**Chapter 2**

# Literature Review

## 2.1 Optimisation in Air Travel

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

### 2.1.1 Fleet Assignment Problem

The Fleet Assignment Problem (FAP), as discussed in [1] involves assigning different types of aircraft to flights based on their capabilities, operational costs, and revenue potential. This decision greatly influences airline revenues and is a vital part of the overall scheduling process. The complexity of FAP is driven by the large number of flights an airline manages daily and its interdependencies with other processes like maintenance and crew scheduling.

2

### 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 airlines, for example in the United Kingdom in 2023, low-cost flights 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: Optimisation tools help manage the complex logistics of matching available aircraft with rescheduled flights, considering factors like airport availability and maintenance requirements.
* Crew recovery: Optimisation tools are used to adjust crew schedules, taking into account factors such as legal working hours, crew availability, and the need to cover all flights efficiently. These tools help in developing feasible and compliant crew rosters that adapt to the new flight schedules.

These optimisation strategies, supported by advanced software, for instance [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

Airline companies must continuously adapt their schedules to meet evolving market demands, particularly with the growing dominance of leisure travel over business travel, which has introduced new patterns of demand. For example, 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.

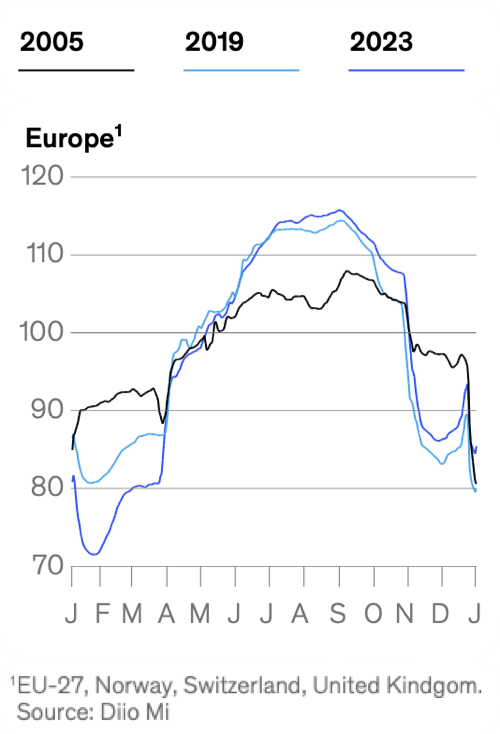
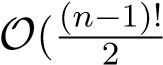


Figure 2.1: European demand seasonality [9]

Since travel demand varies throughout the year, airlines use a variety of techniques to achieve operational efficiency while maximising revenue [9]. For instances, airlines sell nearly 65% more seats in August than they do in February. To ensure their operations remain efficient during peroids of hightened demand, airline companies make the required allowance for additional aircraft and crew by optimisation models that specify priority routes and requirements for additional flights, alongside effective crew rotation management. In contrast, winter months pose a different type of problem where demand drops, which can potenitally lead to underutilisation of aircrafts. To manage this, airlines are known to turn to ACMI leasing (agreement between two airlines, where the lessor agrees to provide an aircraft, crew, maintenance and insurance [10]) during periods of low demand to temporarily reduce fleet size by outsourcing their capacity. Alongside this, airline companies also increase maintenance activities and incentivise crews to take holidays or undergo training to maximise productivity across the operation. Equally, on a year-round basis, airlines apply dynamic pricing algorithms to vary fares in reaction to real-time demand patterns. In high-demand summer months, fares are tactically set so as to maximise revenues from travelers willing to pay more, while in winter, pricing strategies are aimed at stimulating demand with fare reductions to fill seats that otherwise would have gone empty. Such adaptive strategies are critical to the airlines for effectively beating the seasonal ebbs and flows in the travel industry.

