	.3	0	1822	1822.00		3.199
UCB1T	ratio k	greedy	5	.5	1.4	1833	1833.00		.770
UCB1T	ratio k	greedy	10	.3	2.8	1833	1833.00		.850
UCB	ratio k	greedy	10	.5	0	1833	1833.00		.896
UCB1T	ratio k	greedy	5	.8	2.8	1833	1833.00		1.920
UCB1T	top k	greedy	5	.5	0	1833	1833.00		2.552
UCB1T	top k	greedy	5	.8	0	1833	1833.00		3.735
UCB	top k	random	10	.3	0	1837	2984.40	1548.27	.343
UCB1T	ratio k	random	5	1	0	1839	2988.30	987.59	1.384
UCB	ratio k	random	10	.3	0	1839	2936.70	1766.59	3.191
UCB	ratio k	random	10	1	0	1840	2954.60	1488.65	1.376
UCB	ratio k	random	10	0	0	1844	3452.30	1914.01	2.469
UCB1T	top k	random	5	.5	2.8	1845	3487.20	1871.64	1.131
UCB1T	top k	random	10	.8	1.4	1845	2470.80	1026.68	6.022

UCB	top k	greedy	10	0	0	1846	1846.00		.531
UCB1T	ratio k	greedy	5	.8	0	1846	1846.00		2.147
UCB1T	top k	greedy	15	.5	0	1847	1847.00		1.365
UCB1T	top k	greedy	5	1	2.8	1847	1847.00		2.332
UCB1T	top k	random	15	1	2.8	1847	3483.10	1585.60	5.841
UCB	ratio k	random	10	.8	1.4	1849	6728.60	2720.00	.551
UCB1T	top k	random	5	0	0	1849	2920.70	1313.92	3.279
UCB1T	top k	greedy	5	.3	2.8	1850	1850.00		2.709
UCB1T	ratio k	greedy	5	.5	0	1851	1851.00		.718
UCB1T	top k	random	10	0	1.4	1851	2614.40	1646.27	.728
UCB1T	ratio k	greedy	5	1	1.4	1851	1851.00		2.352
UCB	top k	random	5	.5	2.8	1855	5007.10	2533.14	1.484
UCB1T	ratio k	random	5	.3	1.4	1855	3247.80	1139.79	1.912
UCB1T	ratio k	random	10	1	2.8	1855	4480.10	2346.39	6.490
UCB1T	top k	greedy	10	.5	0	1856	1856.00		.920
UCB1T	top k	greedy	10	.3	1.4	1856	1856.00		1.010
UCB1T	top k	greedy	5	0	2.8	1856	1856.00		1.684
UCB1T	top k	random	10	1	1.4	1856	3578.30	2326.86	5.097
UCB	top k	greedy	15	.5	0	1861	1861.00		.648
UCB1T	ratio k	greedy	5	.3	2.8	1861	1861.00		.896
UCB1T	top k	greedy	10	.5	1.4	1861	1861.00		.970
UCB1T	ratio k	greedy	10	0	2.8	1861	1861.00		1.369
UCB	top k	tolerance	10	.3	0	1861	1980.50	93.92	10.326
UCB	top k	greedy	15	.3	0	1862	1862.00		.438
UCB1T	ratio k	random	15	.3	2.8	1863	3343.90	2736.34	2.367
UCB1T	ratio k	random	5	0	1.4	1864	4090.10	2412.61	1.531
UCB1T	top k	random	5	1	1.4	1865	2448.10	971.91	2.612
UCB	ratio k	random	15	1	1.4	1866	6731.90	3376.22	1.466
UCB	ratio k	random	5	.3	0	1871	3780.50	1986.19	.225
UCB	ratio k	random	15	0	0	1871	3548.20	2364.24	1.076
UCB	top k	random	10	.5	2.8	1871	7492.80	3399.24	1.248
UCB1T	ratio k	greedy	10	0	0	1880	1880.00		1.468
UCB1T	ratio k	random	5	.3	2.8	1881	3384.70	1996.79	3.608
UCB	top k	random	10	.5	0	1886	2432.70	1384.12	.622
UCB	ratio k	random	15	.8	1.4	1888	5276.10	3156.41	.173
UCB1T	ratio k	random	5	0	0	1889	4056.40	1786.38	.491
UCB1T	ratio k	random	5	1	1.4	1891	2279.90	630.36	3.158
UCB	top k	random	15	.8	1.4	1894	6800.10	3085.13	1.668
UCB1T	top k	random	15	0	2.8	1894	3414.70	2159.40	5.490
UCB	ratio k	random	10	.5	0	1899	3775.90	1612.17	2.071
UCB1T	ratio k	random	10	1	1.4	1900	3640.40	1458.18	3.926

