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

Chapter 1

Literature Review

1.1 What is Optimization?

1.1.1 Definition and Concept

Discuss the general concept of optimization, emphasizing its role in operations research and decision-making, and explaining how it aims to find the most favorable solution from available alternatives.

1.1.2 Exact vs. Heuristic Methods

Exact Methods: Describe methods that provide guaranteed optimal solutions, such as linear programming and integer programming. **Heuristic Methods:** Introduce heuristic approaches, like genetic algorithms and simulated annealing, used in complex problems where exact solutions are computationally impractical. **Comparison:** Analyze the trade-offs between exactness and computational efficiency.

1.1.3 Day-to-Day Optimization Examples

Illustrate how optimization is applied in everyday scenarios such as route planning, airline scheduling, and personal time management.

1.2 TSP and its variants

1.3 Monte Carlo Tree Search

1.3.1 Markov Chains

Introduce the theory of Markov Chains and their application in modeling stochastic processes and optimization.

1.3.2 Monte Carlo Methods

Discuss the basics of Monte Carlo methods, focusing on their use in solving complex computational problems through random sampling.

Detailed discussion on Monte Carlo Tree Search (MCTS), explaining how it extends Monte Carlo methods to decision-making problems, particularly highlighting its relevance in AI and game theory. Do different part to highlight the differents plociies, the different parameters - how it has been used especially in game theory and so on

1.4 Review of Previous Approaches to the Kiwi.com Problem

We do not explain the problem here - or shall we?

1.4.1 Survey of Approaches

Discuss main approaches like Local Search Algorithms, Reinforcement Learning, and Genetic Algorithms, mentioning key studies that have tackled similar challenges. **Local Search Algorithms:** Explore the principles and applications of Local Search, highlighting specific papers and their results. **Reinforcement Learning:** Review the application of Reinforcement Learning in transportation optimization, citing relevant studies. **Genetic Algorithms:** Describe how Genetic Algorithms have been used to solve complex routing problems and their effectiveness in various scenarios.

1.4.2 Comparative Analysis

Provide a comparative analysis of these methods with Monte Carlo Tree Search, evaluating performance under different conditions based on the nature of the problem.

1.5 Research Motivation and Literature Gaps

1.5.1 Motivation for This Study

Discuss the motivations for using MCTS in solving the Kiwi.com problem, considering the problem's complexity and scale.

1.5.2 Identification of Gaps

Highlight areas not fully explored by previous studies, such as scalability or real-time application constraints, emphasizing the novel aspects of your research.

1.6 Critical Analysis of Existing Research

Analyze the strengths and weaknesses of the studies reviewed, providing a critical perspective in relation to your research. Discuss how convincing the authors' arguments are and where your research might offer improved or alternative solutions.

(TODO) Further refine and expand sections based on feedback and ongoing research.

Chapter 2

Problem Description

2.1 Overview

Kiwi's traveler wants to travel in N different areas in N days, let's denote A the set of areas the traveler wants to visit:

$$A = \{A_1, A_2, \dots A_N\}$$

where each A_j is a set of airports in area j:

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

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

The traveler has to visit one area per day. He has to leave this area to visit a new area by flying from the airport he flew in. He leaves from a known starting airport and has to do his journey and come back to the starting area, not necessarly the starting airport. There are flight connections between different airports, with different prices depending on the day of the travel: we can write c_{ij}^d the cost to travel from $city_i$ to $city_j$ on day d. We do not necessarly have $c_{ij}^d = c_{ji}^d$ neither $c_{ij}^{d_1} = c_{ij}^{d_2}$ if $d_1 \neq d_2$. The problem can hence be characterised as assymetric and time dependant.

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:

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

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

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

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

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

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

- Ensure each area is visited exactly once:

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

- Update the unvisited list:

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

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

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

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

- Binary variable constraints:

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

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

2.2 Instances

2.2.1 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 2.1, 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 2.1: 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.

2.2.2 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**:

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

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

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

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

- **Flight Connections**:

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

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.