## 2.2 Traveling Salesman problem and its adapation

The Traveling Salesman Problem is a well known problem in the Operational Research and Computer Science fields. A simple description of the TSP is to find the best roundtrip for a saleman that has to travel around a given number of cities while minimising the overall journey’s distance. This problem is characterised as NP-Hard [11]. This means that there is no known polynomial-time algorithm that can solve all instances of the problem efficiently . Regarding time complexity, if we were to solve it exploring all the possible solutions, the time complexity would have been) where *n* represents the number of cities.

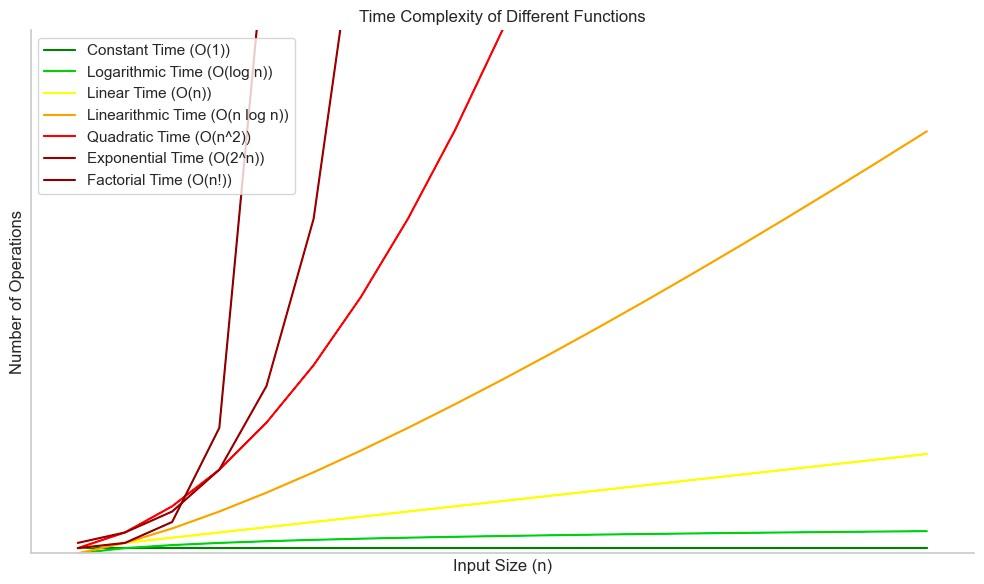


Figure 2.2: Time complexity of different functions [12]

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

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

* Symmetric TSP (STSP): The distance between cities are symmetric, meaning that the distance to travel from city A to city B is the same as from city B to city A.
* Assymetric TSP (ATSP): The distance between cities are assymetric, meaning that the distance to travel from city A to city B is different than the distance to travel from city B to city A.[13]
* Multiple TSP (mTSP): Instead of one salesman, multiple salesman are starting from one city, they visit all the cities such that each city is visited exactly once. [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 minimise the overall traveler’s distance while maximising the price 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 time, 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 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, specifically for largescale 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, which are gradually refined through updates based on pheromone trails. [27]

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

## 2.3 The Monte Carlo Tree Search algorithm

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

Go is a board game from China where two players take turns placing black or white stones on a grid. The goal is to capture territory by surrounding empty spaces or the opponent’s stones. Despite its simple rules, Go is incredibly complex, with countless possible moves and strategies. It is known for its balance between intuition and logic, hence why it has been a significant focus of artificial intelligence researchers ,[35]. In 2016, Lee Sedol [36] - the best Go’s player in the world was 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 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 to select in the tree to perform simulations. The selection policy, typically balances the exploration (looking into parts of the tree that has not been visited yet) and the exploitation (looking into parts of the trees that appear to be promising.) Once the node is selected, a simulation (sequence of available actions based on a simulation policy) is applied from this node until a terminal condition is reached e.g no further actions are possible. [39]

To ensure that the reader understands the various stages of the Monte Carlo Tree Search