UCB	top k	random	5	.8	2.8	1901	4658.20	2594.64	.823
UCB	top k	random	15	.5	1.4	1902	7399.70	2650.98	.834
UCB1T	ratio k	random	15	1	1.4	1904	3899.60	2062.78	5.620
UCB1T	top k	greedy	15	0	2.8	1905	1905.00		.984
UCB	ratio k	greedy	5	.5	0	1910	1910.00		.476
UCB	ratio k	greedy	5	.8	0	1910	1910.00		1.994
UCB	ratio k	random	5	.8	2.8	1915	5654.40	3277.84	.157
UCB1T	ratio k	greedy	5	.5	2.8	1916	1916.00		.606
UCB1T	top k	greedy	10	1	0	1917	1917.00		1.081
UCB	top k	random	10	0	2.8	1917	4964.00	2954.44	1.358
UCB1T	ratio k	greedy	15	1	1.4	1937	1937.00		.818
UCB	ratio k	random	15	.5	1.4	1941	7036.40	3592.63	1.856
UCB1T	ratio k	greedy	5	1	0	1941	1941.00		1.996
UCB1T	ratio k	greedy	5	0	0	1943	1943.00		.723
UCB	ratio k	random	10	0	1.4	1945	4955.90	2402.73	1.456
UCB	ratio k	greedy	15	1	0	1951	1951.00		.764
UCB1T	top k	greedy	5	1	1.4	1951	1951.00		2.631
UCB1T	ratio k	random	10	.5	1.4	1957	3272.20	1591.64	2.647
UCB1T	top k	greedy	15	0	0	1959	1959.00		1.223
UCB1T	ratio k	greedy	10	0	1.4	1960	1960.00		1.371
UCB1T	top k	greedy	10	.3	0	1961	1961.00		1.035
UCB1T	top k	greedy	15	.8	0	1962	1962.00		.757
UCB	ratio k	greedy	10	1	0	1969	1969.00		.610
UCB	top k	greedy	15	1	0	1971	1971.00		1.004
UCB	ratio k	greedy	5	0	0	1972	1972.00		.390
UCB1T	top k	greedy	10	1	2.8	1972	1972.00		.880
UCB1T	ratio k	greedy	15	.8	2.8	1972	1972.00		1.064
UCB1T	ratio k	greedy	10	.8	2.8	1972	1972.00		1.067
UCB	top k	greedy	10	1	0	1972	1972.00		1.072
UCB1T	ratio k	greedy	15	0	1.4	1972	1972.00		1.131
UCB1T	ratio k	greedy	15	0	0	1972	1972.00		1.301
UCB	ratio k	random	5	0	0	1972	4122.00	2790.15	2.345
UCB1T	top k	greedy	5	.5	1.4	1972	1972.00		2.618
UCB	top k	greedy	15	0	0	1977	1977.00		.635
UCB1T	top k	greedy	15	.8	1.4	1979	1979.00		2.432
UCB1T	ratio k	greedy	15	.5	1.4	1992	1992.00		.746
UCB1T	ratio k	greedy	15	1	2.8	1992	1992.00		1.310
UCB1T	top k	greedy	10	.8	1.4	1994	1994.00		1.255
UCB1T	top k	greedy	15	.3	2.8	1995	1995.00		1.126
UCB	top k	random	10	.3	2.8	1999	5307.10	1971.19	1.000
UCB	top k	random	15	1	2.8	2001	6131.90	2126.47	1.596

UCB	ratio k	random	5	1	0	2010	4339.90	1942.49	1.911
UCB	ratio k	greedy	10	0	0	2029	2029.00		.802
UCB1T	top k	greedy	15	1	0	2029	2029.00		1.061
UCB1T	ratio k	greedy	15	.5	2.8	2035	2035.00		1.150
UCB	ratio k	random	15	0	1.4	2040	5329.50	2769.75	1.122
UCB	ratio k	greedy	10	.3	0	2044	2044.00		.755
UCB1T	ratio k	greedy	5	1	2.8	2053	2053.00		2.321
UCB1T	top k	greedy	10	.8	2.8	2068	2068.00		.915
UCB1T	ratio k	greedy	10	.5	0	2074	2074.00		1.003
UCB	top k	random	15	0	1.4	2100	6246.40	3562.81	.221
UCB1T	ratio k	random	10	0	2.8	2105	3032.10	980.16	5.624
UCB1T	ratio k	greedy	15	0	2.8	2108	2108.00		1.132
UCB	ratio k	greedy	15	.3	0	2116	2116.00		.962
UCB1T	top k	greedy	10	0	0	2116	2116.00		1.385
UCB1T	ratio k	greedy	10	.5	2.8	2128	2128.00		.988
UCB1T	top k	greedy	10	0	2.8	2128	2128.00		1.014
UCB1T	top k	greedy	15	0	1.4	2165	2165.00		1.313
UCB1T	top k	greedy	10	0	1.4	2175	2175.00		1.272
UCB	ratio k	greedy	15	.8	0	2188	2188.00		.955
UCB1T	ratio k	greedy	15	.8	1.4	2188	2188.00		1.447
UCB	ratio k	random	15	.5	2.8	2189	4892.40	1736.65	1.437
UCB1T	top k	greedy	15	.5	1.4	2211	2211.00		2.119
UCB	ratio k	random	10	.8	2.8	2229	8014.20	3618.47	.742
UCB1T	ratio k	greedy	10	1	2.8	2258	2258.00		1.518
UCB1T	top k	greedy	15	.3	1.4	2261	2261.00		1.174
UCB1T	ratio k	greedy	10	.3	1.4	2273	2273.00		1.990
UCB	ratio k	random	15	.3	1.4	2437	5466.50	1930.17	1.931
UCB	ratio k	random	5	.3	1.4	2564	5817.90	3017.59	.167
UCB1T	ratio k	greedy	5	0	2.8	2945	2945.00		.469

### C.1.2 Solution not found

[illegible]