2.2.3 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 2.2: 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 2.2.

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 2.3: Instances and their respective parameters

Chapter 3

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.

3.1 Monte Carlo Tree Search

In this section, we are first going to dive into an example to explain the Monte Carlo Tree Search algorithm with a simple case. In the second part we are going to generalise it to our problem.

3.1.1 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 $S_0^{0,0}$ 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 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/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

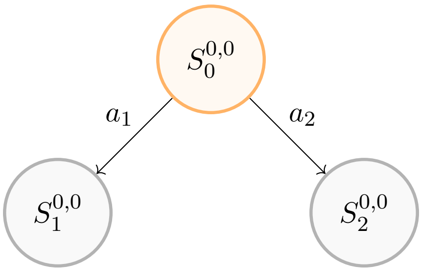


FIGURE 3.1: Selection - $I1$

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 3.1, 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. taking random actions from this node to a terminal state as shown on Figure 3.2:

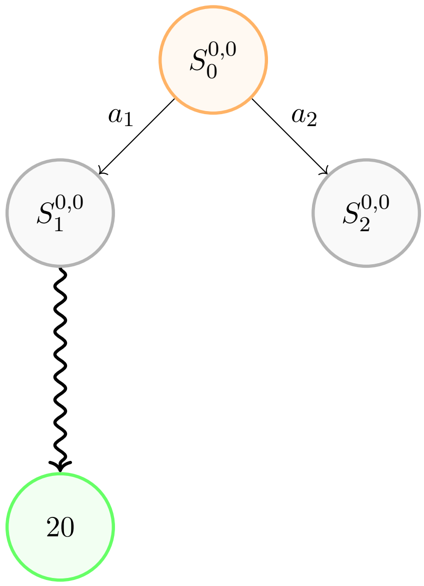


FIGURE 3.2: Simulation - $I1$

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:

$$\forall i \in \mathcal{N}_{\mathcal{R},j}, S_{i,old}^{n_i,t_i} \tag{3.1}$$

- After backpropagation:

$$\forall i \in \mathcal{N}_{\mathcal{R},j}, S_{i,new}^{n_i+1,t_i+\mathcal{R}(S_{i,old}^{n_i,t_i})} \tag{3.2}$$

We can then define a Backpropagate function:

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

Then, back to our example on Figure 3.3 we update the nodes $\mathcal{B}(S_1^{0,0}) = S_1^{\mathbf{1,20}}$ and $\mathcal{B}(S_0^{0,0}) = S_0^{\mathbf{1,20}}$.

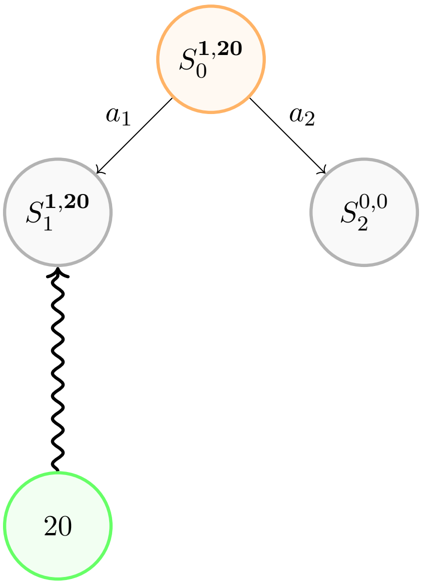
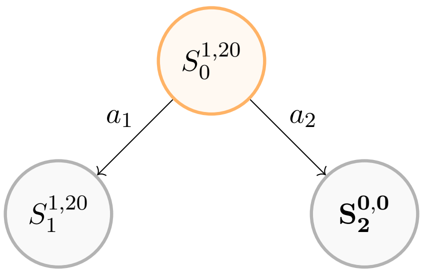


FIGURE 3.3: Backpropagation - I1

The fourth phase of the algorithm have been done for $I1$. We can then start the 2^{nd} iteration $I2$. On Figure 3.4, 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 3.4: 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.

