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

2.1 Optimisation in Air Travel

2.1.1 Airline Scheduling Optimisation

IN [1].

- Key driver of the revenue for a company: assign the wrong airport with not the good capacity is not optimised
- Complex problem - number of daily flights for a company can be very important
- Comparison between the traditional apporach and the new apporach
- Traditionnal: treated as a standalone problem - assumptions like daily flight at fixedhours - point forecasts for the passenger demand
- Modern: integrating the FAP with other airline processes such as maintenance - crew Scheduling problem

2.1.2 crew scheduling problem

[2]

The crew scheduling problem is defined in this paper where the author emphasises the particularity of this problem compared to the set covering/partitonning problem.

The CSP's goal is to assign crews to a sequence of task with defined start and end times. The goal is to ensure that all crew's member can complete all the tasks, without exceeding a total working time.

According to this report [3] - all the airlines companies deal with crew scheduling and especially the low cost companies because that's how they optimise everything and then become competitive.

2.1.3 disruption management

[4]

Diruption in airlines can occur due to various factor -¿ crew unavailability, accumulated delay along the day because of air control that compells to cancel or delay flights, weather, mechanical problems ...

Disruption management for airline company is crucial because one flight' schedule is planned several months in advanced [5], hence the objective is to minimise the impact on flights, passangers, airline operations. The two mains drivers of disruption management are aircraft recovery and crew recovery.

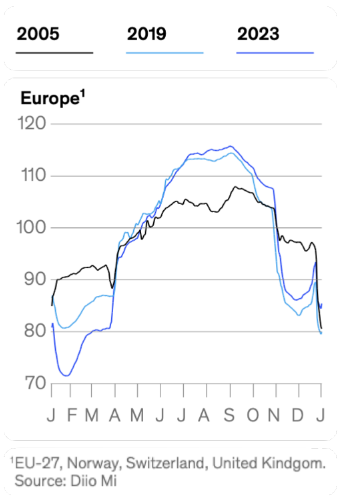
- Aircraft recovery: Reassigning aircraft to flights after disruption by taking care of schedules, airports availibility, maintenance
- Crew recovery: reassigning crew members to flights by respecting all flights are staffed appropriattely

2.1.4 Airline adaptation to new demand

Airline schedule also take care of passenegrs demand.

The airline industry is fastly evolving - growing dominance of leisure travel over business travel. It has created a new demand seasonnality especially in Europe -¿ cairline companies have to handle this seasonnality demand by balancing high peak

Demand' seasonnality : airline companies have to adapt Challenging for airline companies as they have to balance peak seasons with the risk of underutilisation during off-peak times. Off peak seasons -¿ planes and crews underused -¿ driving up costs for passengers but also for the companies



- Summer Peak: The airline schedules 65% more seats in August than in February, deploying more aircraft and crew to meet the demand. Optimization models help the airline decide which routes to prioritize, how many additional flights to add, and how to manage crew rotations to ensure smooth operations despite the high pressure on resources

- Winter Through: In winter, the same airline faces low demand, with many aircraft potentially sitting idle. To optimize, the airline might engage in ACMI leasing, reducing its owned fleet temporarily, and outsourcing capacity to adjust dynamically to lower demand. Maintenance activities are ramped up, and crew members are encouraged to take vacations or attend training during this period, optimizing overall productivity.

- Revenue Management: Dynamic pricing algorithms adjust fares in real-time based on demand patterns. During summer, prices are optimized to capture the maximum willingness to pay from tourists, while in winter, prices might be adjusted to stimulate demand, filling otherwise empty seats and ensuring higher aircraft utilization.

