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]

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. The goal is to provide insight into some important optimisation problems, underscoring their importance for the success and competitiveness of airline companies.

2.1.1 Fleet Assignment Problem

The Fleet Assignment Problem (FAP), as discussed in [1] involves assigning different types of aircraft, each with specific capacities, 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 [2], 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 carriers, for instance in the United Kingdom in 2023, the low-cost carriers comprise 48% of the scheduled capacity (total number of seats offered) [3], 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. [4]

2.1.3 Disruption Management

Disruptions in airline operations, as noted in [5], 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 [6], effective disruption management is crucial to minimise the impact on passengers and overall airline operations.

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

- Aircraft recovery: Reassigning aircraft to flights after disruption by taking care of schedules, airports availibility and maintenance. Optimisation tools help manage the complex logistics of matching available aircraft with rescheduled flights, considering factors like airport availability and maintenance requirements. This ensures that the revised schedules are as efficient and practical as possible.
- Crew recovery: reassigning crew members to flights by respecting all flights are staffed appropriattely. 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 [7] and [8], are crucial for reducing the impact of disruptions and boosting operational resilience in the airline industry.

2.1.4 Airline adaptation to new demand

Airlines 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 seasonality, especially in Europe as shown on Figure 2.1. 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.

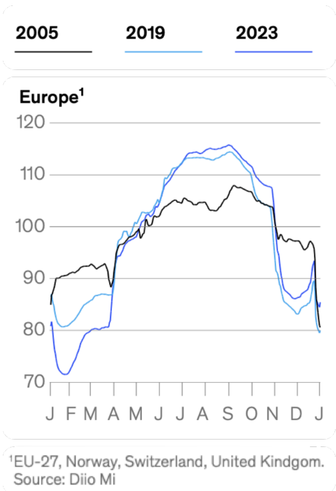


FIGURE 2.1: European demand seasonality [9]

Since travel demand varies throughout the year, seasonal-wise, airlines use a variety of techniques to achieve operational efficiency while maximising revenue [9]. The seats that airlines book are nearly 65% more in August than they do in February when summer peak demand skyrocketed. They 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 to have everything go smoothly due to the heightened demand of resources.

In contrast, winter months pose a different type of problem: demand drops, meaning potential underutilisation of aircraft. To do 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 [10]) during periods of low demand to temporarily reduce fleet size by outsourcing their capacity. Parallel to 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 and improve overall aircraft utilisation. Such adaptive strategies are critical to the airlines for effectively beating the seasonal ebbs and flows in the travel industry.

2.2 Traveling Salesman problem and its adapation

The Traveling Salesman Problem is a well known problem in the Operational Research and Computer Science area. The basic version of the TSP is to find the best roundtrip for a saleman that has to travel around a given number of cities while minimising the overall journey's distance. This problem is characterised as $\mathcal{NP}$-Hard [11]. This means that there is no known polynomial-time algorithm that can solve all instances of the problem efficiently . In terms of time complexity, if we were to solve it explorating all the possible solutions the time complexity would have been $\mathcal{O}(\frac{(n-1)!}{2})$ where n represents the number of cities.

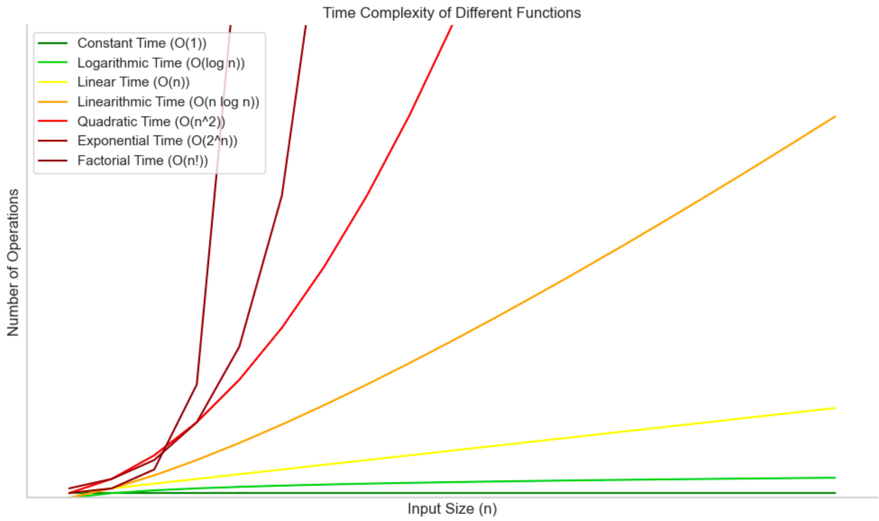


FIGURE 2.2: Time complexity of different functions [12]

On Figure 2.2, different time complexity are compared and the factorial time complexity is the worst. That means that 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 in a much more reasonnable times.

The TSP has been studied extensively, not least because many variants can be derived from it:

- Symmetric TSP (STSP): The distance between cities are symmetric, meaning that the distance to travel from city A to city B is the same as from city B to city A.