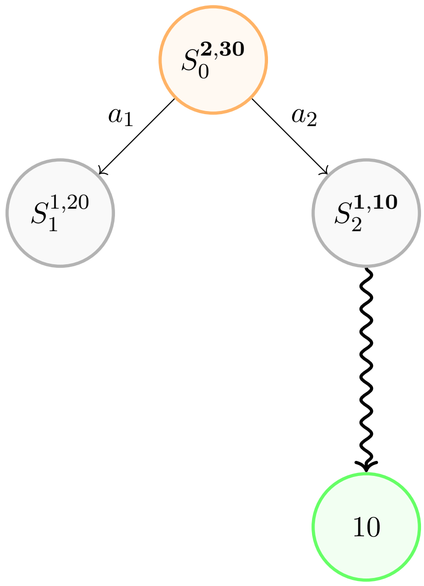


FIGURE 3.5: Simulation and Backpropagation - I2

We can then simulate (Figure 3.5) 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.

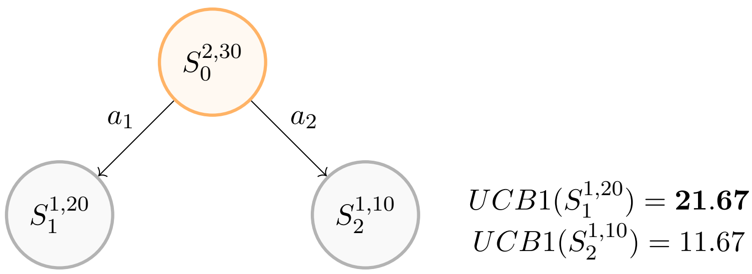


FIGURE 3.6: Selection - I3

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

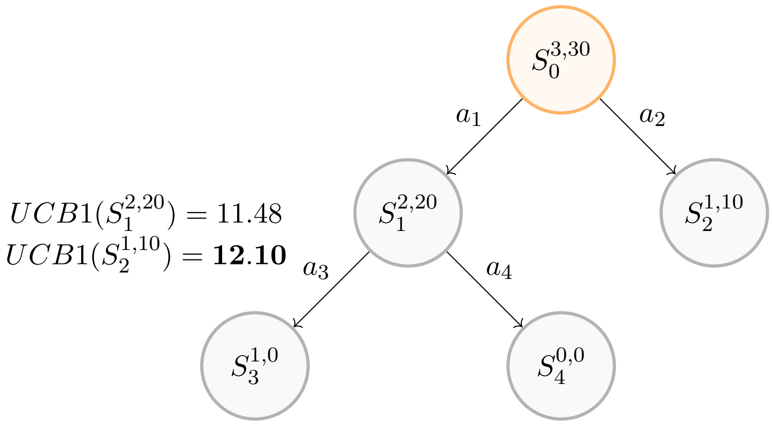


Figure 3.7: Selection and Expansion - I3

Based on $UCB1$ score we decide to simulate from $S_3^{0,0}$ on Figure 3.8

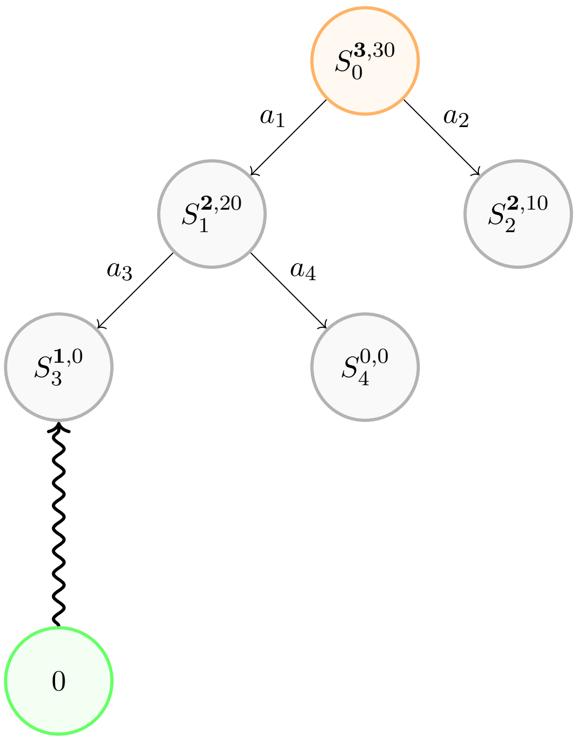