## C.2 Instance 2

### C.2.1 Solution found

Selec policy	Exp policy	Simu policy	N° chil- drens	Ratio	Cp	Best cost	Mean	Std	T(s)
UCB	ratio k	greedy	5	.8	0	1498	1498.00		.082
UCB	ratio k	greedy	5	.3	1.4	1498	1498.00		.083
UCB1T	ratio k	greedy	10	1	1.4	1498	1498.00		.084
UCB1T	ratio k	greedy	15	.5	0	1498	1498.00		.087
UCB	ratio k	greedy	5	.8	1.4	1498	1498.00		.087
UCB	ratio k	greedy	15	.3	2.8	1498	1498.00		.087
UCB	ratio k	greedy	15	0	2.8	1498	1498.00		.088
UCB1T	ratio k	greedy	15	0	1.4	1498	1498.00		.088
UCB	top k	tolerance	10	0	2.8	1498	1498.00	0.00	.089
UCB1T	ratio k	greedy	10	1	2.8	1498	1498.00		.089
UCB	ratio k	greedy	10	.5	2.8	1498	1498.00		.089
UCB	top k	tolerance	15	.5	2.8	1498	1498.00	0.00	.090
UCB1T	ratio k	tolerance	15	1	1.4	1498	1498.00	0.00	.091
UCB1T	ratio k	tolerance	15	.3	1.4	1498	1498.00	0.00	.092
UCB	ratio k	greedy	10	0	2.8	1498	1498.00		.092
UCB	ratio k	greedy	5	.5	0	1498	1498.00		.092
UCB	top k	tolerance	10	0	1.4	1498	1498.00	0.00	.093
UCB	top k	greedy	15	.5	1.4	1498	1498.00		.093
UCB	top k	tolerance	10	.8	2.8	1498	1498.00	0.00	.093
UCB	ratio k	greedy	10	.3	2.8	1498	1498.00		.093
UCB	ratio k	tolerance	15	1	0	1498	1498.00	0.00	.094
UCB	top k	greedy	15	1	2.8	1498	1498.00		.094
UCB	top k	tolerance	15	.8	2.8	1498	1498.00	0.00	.095
UCB	ratio k	greedy	15	1	0	1498	1498.00		.095
UCB1T	ratio k	greedy	15	.3	0	1498	1498.00		.095
UCB	top k	greedy	15	0	0	1498	1498.00		.095
UCB1T	ratio k	greedy	10	0	1.4	1498	1498.00		.096
UCB1T	ratio k	greedy	10	.8	2.8	1498	1498.00		.096
UCB1T	ratio k	greedy	10	.8	1.4	1498	1498.00		.096
UCB	ratio k	greedy	15	0	1.4	1498	1498.00		.096
UCB1T	top k	greedy	15	1	1.4	1498	1498.00		.097
UCB	ratio k	greedy	5	.3	0	1498	1498.00		.097
UCB	top k	greedy	15	.3	1.4	1498	1498.00		.097
UCB1T	ratio k	tolerance	15	1	0	1498	1498.00	0.00	.098



UCB1T	ratio k	greedy	15	.8	2.8	1498	1498.00		.098
UCB	ratio k	tolerance	5	.5	0	1498	1498.00	0.00	.099
UCB1T	ratio k	greedy	15	.3	1.4	1498	1498.00		.099
UCB	top k	greedy	10	0	2.8	1498	1498.00		.099
UCB	ratio k	greedy	15	1	2.8	1498	1498.00		.099
UCB	ratio k	greedy	10	0	1.4	1498	1498.00		.100
UCB1T	ratio k	greedy	15	.5	1.4	1498	1498.00		.100
UCB	top k	greedy	10	.5	1.4	1498	1498.00		.100
UCB1T	top k	greedy	15	0	1.4	1498	1498.00		.100
UCB1T	ratio k	greedy	10	.5	0	1498	1498.00		.100
UCB1T	top k	greedy	10	0	2.8	1498	1498.00		.100
UCB1T	ratio k	tolerance	5	.5	2.8	1498	1498.00	0.00	.101
UCB1T	ratio k	greedy	10	.8	0	1498	1498.00		.101
UCB1T	ratio k	tolerance	10	0	2.8	1498	1498.00	0.00	.101
UCB	ratio k	greedy	15	1	1.4	1498	1498.00		.101
UCB1T	ratio k	tolerance	15	.3	2.8	1498	1498.00	0.00	.101
UCB1T	ratio k	tolerance	15	.5	1.4	1498	1498.00	0.00	.102
UCB	ratio k	greedy	15	0	0	1498	1498.00		.102
UCB1T	ratio k	greedy	10	.5	1.4	1498	1498.00		.102
UCB	top k	greedy	10	.3	1.4	1498	1498.00		.103
UCB1T	top k	greedy	15	.8	2.8	1498	1498.00		.103
UCB	ratio k	tolerance	15	1	2.8	1498	1498.00	0.00	.103
UCB1T	ratio k	tolerance	15	0	0	1498	1498.00	0.00	.104
UCB	top k	greedy	15	.8	2.8	1498	1498.00		.104
UCB	ratio k	greedy	15	.8	1.4	1498	1498.00		.104
UCB	top k	greedy	10	0	1.4	1498	1498.00		.104
UCB	ratio k	greedy	10	.8	2.8	1498	1498.00		.104
UCB	ratio k	greedy	10	1	1.4	1498	1498.00		.105
UCB	ratio k	greedy	15	.8	0	1498	1498.00		.105
UCB	ratio k	greedy	10	.3	0	1498	1498.00		.105
UCB	top k	tolerance	10	1	1.4	1498	1498.00	0.00	.106
UCB	top k	greedy	10	.8	1.4	1498	1498.00		.106
UCB	ratio k	tolerance	15	.5	1.4	1498	1498.00	0.00	.106
UCB	top k	tolerance	15	.3	1.4	1498	1498.00	0.00	.106
UCB	ratio k	greedy	15	.5	2.8	1498	1498.00		.107
UCB1T	ratio k	tolerance	15	0	2.8	1498	1498.00	0.00	.107
UCB	top k	greedy	10	.8	0	1498	1498.00		.108
UCB	ratio k	greedy	5	0	0	1498	1498.00		.108
UCB	top k	greedy	15	0	1.4	1498	1498.00		.108
UCB	ratio k	greedy	15	.3	1.4	1498	1498.00		.109
UCB	top k	tolerance	15	.3	0	1498	1498.00	0.00	.109