- Assymetric TSP (ATSP): The distance between cities are assymetric, meaning that the distance to travel from city A to city B is different than the distance to travel from city B to city A.[13]
- Multiple TSP (mTSP): Instead of one salesman, multiple salesman are starting from one city visit all the cities such that each city is visited exactly once. [14]
- Time Window TSP (TWTSP): Each city has to be visited in a defined time slot. [15]
- Price-collection TSP (PCTSP): Not all the cities have to be visited, the goal is to to minimise the overall traveler's distance while maximising the price collected earned when visiting a city. [16]
- Stochastic TSP (STSP): The distances between the cities or the cost of travels are stochastic (ı.e random variables) rather than deterministic. [17]
- Dynamic TSP (DTSP): The problem can change over the times, that means that new cities can be added or distances between cities can change while the salesman has already started his journey. [18]
- Generalised TSP (GTSP): The cities are grouped into clusters, the goal is to visit exactly one city from each cluster. [19]
- Open TSP (OTSP): The traveler does not have to end his journey at the starting city. [20]

Multiple algorithms have been developed to address these TSP variants, we can classify them into two major 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. [21]
  - **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. [22]

- **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. [23]

- **Approximation and Heuristic Algorithms**: These algorithms are designed to find near-optimal solutions within a reasonable time frame, especially 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. [24]
  - **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. [25]
  - **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. [26]
  - **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, gradually improving them through pheromone updates. [27]

Some TSP problems (or its variants) have been solved using other algorithms, hence they are less traditionnal than those mentionned.

2.3 The Monte Carlo Tree Search algorithm

The Monte Carlo Tree Search (MCTS) algorithm can be characterised less traditionnal than the previously enounced method in Section 2.2 to solve TSP because MCTS is typically used in games. MCTS' (and its variants) have been successfully implemented across a range of games, such as Havannah [28], Amazons [29], Lines of Actions [30], Go, chess, and Shogi [31], establishing it as the state-of-the-art method for many of these games [32], [33], [34]. It is widely use in board games and has been really popular when Google DeepMind developed AlphaGo. AlphaGo is a software that was created to beat the best Go's player in the world.

Go is an ancient 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 incredibly deep and complex, with countless possible moves and strategies. It is known for its balance between intuition and logic, hence it has been a significant focus of artificial intelligence research ,[35]. In 2016, Lee Sedol [36] - the best Go's player in the world has been beaten by AlphaGo 4-1 [37].

MCTS with policy and value networks are at the heart of AlphaGo's decision-making process, enabling AlphaGo's to pick the optimal moves in the complex search of Go. [38]

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 we have to select in the tree to perform simulations. The selection policy, typically balances the exploration i.e. look into part of the tree that have not been visited yet and the exploitation i.e. look into part of the trees that appear to be promising. Once the node is selected, a simulation i.e. a sequence of available actions, based on a simulation policy, are applied from this node until a terminal condition is reached e.g no further actions are possible. [39]

To ensure that the reader understands the various stages of the Monte Carlo Tree Search Algorithm, we will begin by looking at a detailed example. This example will illustrate

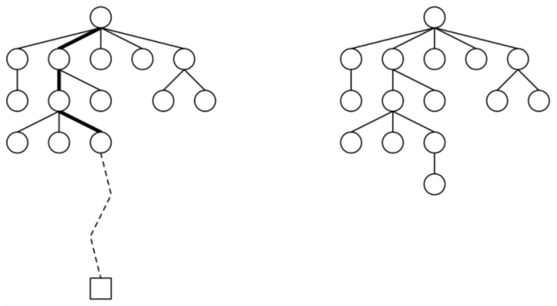


FIGURE 2.3: Assymetrical growth of MCTS - Simulation and Expansion - [40]

each component of the algorithm in action. We will then generalise the principles discussed, as the basic methodology of this paper is based on the application of the MCTS algorithm.

2.3.2 Example

Let's say we are given a maximisation problem. When starting 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. Furthermore, for every node - we can compute a selection metric, for instance the $UCB1$ value: $UCB1(S_i^{n_i,t_i}) = \bar{V}_i + 2\sqrt{\frac{\ln N}{n_i}}$ 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

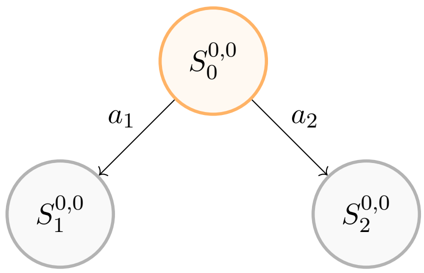


FIGURE 2.4: Selection - $I1$

of $I1$, we then have to choose between these two child nodes (or choose between taking a_1 or a_2). We then have to calculate the $UCB1$ value for these two nodes and pick the node that maximises the $UCB1$ value (as we are dealing with a maximisation problem).

In Figure 2.4, neither of these have been visited yet so $USB(S_1^{0,0}) = UCB1(S_2^{0,0}) = \infty$. 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 i.e. selecting actions from this node based on the simulation policy to a terminal state as shown on Figure 2.5:

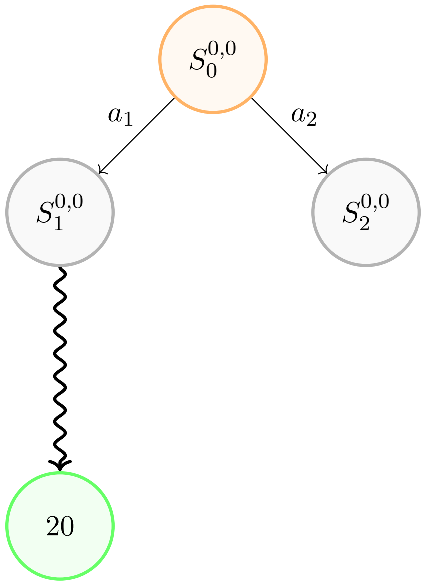


Figure 2.5: Simulation - $I1$

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

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

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

We can then define a backpropagation function:

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

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

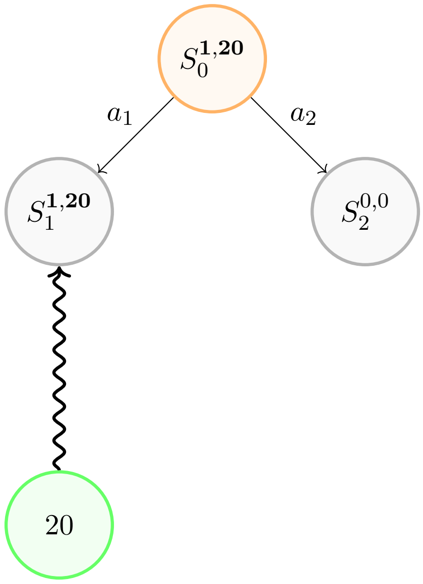
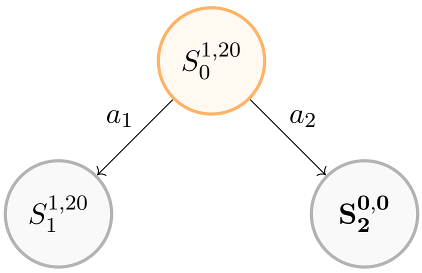


FIGURE 2.6: Backpropagation - I1

The fourth phase of the algorithm have been done for $I1$. 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 has not



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

FIGURE 2.7: Selection - I2

been visited yet, you pick this node for the Selection or you can compute the $UCB1$ value, it leads to the same conclusion.

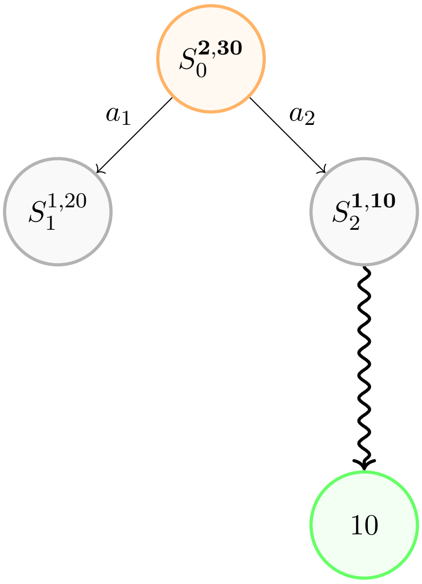


FIGURE 2.8: Simulation and Backpropagation - I2

We can then simulate (Figure 2.8) from the chosen node $S_2^{0,0}$ and $\mathcal{R}(S_2^{0,0}) = 10$ and backpropagate all the visited nodes: $\mathcal{B}(S_2^{0,0}) = S_2^{1,10}$ and $\mathcal{B}(S_0^{1,20}) = S_0^{2,30}$. We now start the 3^{rd} iteration, based on the $UCB1$ score we decide to choose a_1.