Figure 3.8: Simulation and Backpropagation - I3

Let's do the fourth iteration $I4$ represented on Figure 3.9:

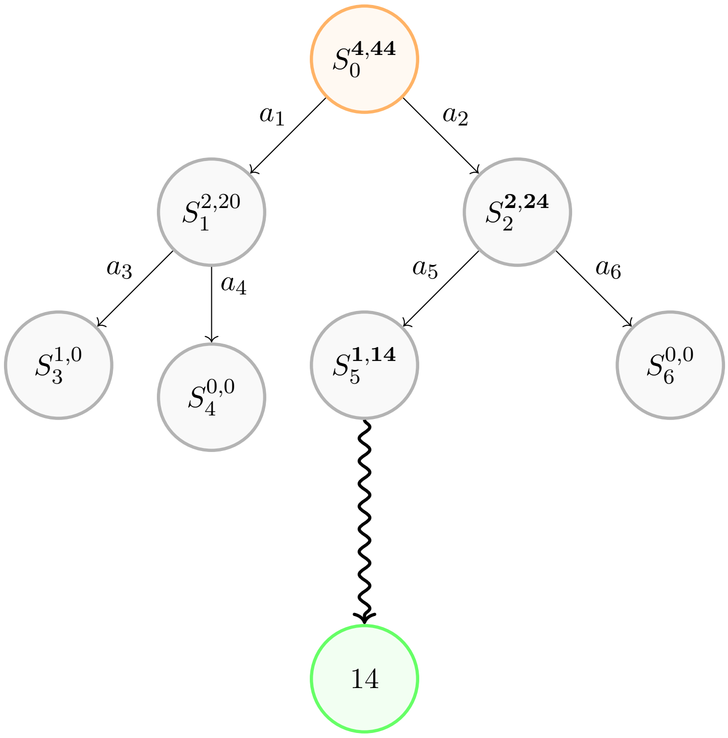


FIGURE 3.9: Selection - Simulation - Backpropagation - I4

If we were to stop at this stage of the algortihm, the best action to undertake is a_2 because it has the higher average value: $\bar{V_1} = \frac{20}{2} \leq \bar{V_2} = \frac{24}{2}$.

3.1.2 Generalisation

The Monte Carlo Tree Search algorithm can be summarised on Figure 3.10:

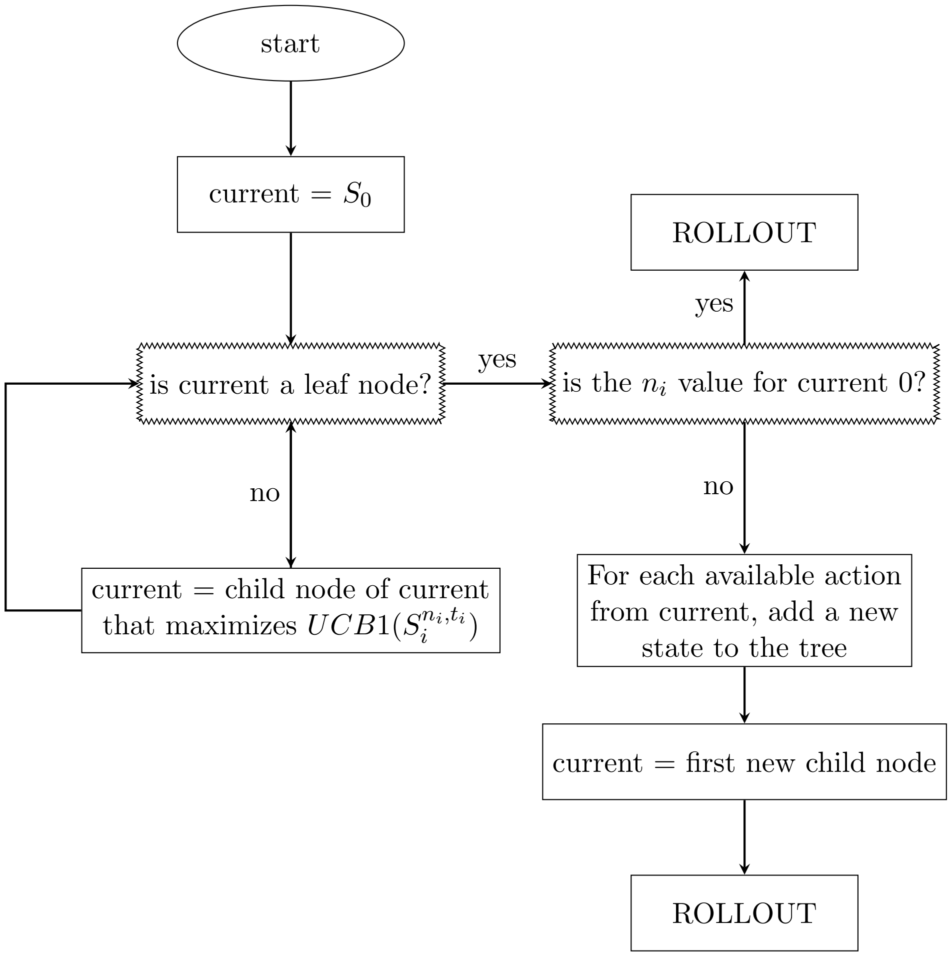


FIGURE 3.10: Flow MCTS

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

$$UCB1(S_i^{n_i,t_i}) = \bar{V}_i + c\sqrt{\frac{\ln N}{n_i}} \tag{3.3}$$

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 3.1.1, 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.

$$\mathcal{B}(S_i^{n_i,t_i}) = S_i^{n_i+1,t_i+\mathcal{R}(S_i^{n_i,t_i})} \tag{3.4}$$

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

3.1.2.1 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 3.11. The input is an instance I_i, as defined in Chapter 2:

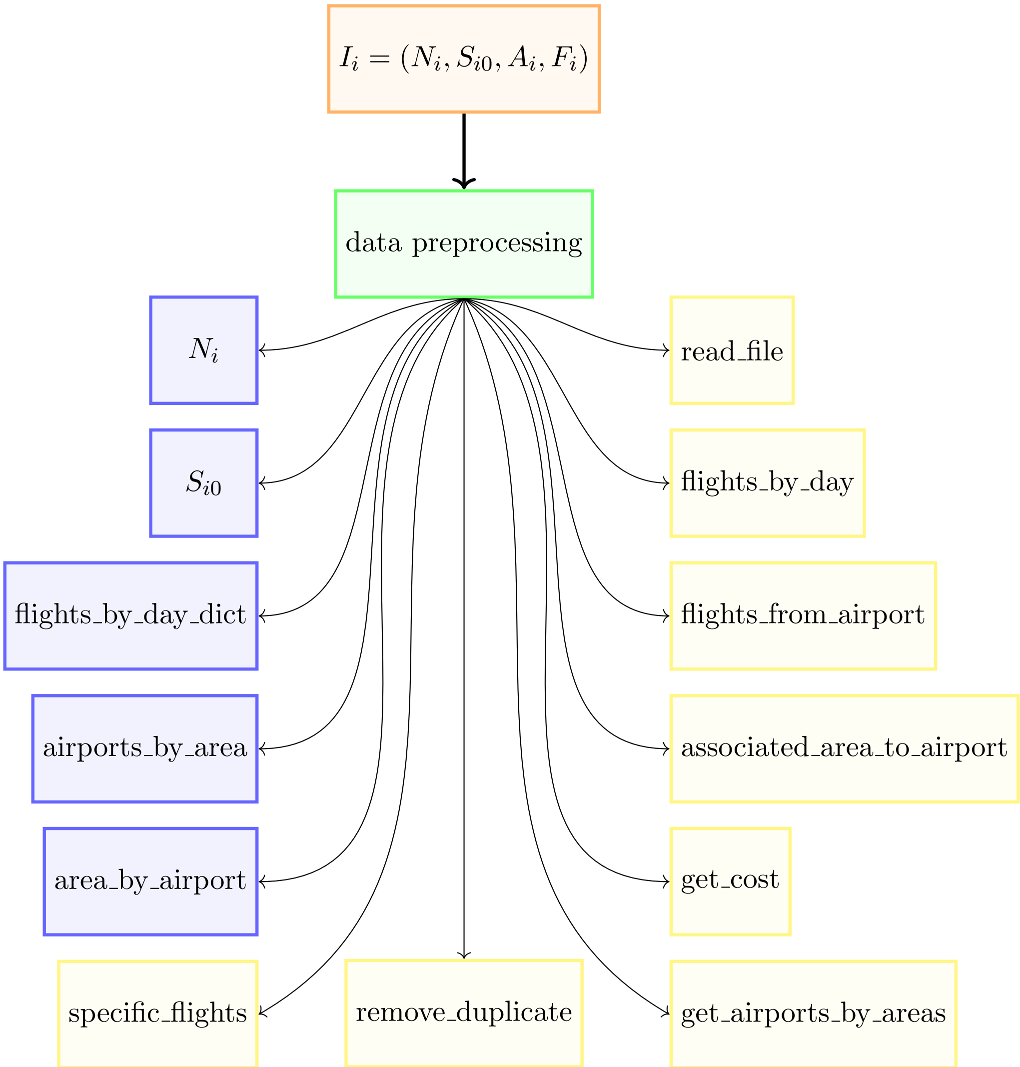


FIGURE 3.11: Explanation of the data preprocessing class

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.

3.1.2.2 Node

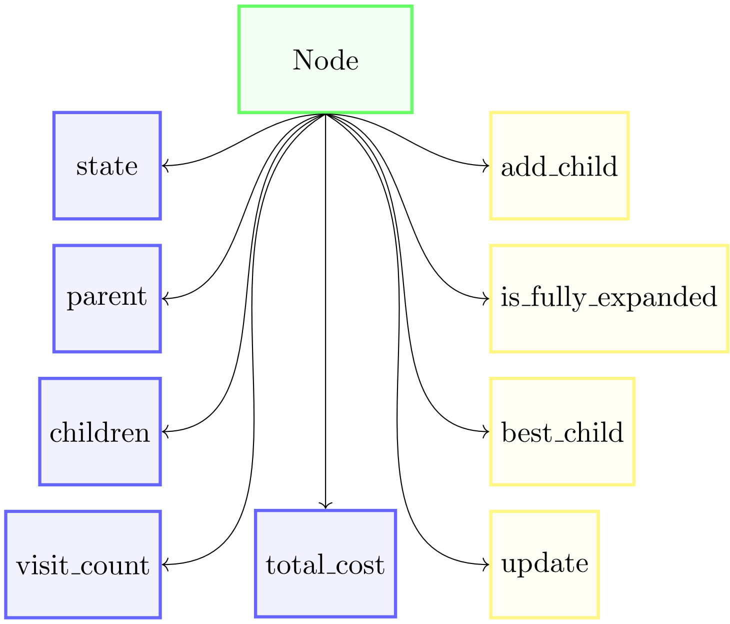


Figure 3.12: Explanation of the Node class

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.

3.1.3 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
```

3.1.4 Different policies

3.1.4.1 Selection policy

Our selection policy is based on

3.1.4.2 Simulation policy

Chapter 4

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 5

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.

5.1 Summary of Work

5.2 Critics

5.3 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.