UCB	ratio k	greedy	10	.5	1.4	1498	1498.00		.109
UCB	ratio k	greedy	10	.3	1.4	1498	1498.00		.109
UCB	ratio k	greedy	10	.8	1.4	1498	1498.00		.109
UCB	top k	tolerance	10	.8	1.4	1498	1498.00	0.00	.110
UCB1T	ratio k	greedy	15	0	0	1498	1498.00		.110
UCB1T	top k	tolerance	10	1	2.8	1498	1498.00	0.00	.110
UCB1T	ratio k	greedy	10	.3	0	1498	1498.00		.110
UCB1T	top k	greedy	10	.3	2.8	1498	1498.00		.110
UCB	top k	tolerance	10	.5	1.4	1498	1498.00	0.00	.110
UCB1T	top k	greedy	10	.3	0	1498	1498.00		.110
UCB	ratio k	greedy	10	.8	0	1498	1498.00		.110
UCB1T	ratio k	greedy	10	.3	1.4	1498	1498.00		.110
UCB1T	top k	tolerance	10	.5	0	1498	1498.00	0.00	.110
UCB	ratio k	tolerance	15	.5	2.8	1498	1498.00	0.00	.111
UCB1T	top k	greedy	15	.3	0	1498	1498.00		.111
UCB	top k	greedy	10	.8	2.8	1498	1498.00		.111
UCB	ratio k	greedy	10	1	2.8	1498	1498.00		.111
UCB	ratio k	tolerance	15	.3	1.4	1498	1498.00	0.00	.111
UCB1T	top k	greedy	10	1	1.4	1498	1498.00		.111
UCB	top k	tolerance	15	1	1.4	1498	1498.00	0.00	.112
UCB	top k	greedy	15	.3	0	1498	1498.00		.112
UCB1T	top k	greedy	15	.5	1.4	1498	1498.00		.112
UCB	top k	tolerance	15	.8	1.4	1498	1498.00	0.00	.112
UCB1T	ratio k	tolerance	5	0	2.8	1498	1498.00	0.00	.112
UCB1T	ratio k	greedy	15	.5	2.8	1498	1498.00		.112
UCB1T	top k	greedy	15	.8	1.4	1498	1498.00		.112
UCB	top k	greedy	15	.8	1.4	1498	1498.00		.112
UCB1T	top k	tolerance	15	.5	2.8	1498	1498.00	0.00	.112
UCB1T	ratio k	greedy	5	.5	2.8	1498	1498.00		.113
UCB	top k	tolerance	10	1	2.8	1498	1498.00	0.00	.113
UCB	ratio k	tolerance	10	.3	1.4	1498	1498.00	0.00	.113
UCB1T	ratio k	tolerance	10	.5	0	1498	1498.00	0.00	.113
UCB	ratio k	tolerance	15	.8	2.8	1498	1498.00	0.00	.114
UCB1T	ratio k	tolerance	15	.5	0	1498	1498.00	0.00	.114
UCB1T	top k	greedy	10	.5	2.8	1498	1498.00		.114
UCB	ratio k	greedy	15	.8	2.8	1498	1498.00		.114
UCB	top k	greedy	15	0	2.8	1498	1498.00		.114
UCB	top k	greedy	10	.5	2.8	1498	1498.00		.114
UCB	top k	tolerance	15	1	2.8	1498	1498.00	0.00	.115
UCB	top k	greedy	10	.3	0	1498	1498.00		.115
UCB1T	ratio k	greedy	10	.3	2.8	1498	1498.00		.115