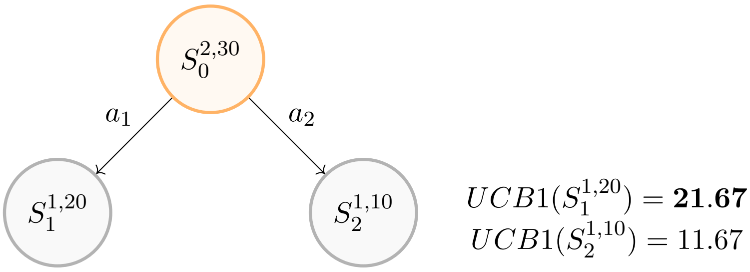


FIGURE 2.9: Selection - I3

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

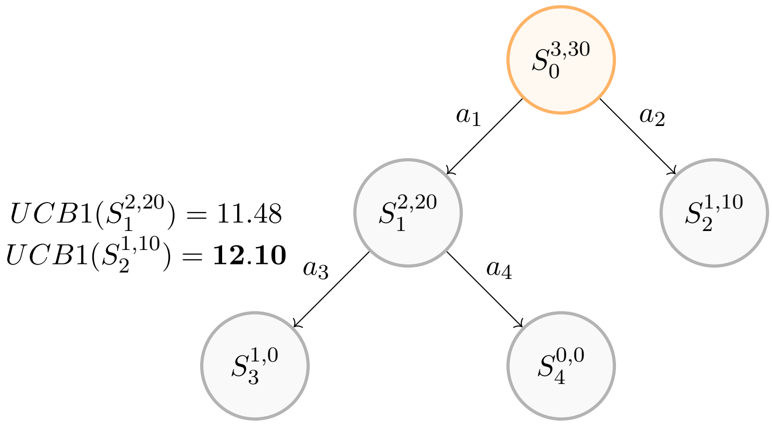


FIGURE 2.10: Selection and Expansion - I3

Based on $UCB1$ score we decide to simulate from $S_3^{0,0}$ on Figure 2.11

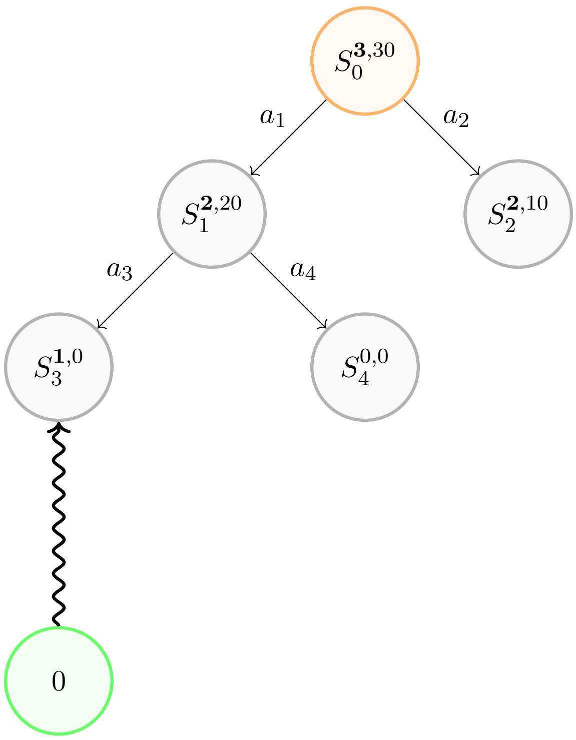


FIGURE 2.11: Simulation and Backpropagation - I3

Let's do the fourth iteration $I4$ represented on Figure 2.12:

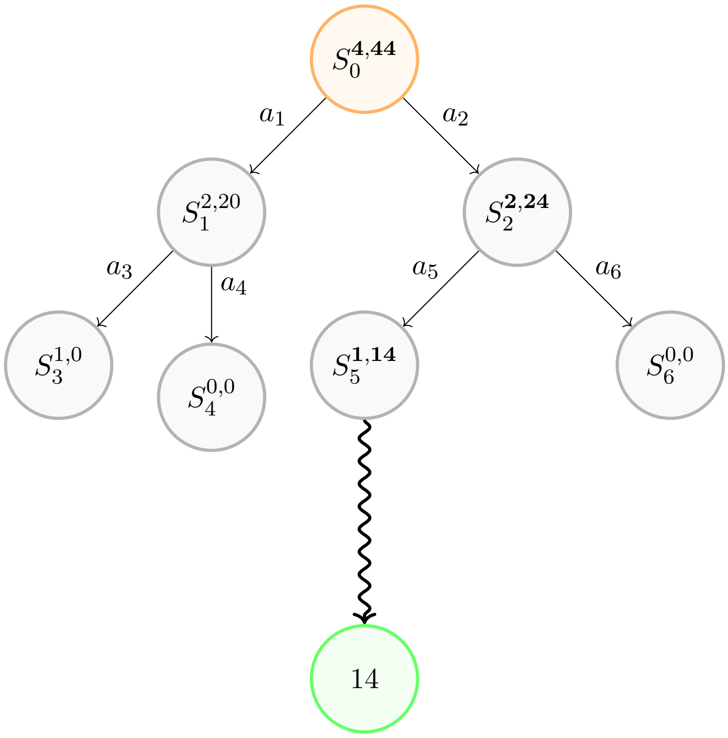


FIGURE 2.12: Selection - Simulation - Backpropagation - I4

The MCTS algorithm can be either 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

2.3.4 Selection policy

[41]

- **Upper Confidence Bound (UCB):**
  - *Description*: UCB is a popular selection policy in MCTS that balances exploration and exploitation using a mathematical formulation that considers both the average reward of a node and the uncertainty of that reward.
  - *Trade-offs*:

    * **Exploration vs. Exploitation**: UCB adjusts the balance between exploring less-visited nodes and exploiting nodes with high rewards.
    * **Parameter Sensitivity**: The performance of UCB depends on the constant C_p in its formula, which controls the level of exploration.
    * **Children's Count**: UCB can lead to varying numbers of children being explored, depending on the setting of C_p.

- **UCB1-Tuned:**
  - *Description*: A variation of UCB that dynamically adjusts the exploration term based on the variance of rewards, making it more adaptive to different scenarios.
  - *Trade-offs*:
    * **Adaptivity**: UCB1-Tuned can adapt to the variance in the rewards, potentially leading to better performance in complex environments.
    * **Computational Cost**: The added complexity in adjusting the exploration term may lead to higher computational costs.
    * **Children's Count**: This policy may result in fewer nodes being selected for expansion when the variance is low, focusing more on exploitation.
- **Single-Player (SP-MCTS):**
  - *Description*: A variant of MCTS specifically designed for single-player scenarios, incorporating a third term in the UCB formula to account for the uncertainty in node values.
  - *Trade-offs*:
    * **Exploration of Uncertainty**: SP-MCTS inflates the uncertainty term for less-visited nodes, which can lead to more thorough exploration in single-player settings.
    * **Focus on Strong Lines**: This policy tends to favor strong lines of play, potentially neglecting less promising but necessary exploratory paths.
    * **Children's Count**: Tends to explore a wider range of children nodes, balancing between known strong strategies and potential new solutions.
- **Bayesian UCT:**
  - *Description*: Bayesian UCT integrates Bayesian statistics into the selection policy, allowing for a probabilistic approach to balancing exploration and exploitation.

- *Trade-offs*:
  * **Probabilistic Exploration**: Bayesian UCT provides a more nuanced exploration strategy by considering prior knowledge and updating beliefs as more information is gathered.
  * **Complexity**: The Bayesian approach can be computationally expensive, especially in large search spaces.
  * **Children's Count**: The number of children explored under Bayesian UCT can be influenced by the prior distributions used, potentially leading to more focused exploration in areas of high uncertainty.