2.1.5 Flight connection challenges

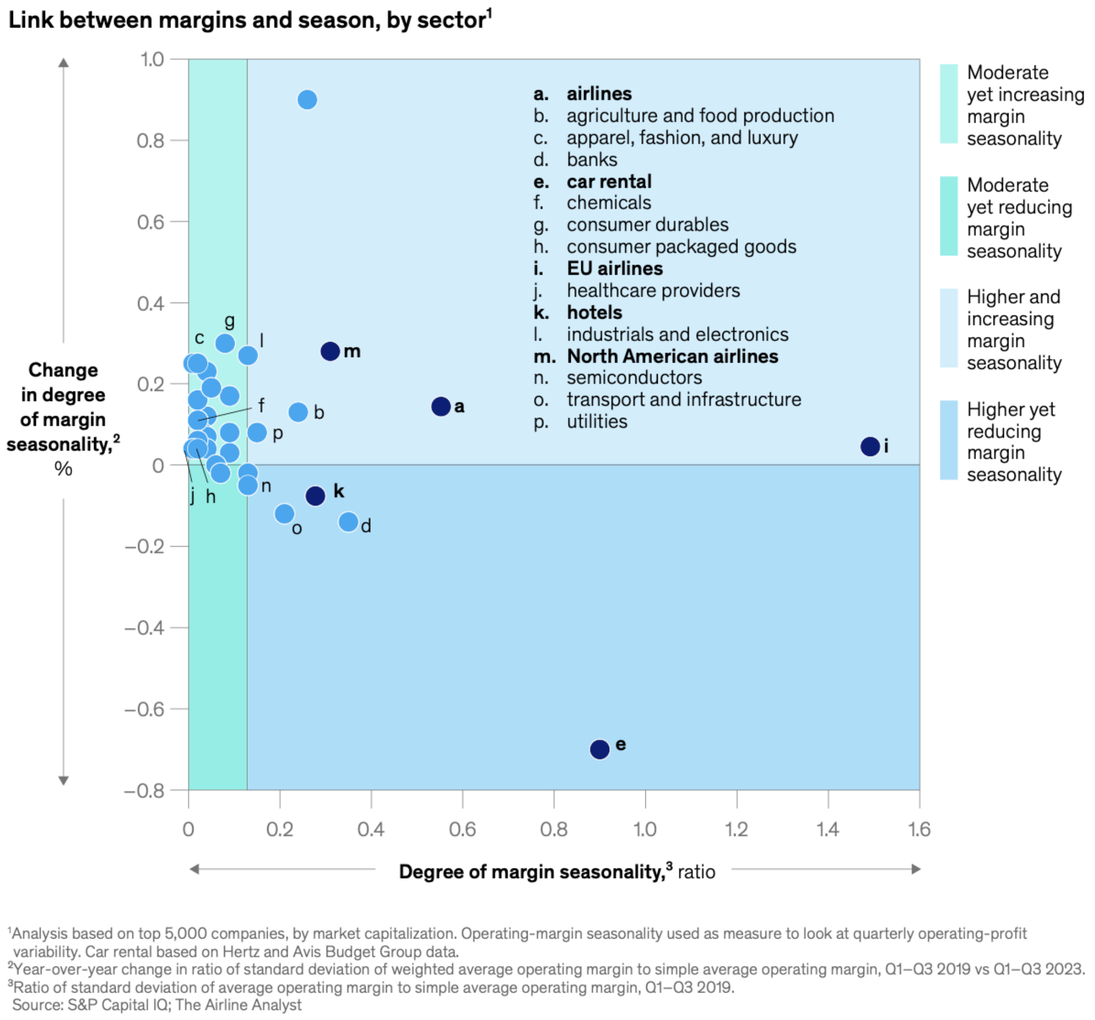


FIGURE 2.1: Compared with other sectors, airlines exhibit a significant and growing link between margins and seasons.

2.2 Traveling Salesman problem and its adapation

[6]

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. 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.