UCB1T	top k	greedy	10	.3	1.4	1498	1498.00		.115
UCB1T	top k	greedy	10	.8	2.8	1498	1498.00		.116
UCB1T	ratio k	tolerance	15	.8	0	1498	1498.00	0.00	.116
UCB	ratio k	greedy	15	.5	1.4	1498	1498.00		.116
UCB1T	top k	greedy	15	.3	1.4	1498	1498.00		.116
UCB1T	top k	greedy	15	1	0	1498	1498.00		.116
UCB	ratio k	tolerance	5	0	0	1498	1498.00	0.00	.117
UCB	top k	greedy	10	1	0	1498	1498.00		.117
UCB	ratio k	tolerance	5	.8	1.4	1498	1498.00	0.00	.117
UCB1T	top k	tolerance	10	.5	2.8	1498	1498.00	0.00	.117
UCB1T	top k	tolerance	10	.3	1.4	1498	1498.00	0.00	.117
UCB	top k	greedy	10	.3	2.8	1498	1498.00		.118
UCB1T	top k	tolerance	10	0	1.4	1498	1498.00	0.00	.118
UCB	ratio k	greedy	15	.3	0	1498	1498.00		.118
UCB	top k	greedy	15	.5	2.8	1498	1498.00		.118
UCB	top k	greedy	15	1	0	1498	1498.00		.118
UCB	top k	tolerance	10	.3	2.8	1498	1498.00	0.00	.119
UCB1T	top k	greedy	10	.5	0	1498	1498.00		.119
UCB	ratio k	greedy	5	.5	2.8	1498	1498.00		.119
UCB1T	top k	greedy	10	0	0	1498	1498.00		.119
UCB1T	ratio k	tolerance	5	.3	1.4	1498	1498.00	0.00	.120
UCB	ratio k	tolerance	10	1	0	1498	1498.00	0.00	.120
UCB	top k	greedy	10	1	1.4	1498	1498.00		.120
UCB1T	ratio k	greedy	15	.8	1.4	1498	1498.00		.120
UCB1T	top k	greedy	10	1	0	1498	1498.00		.120
UCB1T	ratio k	greedy	10	1	0	1498	1498.00		.120
UCB	ratio k	greedy	10	0	0	1498	1498.00		.121
UCB1T	top k	greedy	15	.5	2.8	1498	1498.00		.121
UCB1T	top k	greedy	15	.3	2.8	1498	1498.00		.121
UCB	ratio k	tolerance	10	0	0	1498	1498.00	0.00	.121
UCB	ratio k	greedy	10	.5	0	1498	1498.00		.121
UCB	ratio k	tolerance	15	.3	2.8	1498	1498.00	0.00	.121
UCB1T	ratio k	greedy	15	.3	2.8	1498	1498.00		.122
UCB1T	top k	greedy	15	.5	0	1498	1498.00		.122
UCB1T	top k	tolerance	10	.3	0	1498	1498.00	0.00	.122
UCB1T	ratio k	tolerance	15	.3	0	1498	1498.00	0.00	.122
UCB1T	ratio k	tolerance	15	0	1.4	1498	1498.00	0.00	.123
UCB1T	ratio k	greedy	15	1	2.8	1498	1498.00		.123
UCB1T	top k	tolerance	15	.8	0	1498	1498.00	0.00	.123
UCB	ratio k	tolerance	10	0	1.4	1498	1498.00	0.00	.123
UCB	top k	greedy	10	.5	0	1498	1498.00		.124

UCB1T	top k	tolerance	15	.5	1.4	1498	1498.00	0.00	.124
UCB	top k	greedy	15	1	1.4	1498	1498.00		.124
UCB	top k	greedy	10	1	2.8	1498	1498.00		.124
UCB1T	ratio k	greedy	15	0	2.8	1498	1498.00		.125
UCB1T	top k	tolerance	10	.8	0	1498	1498.00	0.00	.125
UCB1T	top k	tolerance	15	0	1.4	1498	1498.00	0.00	.125
UCB1T	ratio k	greedy	15	1	0	1498	1498.00		.125
UCB1T	top k	tolerance	10	.8	2.8	1498	1498.00	0.00	.125
UCB	ratio k	tolerance	5	.8	0	1498	1498.00	0.00	.125
UCB	ratio k	greedy	15	.5	0	1498	1498.00		.126
UCB1T	top k	tolerance	10	.8	1.4	1498	1498.00	0.00	.126
UCB1T	ratio k	tolerance	5	0	0	1498	1498.00	0.00	.126
UCB1T	top k	tolerance	10	1	1.4	1498	1498.00	0.00	.126
UCB	ratio k	greedy	10	1	0	1498	1498.00		.126
UCB1T	top k	tolerance	10	.5	1.4	1498	1498.00	0.00	.126
UCB1T	top k	tolerance	15	.8	1.4	1498	1498.00	0.00	.126
UCB	top k	tolerance	15	1	0	1498	1498.00	0.00	.127
UCB	top k	tolerance	15	0	0	1498	1498.00	0.00	.127
UCB	ratio k	tolerance	5	.3	1.4	1498	1498.00	0.00	.127
UCB1T	top k	tolerance	15	0	2.8	1498	1498.00	0.00	.127
UCB1T	top k	greedy	10	.8	1.4	1498	1498.00		.128
UCB1T	ratio k	greedy	10	.5	2.8	1498	1498.00		.128
UCB	top k	tolerance	15	.5	1.4	1498	1498.00	0.00	.129
UCB1T	top k	greedy	15	.8	0	1498	1498.00		.129
UCB1T	top k	greedy	10	1	2.8	1498	1498.00		.130
UCB1T	top k	tolerance	15	.3	0	1498	1498.00	0.00	.130
UCB1T	top k	greedy	15	0	2.8	1498	1498.00		.130
UCB	ratio k	tolerance	10	.5	1.4	1498	1498.00	0.00	.131
UCB	ratio k	tolerance	10	.5	0	1498	1498.00	0.00	.131
UCB1T	ratio k	tolerance	15	.8	1.4	1498	1498.00	0.00	.132
UCB1T	ratio k	greedy	15	1	1.4	1498	1498.00		.133
UCB1T	ratio k	tolerance	15	.5	2.8	1498	1498.00	0.00	.133
UCB1T	top k	greedy	10	.5	1.4	1498	1498.00		.134
UCB1T	top k	greedy	10	.8	0	1498	1498.00		.134
UCB	top k	greedy	15	.3	2.8	1498	1498.00		.134
UCB1T	top k	tolerance	15	1	0	1498	1498.00	0.00	.135
UCB1T	top k	tolerance	15	0	0	1498	1498.00	0.00	.135
UCB1T	ratio k	tolerance	5	.8	1.4	1498	1498.00	0.00	.136
UCB	top k	tolerance	15	0	1.4	1498	1498.00	0.00	.136
UCB1T	ratio k	tolerance	5	.5	1.4	1498	1498.00	0.00	.137
UCB1T	ratio k	tolerance	10	0	0	1498	1498.00	0.00	.137