2.3.5 Simulation policy

2.3.5.1 Different nodes

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 necessary the starting airport. There are flight connections between different airports, with different prices depending on the day of the travel: we can write c_{ij}^d the cost to travel from $city_i$ to $city_j$ on day d. We do not necessarly have $c_{ij}^d = c_{ji}^d$ neither $c_{ij}^{d_1} = c_{ij}^{d_2}$ if $d_1 \neq d_2$. The problem can hence be characterised as assymetric and time dependant as discussed in Section 2.2.

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

We can then formulate the problem more effectively:

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

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

Parameters

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

Variables

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

Constraints

1. Starting and Ending Constraints:

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

2. Flow Constraints:

   - The traveler must leave each area and arrive at the next area on consecutive days, the next area has not been visited yet.
   - Ensure that the traveler can only fly into and out of the same airport within an area.
   - Ensure each area is visited exactly once.
   - Update the unvisited list as areas are visited.

Objective Function

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

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

Constraints

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

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

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

- Visit exactly one airport in each area each day:

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

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

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

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

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

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

- Ensure each area is visited exactly once:

$$\sum_{d \in \mathcal{D}} v_j^d = 1 \quad \forall j \in \mathcal{A}$$

- Update the unvisited list:

$$v_j^d = 1 \implies j \notin U_d \quad \forall j \in \mathcal{A}, \forall d \in \mathcal{D}$$

- Ensure a flight on day d between i and j exists only if the cost exists and j is in the unvisited areas on day d:

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

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

- Binary variable constraints:

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

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

3.2 Instances

3.2.1 Description

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

For example, the first few lines of I_4 are:

13 GDN
first
WRO DL1
second
BZG KJ1
third
BXP LB1

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

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

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

TABLE 3.1: Flight connections sample I6

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 same flight connections at a specific day but with different prices. We then have to consider only the more relevant connections i.e. the flight connection with the lowest fare.

3.2.2 General formulation

An instance I_i can be mathematically defined as follows:

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

where:

- **Number of Areas** N_i:

  $$N_i \in \mathbb{N}$$

  The total number of distinct areas in instance I_i.

- **Starting Airport** S_{i0}:

$$S_{i0} \in \text{Airports}$$

The starting airport of the traveller.

- **Airports in Each Area**:

$$A_i = \{A_{i,1}, A_{i,2}, \dots A_{i,N_i}\}$$

where each $A_{i,j}$ is a set of airports in area j for instance i:

$$A_{i,j} = \{a_{i,j,1}, a_{i,j,2}, \dots, a_{i,j,k_j}\}$$

with a_{i,j,k_j} being airports in area j and k_j is the number of airports in area j.

- **Flight Connections**:

$$F_i = \{F_{i,0}, F_{i,1}, F_{i,2}, \dots, F_{i,N_i}\}$$

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

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

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

3.2.3 Kiwi's rules

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

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

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

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 defined in Table 3.2.

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

TABLE 3.3: Instances and their respective parameters

Chapter 4

Methodology

(TODO) Describe implementation details In this section we are going to discuss the different classes we used to implmeent our solution.

We have used Python's latest version 3.10 on VS Code.

4.1 Monte Carlo Tree Search

4.1.1 Generalisation

The Monte Carlo Tree Search algorithm can be summarised on Figure 4.1:

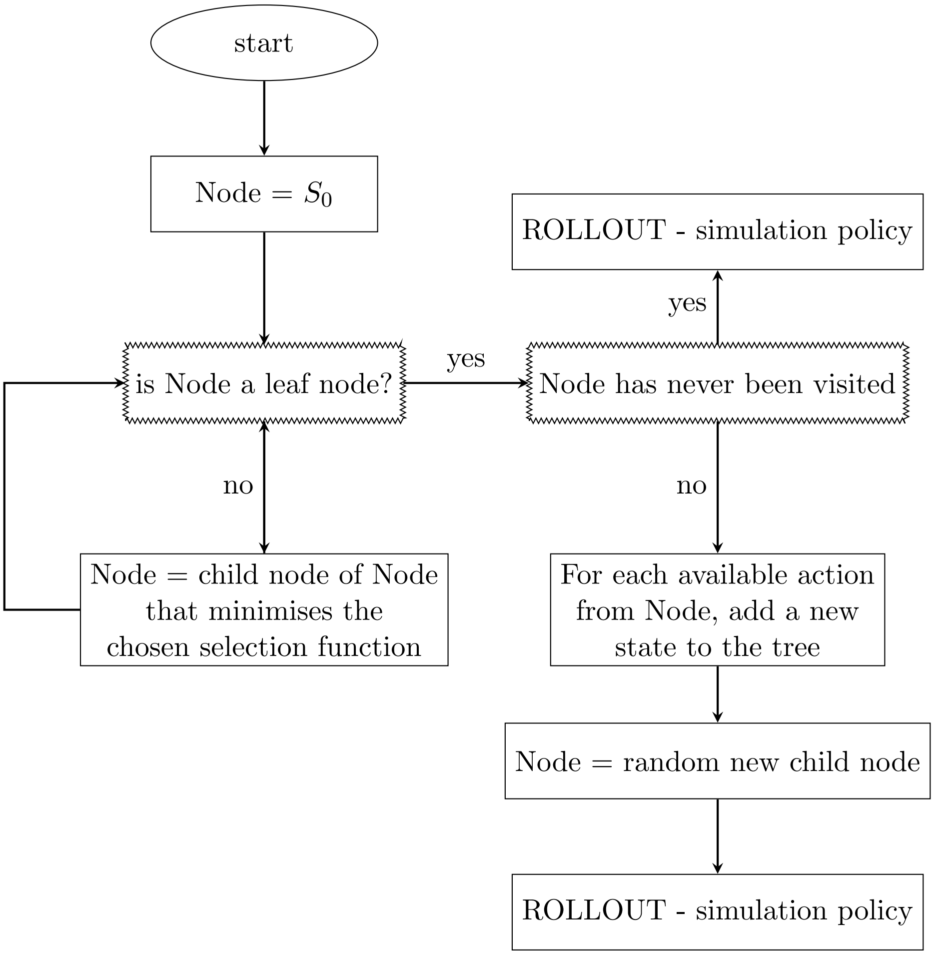


FIGURE 4.1: Flow MCTS

In 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 necessary the starting airport) is reached. Use the Upper Confidence Bound for Trees (UCB1) formula to balance exploration and exploitation.

$$UCB1(S_i^{n_i,t_i}) = \bar{V}_i + c\sqrt{\frac{\ln N}{n_i}} \tag{4.1}$$

where:

- $\bar{V}_i = \frac{t_i}{n_i}$ is the average value of the node.
- c is the exploration parameter - theoretically equal to $\sqrt{2}$; in practice it is chosen empirically.
- n_i is the number of times node i has been visited.
- N is the total number of visits for the root node.

As oppose to the example in Section 2.3.2, here we are going to select node with the lowest $UCB1$ value because we want to minimise the overall traveler's cost.

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 (for example take random available flights from these nodes to a terminal node).

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

$$\mathcal{B}(S_i^{n_i,t_i}) = S_i^{n_i+1,t_i+\mathcal{R}(S_i^{n_i,t_i})} \tag{4.2}$$

where $\mathcal{R}(S_i^{n_i,t_i})$ is the result of the simulation starting from node $S_i^{n_i,t_i}$.

4.1.1.1 Data Preprocessing

In order to implement our solution, the first thing to implement was a data_preprocessing class. All our Python code is oriented-object programmed. We have represented our class on Figure 4.2. The input is an instance I_i, as defined in Chapter 3:

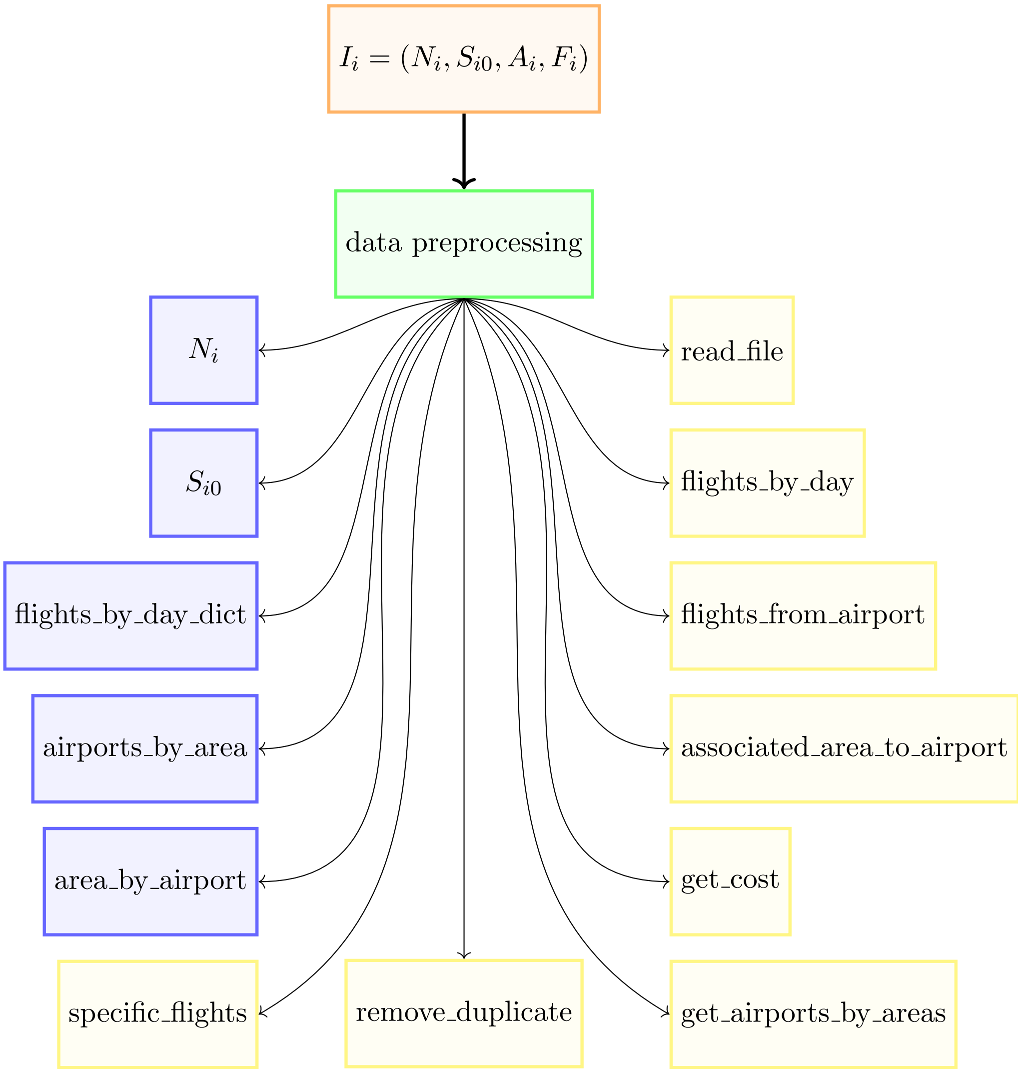


FIGURE 4.2: Explanation of the data preprocessing class

Different useful methods are implemented to compute data preprocessing attributes. For example, remove_duplicate considers the cheapest flight connections if multiple flight connections between two airports exist at different prices on the same day. Thanks to the different methods we can then compute data preprocessing's attributes such as flights_by_day_dict that group all the flights by day, airports_by_area group all the airports per area. Other methods are also defined like specific_flifhts that will be helpful later and gives you all the possible flight connections from a specific airport on a specific day considering the visited_areas, it hence gives you all the possibles actions from a node.

It is known that Python is relatively slow in terms of computation, so we decided to use as much as possible hasmaps. Hashmaps allow to retrieve data efficiently based on a key - with a $\mathcal{O}(1)$ in term of time complexity.

4.1.1.2 Node

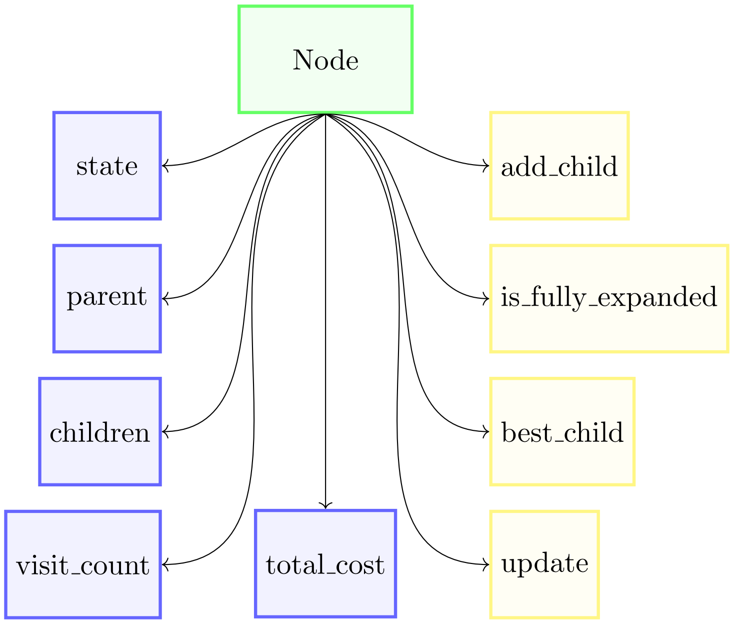


FIGURE 4.3: Explanation of the Node class

As already mentionned during the example, we use Node structure in our algorithm, we hence defined a Node class. A node has one parent (if it is not the root node) or children(s) (if it is not a leaf node), they also have a number of times they have been visited - visit_count and the total_cost. We also add a state which is a dictionnary where we keep track of the current_airport and the current_day, the remaining_zones we have to visit from this node to end the traveler's journey, the visited_zones so far, and the total_cost of the undertaken flight connections that is going to evolve for the simulation and then be backpropagated to the total_cost of the node if a terminal node is reached.

4.1.2 Pseudo-code

Algorithm 1 Monte Carlo Tree Search for Minimisation Problem

```
function MonteCarloTreeSearch(root)
    root.visitedCount ← 0
    root.totalValue ← 0
    repeat
        node ← Selection(root)
        reward ← Simulation(node) Backpropagation(node, reward)
    until termination condition is met
    return root
end function

function Selection(node)
    while node is not a leaf do
        if node.children are not fully expanded then
            node ← Expansion(node)
        else
            node ← arg max_{child∈node.children} UCB1(child)
        end if
    end while
    return node
end function

function Expansion(node)
    Create new child nodes for each possible action from node
    node ← first new child
    return node
end function

function Simulation(node)
    while node is not a terminal state do
        action ← select_action(node)
        node ← perform_action(node, action)
    end while
    reward ← calculate_total_cost(node)
    return reward
end function

function Backpropagation(node, reward)
    while node is not null do
        node.visitedCount ← node.visitedCount + 1
        node.totalValue ← node.totalValue + reward
        node ← node.parent
    end while
end function

function UCB1(node, c)
    root_node ← node.parent.visitedCount
    value ← node.totalValue / node.visitedCount
    ucbValue ← value + c * sqrt( ln(parentVisits) / node.visitedCount )
    return ucbValue
end function
```

4.1.3 Different policies

4.1.3.1 Selection policy

Our selection policy is based on

4.1.3.2 Simulation policy

Algorithm 1 Monte Carlo Tree Search (MCTS)

```
Initialize root node with initial state
for number of simulations do
    node ← Select(root)
    if node is not fully expanded then
        node ← Expand(node)
    end if
    cost ← Simulate(node)
    Backpropagate(node, cost)
end for
return BestLeafNode
```

Algorithm 2 Select Function

```
currentNode ← root
while True do
    unvisitedChildren ← GetUnvisitedChildren(currentNode)
    if unvisitedChildren is not empty then
        return Randomly select from unvisitedChildren
    end if
    if currentNode has no children then
        ExpandNode(currentNode)
        if currentNode has no children after expansion then
            return None
        end if
        return Randomly select from currentNode.children
    end if
    currentNode ← BestChild(currentNode)
end while
```