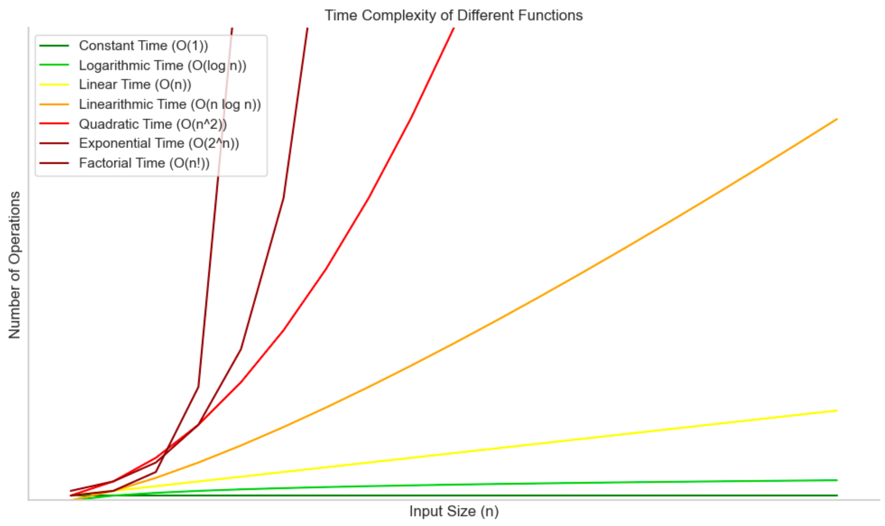


FIGURE 2.2: Time complexity of different functions

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 solved 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. From [7]

The TSP has been studied a lot, The TSP has been studied a lot, especially because from the TSP il découle de nombreux variantes:

- 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.
- 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.
- Time Window TSP (TWTSP): Each city has to be visited in a defined time slot.
- 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.

- Stochastic TSP (STSP): The distances between the cities or the cost of travels are stochastic (ı.e random variables) rather than deterministic.
- 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.
- Generalised TSP (GTSP): The cities are grouped into clusters, the goal is to visit exactly one city from each cluster.
- Open TSP (OTSP): The traveler does not have to end his journey at the starting city.

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.
  - **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.
  - **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.
- **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.

  - **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.
  - **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.
  - **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.

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 the previous subsection to solve TSP because MCTS is typically used in games. 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. [8]

In 2016, Lee Sedol [9] - the best Go's player in the world has been beaten by AlphaGo 4-1 [10].

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. [11]

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 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 statiscally biased sequence is applied from this node until a terminal condition is reached e.g no further actions are possible. [12]

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 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.

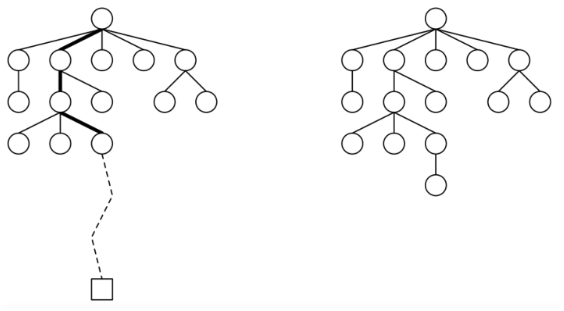


FIGURE 2.3: Assymetrical growth of MCTS - Simulation and Expansion - [13]

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 $S_0^{0,0}$ 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

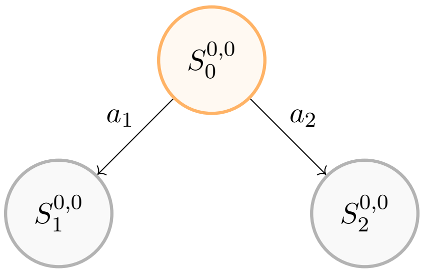


FIGURE 2.4: Selection - $I1$

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. taking random actions from this node to a terminal state as shown on Figure 2.5:

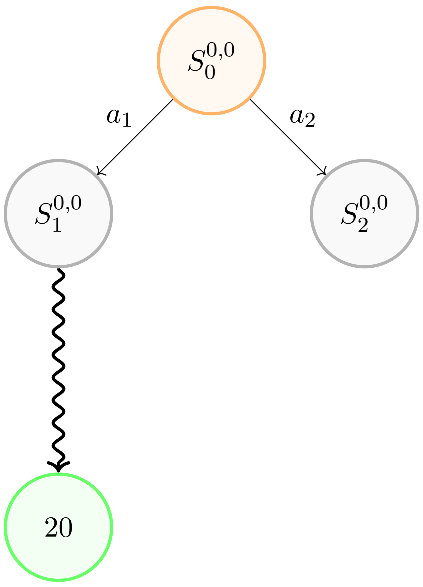


FIGURE 2.5: Simulation - $I1$

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

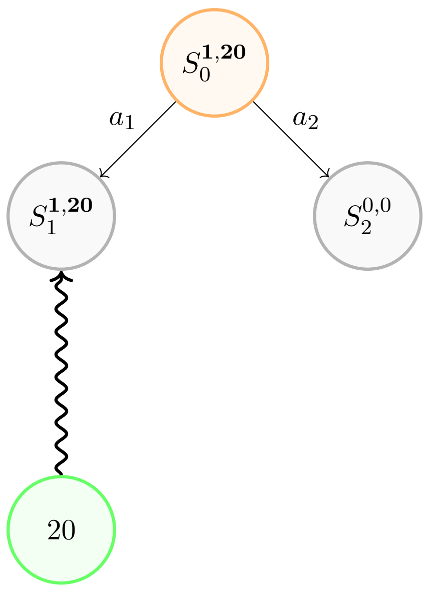
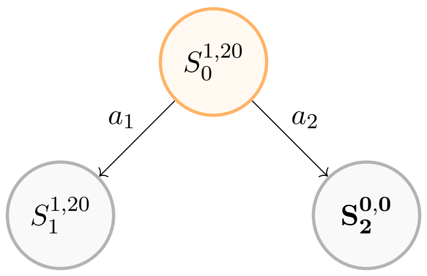


FIGURE 2.6: Backpropagation - I1

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.

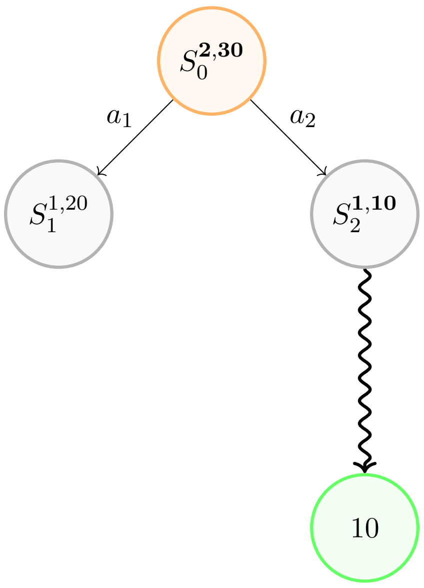


FIGURE 2.8: Simulation and Backpropagation - I2

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 3rd iteration, based on the $UCB1$ score we decide to choose a_1.

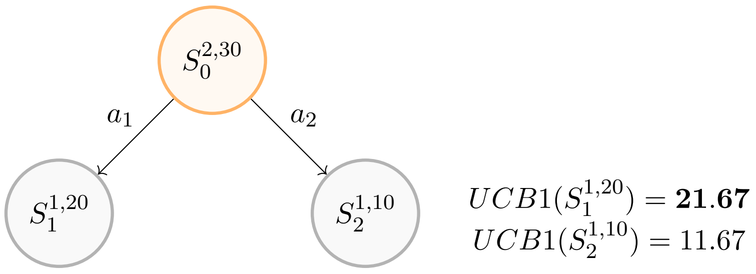

$S_0^{2,30}$
a_1
a_2
$S_1^{1,20}$
$S_2^{1,10}$
$UCB1(S_1^{1,20}) = \mathbf{21.67}$
$UCB1(S_2^{1,10}) = 11.67$