UCB	ratio k	tolerance	10	.8	0	1498	1498.00	0.00	.139
UCB1T	top k	greedy	10	0	1.4	1498	1498.00		.140
UCB1T	top k	tolerance	10	0	2.8	1498	1498.00	0.00	.140
UCB	top k	tolerance	10	.5	2.8	1498	1498.00	0.00	.140
UCB	ratio k	tolerance	10	0	2.8	1498	1498.00	0.00	.141
UCB	ratio k	tolerance	10	.5	2.8	1498	1498.00	0.00	.141
UCB1T	ratio k	tolerance	5	.5	0	1498	1498.00	0.00	.141
UCB1T	ratio k	tolerance	10	0	1.4	1498	1498.00	0.00	.142
UCB	ratio k	tolerance	15	.8	0	1498	1498.00	0.00	.142
UCB	ratio k	tolerance	15	1	1.4	1498	1498.00	0.00	.142
UCB	ratio k	tolerance	15	0	0	1498	1498.00	0.00	.142
UCB1T	ratio k	greedy	5	.5	0	1498	1498.00		.142
UCB1T	ratio k	tolerance	10	.3	2.8	1498	1498.00	0.00	.143
UCB1T	ratio k	tolerance	10	.5	1.4	1498	1498.00	0.00	.143
UCB1T	top k	greedy	15	1	2.8	1498	1498.00		.143
UCB	top k	greedy	10	0	0	1498	1498.00		.145
UCB1T	ratio k	tolerance	10	.5	2.8	1498	1498.00	0.00	.145
UCB	ratio k	tolerance	15	.3	0	1498	1498.00	0.00	.145
UCB1T	ratio k	tolerance	15	1	2.8	1498	1498.00	0.00	.146
UCB	ratio k	tolerance	15	0	2.8	1498	1498.00	0.00	.146
UCB	ratio k	tolerance	15	.5	0	1498	1498.00	0.00	.146
UCB	ratio k	tolerance	5	.5	1.4	1498	1498.00	0.00	.147
UCB1T	ratio k	tolerance	10	.3	0	1498	1498.00	0.00	.148
UCB	top k	tolerance	15	0	2.8	1498	1498.00	0.00	.148
UCB	ratio k	tolerance	10	.3	2.8	1498	1498.00	0.00	.149
UCB	top k	tolerance	15	.5	0	1498	1498.00	0.00	.149
UCB1T	top k	tolerance	10	.3	2.8	1498	1498.00	0.00	.149
UCB1T	ratio k	tolerance	10	.8	2.8	1498	1498.00	0.00	.150
UCB1T	top k	greedy	15	0	0	1498	1498.00		.151
UCB	ratio k	tolerance	10	.3	0	1498	1498.00	0.00	.152
UCB	ratio k	tolerance	5	.3	2.8	1498	1498.00	0.00	.152
UCB	ratio k	tolerance	15	0	1.4	1498	1498.00	0.00	.152
UCB1T	top k	tolerance	15	1	2.8	1498	1498.00	0.00	.152
UCB	ratio k	tolerance	5	.5	2.8	1498	1498.00	0.00	.153
UCB	ratio k	tolerance	10	1	1.4	1498	1498.00	0.00	.153
UCB1T	top k	tolerance	15	.3	2.8	1498	1498.00	0.00	.153
UCB1T	top k	tolerance	10	0	0	1498	1498.00	0.00	.154
UCB1T	top k	tolerance	15	.5	0	1498	1498.00	0.00	.154
UCB1T	ratio k	greedy	10	0	0	1498	1498.00		.155
UCB	top k	greedy	15	.5	0	1498	1498.00		.155
UCB	top k	tolerance	10	1	0	1498	1498.00	0.00	.156