Algorithm 3 Expand Function

```
actions ← PossibleActions(currentNode.state)
expansionPolicy ← GetExpansionPolicy()
actions ← expansionPolicy(actions)
for each action in actions do
    newState ← TransitionFunction(currentNode.state, action)
    AddChild(currentNode, newState)
end for
```

Algorithm 4 Simulate Function

```
currentSimulationState ← Copy of node.state
while currentSimulationState.currentDay ≠ Number of Areas do
    actions ← PossibleActions(currentSimulationState)
    action ← SimulationPolicy(actions)
    if action is None then
        return False
    end if
    currentSimulationState ← TransitionFunction(currentSimulationState, action)
end while
if currentSimulationState.currentDay == Number of Areas then
    returnFlightActions ← FlightsToReturn(currentSimulationState)
    if returnFlightActions is empty then
        return False
    end if
    action ← SimulationPolicy(returnFlightActions)
    currentSimulationState ← TransitionFunction(currentSimulationState, action)
end if
return currentSimulationState.totalCost
```

Algorithm 5 Backpropagate Function

```
while node is not None do
    node.update(cost)
    node ← node.parent
end while
```

Algorithm 6 Transition Function

$newState \leftarrow$ Copy of $state$
$newState.currentDay \leftarrow state.currentDay + 1$
$newState.currentAirport \leftarrow action[0]$
$newState.totalCost \leftarrow state.totalCost + action[1]$
Update($newState.path$, $newState.currentAirport$)
RemoveVisitedZone($newState.remainingZones$, $newState.currentAirport$)
AddVisitedZone($newState.visitedZones$, $newState.currentAirport$)
return $newState$

4.1.4 selection policies

Algorithm 7 UCB Selection

$\epsilon \leftarrow 0$
$visitedChildren \leftarrow$ Children with $visitCount > 0$
$sortedChildren \leftarrow$ Sort $visitedChildren$ by $\frac{totalCost}{visitCount+\epsilon}$
$scores \leftarrow$ Assign ranks to sortedChildren
$totalScores \leftarrow \sum scores$
$normalizedScore(child) \leftarrow \frac{scores[child]}{totalScores}$
$choicesWeights \leftarrow \Big[normalizedScore(child) + cParam \times \sqrt{\frac{2\times\log(visitCount)}{child.visitCount+\epsilon}}$ for each child in $visi$
$bestChildNode \leftarrow$ Child with minimum $choicesWeights$
return $bestChildNode$

Algorithm 8 SP Selection

$visitedChildren \leftarrow$ Children with $visitCount > 0$
$D \leftarrow 1$
$spMctsScore(child) \leftarrow$ meanCost $- \, cp \times$ possibleDeviation
$choicesWeights \leftarrow [spMctsScore(child)$ for each child in $visitedChildren]$
$bestChildNode \leftarrow$ Child with minimum $choicesWeights$
return $bestChildNode$

Algorithm 9 Bayesian UCT Selection

$visitedChildren \leftarrow$ Children with $visitCount > 0$
$N \leftarrow visitCount$
$useVariance \leftarrow$ True or False depending on the formula
$bayesianUctScore(child) \leftarrow meanCost + explorationTerm$
$choicesWeights \leftarrow [bayesianUctScore(child, useVariance)$ for each child in $visitedChildren]$

$bestChildNode \leftarrow$ Child with minimum $choicesWeights$
return $bestChildNode$

Algorithm 10 UCB1-Tuned Selection

$visitedChildren \leftarrow$ Children with $visitCount > 0$
$ucb1TunedScore(child) \leftarrow meanCost + cParam \times \sqrt{\left(\frac{\log(visitCount)}{child.visitCount}\right) \times \min\left(0.25, variance + \sqrt{\frac{2\times\log(visitCount)}{child.visitCount}}\right)}$
$choicesWeights \leftarrow [ucb1TunedScore(child)$ for each child in $visitedChildren]$
$bestChildNode \leftarrow$ Child with minimum $choicesWeights$
return $bestChildNode$

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

- Based on the different rollout policy, analyse the output -¿ the greey is deterministic whereqs the 2 others are stochastic
- Maybe try to fine tune the selection policy -¿ one idea could be instead of returning the average score is to return the min score
- Compare outputs with known solutions - Yaro + litterature -¿ efficiency in terms of time
- Ahmed - litterature : https://code.kiwi.com/articles/travelling-salesman-challenge-2-0-wrap-up/

Chapter 6

Conclusion

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

6.1 Summary of Work

6.2 Critics

6.3 Future Work

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

Chapter 7

Progress and next steps

Bibliography

[1] 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.

[2] J.E. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers and Operations Research*, 25(7):567–582, 1998. ISSN 0305-0548. doi: https://doi.org/10.1016/S0305-0548(98)00019-7. URL https://www.sciencedirect.com/science/article/pii/S0305054898000197.

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

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

[5] 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.

[6] 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.

[7] None. 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.

[8] None. A modern cloud platform to optimize end-to-end airline operations and crew management. iflight drives unmatched efficiencies, cost-savings, and productivity for the world's top airlines. https://www.ibsplc.com/product/airline-operations-solutions/iflight, 2024.

[9] 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#/.

[10] None. What is acmi leasing? ACC Aviation, 2024.

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

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

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

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

[26] 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.

[27] 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`.

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

[29] 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].

[30] 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].

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

[32] 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.

[33] 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.

[34] 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.

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

[36] 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].

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

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

[39] 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.

[40] 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.

[41] 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.