FIGURE 2.9: Selection - I3

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

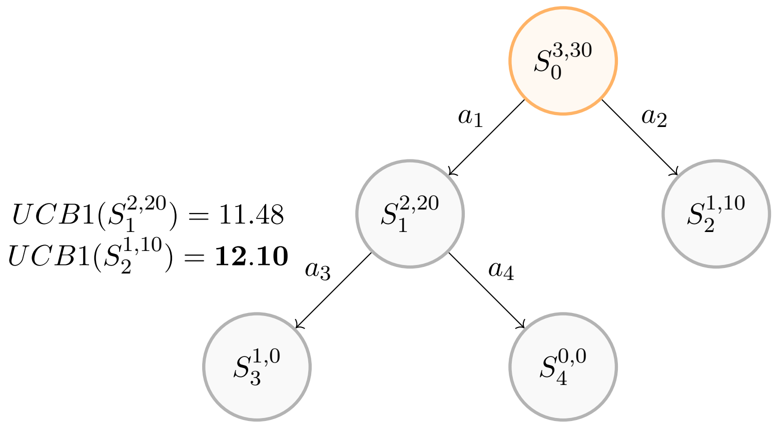


FIGURE 2.10: Selection and Expansion - I3

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

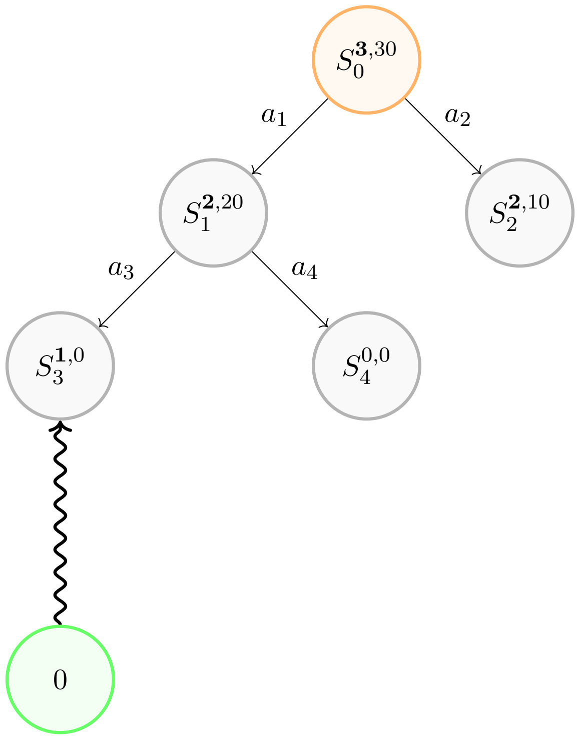


FIGURE 2.11: Simulation and Backpropagation - I3

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

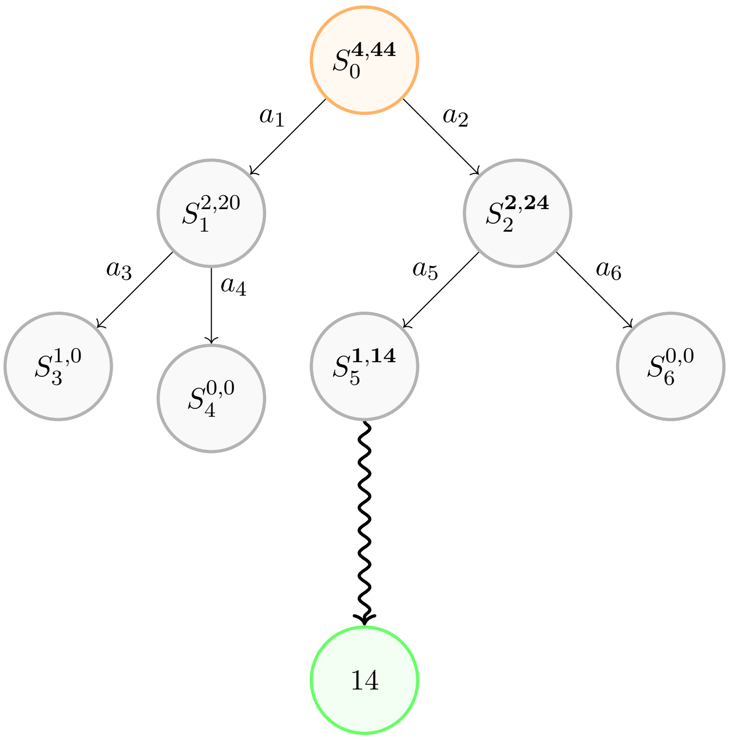


FIGURE 2.12: Selection - Simulation - Backpropagation - I4

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: $\bar{V_1} = \frac{20}{2} \leq \bar{V_2} = \frac{24}{2}$.

2.4 Driving Factors

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:

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

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

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

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:

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

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

Parameters

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

Variables

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

Constraints

1. Starting and Ending Constraints:

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

2. Flow Constraints:

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

Objective Function

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

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

Constraints

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

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

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

- Visit exactly one airport in each area each day:

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

- 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

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.

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] 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.

[4] 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.

[5] 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.

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

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

[8] 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].

[9] 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].

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

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

[12] 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.

[13] 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.