UCB	ratio k	tolerance	5	.3	0	1498	1498.00	0.00	.156
UCB1T	top k	tolerance	15	.3	1.4	1498	1498.00	0.00	.156
UCB1T	ratio k	tolerance	10	.8	0	1498	1498.00	0.00	.157
UCB1T	ratio k	greedy	10	0	2.8	1498	1498.00		.157
UCB	top k	tolerance	10	0	0	1498	1498.00	0.00	.160
UCB	top k	tolerance	15	.8	0	1498	1498.00	0.00	.161
UCB1T	ratio k	tolerance	10	1	0	1498	1498.00	0.00	.163
UCB	top k	tolerance	10	.5	0	1498	1498.00	0.00	.163
UCB	top k	tolerance	15	.3	2.8	1498	1498.00	0.00	.164
UCB1T	top k	tolerance	15	1	1.4	1498	1498.00	0.00	.164
UCB	ratio k	tolerance	10	1	2.8	1498	1498.00	0.00	.164
UCB1T	top k	tolerance	10	1	0	1498	1498.00	0.00	.165
UCB1T	ratio k	greedy	15	.8	0	1498	1498.00		.167
UCB1T	ratio k	tolerance	5	.8	0	1498	1498.00	0.00	.168
UCB1T	ratio k	tolerance	10	1	2.8	1498	1498.00	0.00	.171
UCB	ratio k	tolerance	10	.8	1.4	1498	1498.00	0.00	.174
UCB	ratio k	tolerance	15	.8	1.4	1498	1498.00	0.00	.174
UCB	top k	tolerance	10	.8	0	1498	1498.00	0.00	.175
UCB	ratio k	tolerance	10	.8	2.8	1498	1498.00	0.00	.175
UCB	ratio k	tolerance	5	.8	2.8	1498	1498.00	0.00	.176
UCB	ratio k	tolerance	5	0	1.4	1498	1498.00		.177
UCB1T	ratio k	tolerance	10	.8	1.4	1498	1498.00	0.00	.179
UCB	top k	tolerance	10	.3	1.4	1498	1498.00	0.00	.183
UCB1T	ratio k	tolerance	10	1	1.4	1498	1498.00	0.00	.186
UCB1T	ratio k	tolerance	5	.3	0	1498	1498.00	0.00	.186
UCB1T	ratio k	tolerance	15	.8	2.8	1498	1498.00	0.00	.189
UCB	top k	greedy	15	.8	0	1498	1498.00		.191
UCB1T	ratio k	tolerance	5	.3	2.8	1498	1498.00	0.00	.192
UCB1T	top k	tolerance	15	.8	2.8	1498	1498.00	0.00	.200
UCB1T	ratio k	tolerance	10	.3	1.4	1498	1498.00	0.00	.219
UCB	top k	tolerance	10	.3	0	1498	1498.00	0.00	.231
UCB1T	ratio k	tolerance	5	.8	2.8	1498	1498.00	0.00	.241
UCB	ratio k	tolerance	5	0	2.8	1498	1498.00	0.00	.258
UCB1T	ratio k	tolerance	5	0	1.4	1498	1498.00		.362

**C.2.2 Solution no found**

Selec policy	Exp policy	Simu policy	N° chil- drens	Ratio	Cp	Best cost	Mean	Std	T(s)
UCB	ratio k	greedy	5	0	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	0	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	.3	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	.5	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	.8	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	1	0	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	1	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	greedy	5	1	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	0	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	0	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	0	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.3	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.8	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	1	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	1	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	greedy	5	1	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	0	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	0	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	0	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.3	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.3	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.3	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.5	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.5	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.5	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.8	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.8	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	.8	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	1	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	1	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	5	1	2.8	NaN	NaN	NaN	NaN

UCB	ratio k	random	10	0	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	0	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	0	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.3	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.3	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.3	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.5	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.5	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.5	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.8	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.8	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	.8	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	1	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	1	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	10	1	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	0	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	0	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	0	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.3	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.3	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.3	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.5	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.5	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.5	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.8	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.8	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	.8	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	1	0	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	1	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	random	15	1	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	0	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	0	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	0	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.3	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.5	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.8	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	.8	1.4	NaN	NaN	NaN	NaN



UCB1T	ratio k	random	5	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	1	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	1	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	5	1	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	0	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	0	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	0	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.3	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.5	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.8	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	1	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	1	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	10	1	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	0	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	0	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	0	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.3	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.5	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.8	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	1	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	1	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	random	15	1	2.8	NaN	NaN	NaN	NaN
UCB	ratio k	tolerance	5	1	0	NaN	NaN	NaN	NaN
UCB	ratio k	tolerance	5	1	1.4	NaN	NaN	NaN	NaN
UCB	ratio k	tolerance	5	1	2.8	NaN	NaN	NaN	NaN
UCB1T	ratio k	tolerance	5	1	0	NaN	NaN	NaN	NaN
UCB1T	ratio k	tolerance	5	1	1.4	NaN	NaN	NaN	NaN
UCB1T	ratio k	tolerance	5	1	2.8	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	0	0	NaN	NaN	NaN	NaN

UCB	top k	greedy	5	0	1.4	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	0	2.8	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.3	0	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.3	1.4	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.3	2.8	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.5	0	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.5	1.4	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.5	2.8	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.8	0	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.8	1.4	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	.8	2.8	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	1	0	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	1	1.4	NaN	NaN	NaN	NaN
UCB	top k	greedy	5	1	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	0	0	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	0	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	0	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.3	0	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.5	0	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.8	0	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	1	0	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	1	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	greedy	5	1	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	5	0	0	NaN	NaN	NaN	NaN
UCB	top k	random	5	0	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	5	0	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	5	.3	0	NaN	NaN	NaN	NaN
UCB	top k	random	5	.3	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	5	.3	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	5	.5	0	NaN	NaN	NaN	NaN
UCB	top k	random	5	.5	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	5	.5	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	5	.8	0	NaN	NaN	NaN	NaN
UCB	top k	random	5	.8	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	5	.8	2.8	NaN	NaN	NaN	NaN

UCB	top k	random	5	1	0	NaN	NaN	NaN	NaN
UCB	top k	random	5	1	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	5	1	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	10	0	0	NaN	NaN	NaN	NaN
UCB	top k	random	10	0	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	10	0	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	10	.3	0	NaN	NaN	NaN	NaN
UCB	top k	random	10	.3	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	10	.3	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	10	.5	0	NaN	NaN	NaN	NaN
UCB	top k	random	10	.5	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	10	.5	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	10	.8	0	NaN	NaN	NaN	NaN
UCB	top k	random	10	.8	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	10	.8	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	10	1	0	NaN	NaN	NaN	NaN
UCB	top k	random	10	1	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	10	1	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	15	0	0	NaN	NaN	NaN	NaN
UCB	top k	random	15	0	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	15	0	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	15	.3	0	NaN	NaN	NaN	NaN
UCB	top k	random	15	.3	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	15	.3	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	15	.5	0	NaN	NaN	NaN	NaN
UCB	top k	random	15	.5	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	15	.5	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	15	.8	0	NaN	NaN	NaN	NaN
UCB	top k	random	15	.8	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	15	.8	2.8	NaN	NaN	NaN	NaN
UCB	top k	random	15	1	0	NaN	NaN	NaN	NaN
UCB	top k	random	15	1	1.4	NaN	NaN	NaN	NaN
UCB	top k	random	15	1	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	0	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	0	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	0	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.3	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.5	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.5	1.4	NaN	NaN	NaN	NaN

UCB1T	top k	random	5	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.8	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	1	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	1	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	5	1	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	0	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	0	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	0	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.3	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.5	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.8	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	1	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	1	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	10	1	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	0	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	0	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	0	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.3	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.5	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.8	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	1	0	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	1	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	random	15	1	2.8	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	0	0	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	0	1.4	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	0	2.8	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.3	0	NaN	NaN	NaN	NaN

UCB	top k	tolerance	5	.3	1.4	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.3	2.8	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.5	0	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.5	1.4	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.5	2.8	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.8	0	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.8	1.4	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	.8	2.8	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	1	0	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	1	1.4	NaN	NaN	NaN	NaN
UCB	top k	tolerance	5	1	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	0	0	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	0	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	0	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.3	0	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.3	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.3	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.5	0	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.5	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.5	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.8	0	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.8	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	.8	2.8	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	1	0	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	1	1.4	NaN	NaN	NaN	NaN
UCB1T	top k	tolerance	5	1	2.8	NaN	NaN	NaN	NaN

**C.3 Instance 3**

Selec policy	Exp policy	Simu policy	N° chil- drens	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

## Appendix D

### 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', 'XSJ', '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#](https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/how-airlines-can-handle-busier-summers-and-comparatively-quiet-winters#/) /.
- [2] 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>.
- [3] 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.
- [4] 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.
- [5] 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>.
- [6] 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](https://doi.org/10.1016/S0305-0548(98)00019-7). URL <https://www.sciencedirect.com/science/article/pii/S0305054898000197>.

- [7] Deirdre Fulton. Unstoppable lccs - growth indicates a new norm. <https://www.oag.com/blog/unstoppable-lccs-growth-indicates-new-norm>, 2023.
- [8] 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=4T0soX6aPiA>. Accessed: 2024-07-05.
- [9] 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.
- [10] 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>.
- [11] 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.
- [12] 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/flight>, 2024.
- [13] Not mentionned. What is acmi leasing? ACC Aviation, 2024.
- [14] Lark Editorial Team. Np hard definition of np hardness. *Lark*, 26 December, 2023.
- [15] 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](https://doi.org/10.1016/0167-6377(83)90048-2). URL <https://www.sciencedirect.com/science/article/pii/0167637783900482>.
- [16] 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.2005.12.001>.

- [//doi.org/10.1016/j.omega.2004.10.004](https://doi.org/10.1016/j.omega.2004.10.004). URL <https://www.sciencedirect.com/science/article/pii/S0305048304001550>.
- [17] 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>.
- [18] 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>.
- [19] 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>.
- [20] Aviv Adler. The traveling salesman problem under dynamic constraints. *Massachusetts Institute of Technology*, Feb 2023.
- [21] 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>.
- [22] 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.
- [23] 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.
- [24] 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](https://doi.org/10.1016/0377-2217(85)90151-1). URL <https://www.sciencedirect.com/science/article/pii/0377221785901511>.
- [25] Not specified. Travelling salesman problem using dynamic programming. *Geeks-forgeeks*, 19 April, 2023.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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>.
- [30] Wikipedia. Havannah (board game) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Havannah%20\(board%20game\)&oldid=1240631485](http://en.wikipedia.org/w/index.php?title=Havannah%20(board%20game)&oldid=1240631485), 2024. [Online; accessed 18-August-2024].
- [31] 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].
- [32] 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].
- [33] Wikipedia. Shogi — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Shogi&oldid=1240175752>, 2024. [Online; accessed 18-August-2024].



- [34] 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.
- [35] 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.
- [36] 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.
- [37] Wikipedia. Go (game) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Go%20\(game\)&oldid=1239511822](http://en.wikipedia.org/w/index.php?title=Go%20(game)&oldid=1239511822), 2024. [Online; accessed 18-July-2024].
- [38] 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].
- [39] Google DeepMind. Alphago - the movie / full award-winning documentary. Youtube, 2020.
- [40] Not mentionned. Explain the role of monte carlo tree search (mcts) in alphago and how it integrates with policy and value networks. EITCA, 2024.
- [41] 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.
- [42] 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=UXW2yZnd17U&t=385s>. Accessed: June, 2024.
- [43] 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.
- [44] Kiwi.com. Travelling salesman challenge 2.0: Award ceremony. Youtube, 2019. URL <https://www.youtube.com/watch?v=Fp7LaEUwCjE>.
- [45] Maab Alrasheed, Wafaa Mohammed, Yaroslav Pylyavskyy, and Ahmed Kheiri. Local search heuristic for the optimisation of flight connections. pages 1–4, 09 2019. doi: 10.1109/ICCCEEE46830.2019.9071395.
- [46] Rowaina Abdelnasser. Python naming conventions: 10 essential guidelines for clean and readable code.