Algorithm, we will examine a detailed example. This example will illustrate

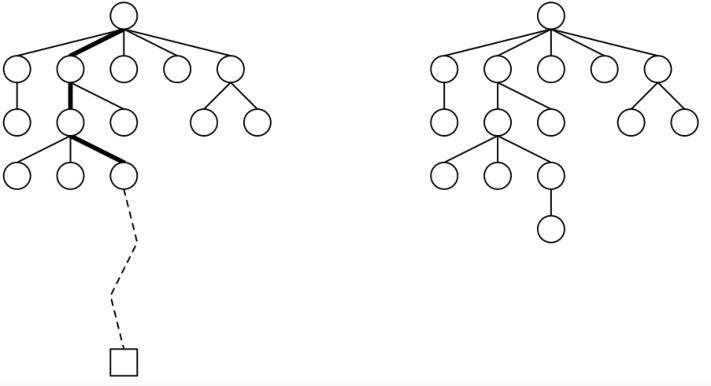
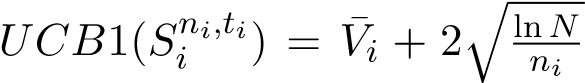


Figure 2.3: Assymetrical growth of MCTS - Simulation and Expansion - [40]

each component of the algorithm in action. Furthermore, we will generalise the principles discussed, as the methodology of this paper is built on the application of the MCTS algorithm.

### 2.3.2 Example

Let’s say we are given a maximisation problem. When begening the game, you have two possible actions *a*1 and *a*2 from in the tree T . Every node is defined like so:

where *ni* represents the number of times node *i* has been visited, *ti* the total score of this node. Moreover, for every node - we can compute a selection metric, for instance the *UCB*1 value:  where represents the average value of the node, *ni* 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 -. At the beginning



of *I*1, we have to choose between these two child nodes (or choose between taking *a*1 or *a*2). After, we have to calculate the *UCB*1 value for these two nodes and pick the node that maximises the *UCB*1 value (as we are dealing with a maximisation problem).

In Figure 2.4, neither of these have been visited yet so.

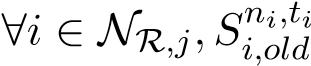
Hence we decide to choose randomly.

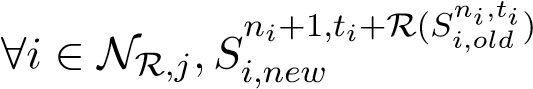
 is a leaf node that has not been visited - then we can simulate from this node, which means selecting actions from this node based on the simulation policy to a terminal state as shown on Figure 2.5:



The terminal state has a value of 20, we can write that the rollout/simulation from node  node is ) = 20 . The final step of *I*1 is backpropagation. Every node that has been visited in the iteration is updated. Let NR*,j* be the indexes of the nodes visited during the *j* − *th* iteration of the MCTS:

• Before backpropagation:

 (2.1) • After backpropagation:

 (2.2)

We can then define a backpropagation function:

B : NR*,j* → NR*,j*

*Sini,ti* →7 *Sini*+1*,ti*+R(*Sini,ti*)

Then, back to the example on Figure 2.6 we update the nodes  and

.



The fourth phase of the algorithm has been done for *I*1. Therefore, we can start the 2*nd* iteration *I*2. On Figure 2.7, we can either choose *a*1 or *a*2. When a child node has not



Figure 2.7: Selection - I2

been visited yet, you pick this node for the Selection or you can compute the *UCB*1 value, it leads to the same conclusion.



Figure 2.8: Simulation and Backpropagation - I2

We can simulate (Figure 2.8) from the chosen node and ) = 10 and backpropagate all the visited nodes:  and . Next, we start the 3*rd* iteration, based on the *UCB*1 score we decide to choose *a*1.

*UCB*1(*S*11*,*20) = **21***.***67** *UCB*1(*S*21*,*10) = 11*.*67 Figure 2.9: Selection - I3

 is a leaf node and has been visited so we can expand this node.



Figure 2.10: Selection and Expansion - I3

Based on *UCB*1 score we decide to simulate from on Figure 2.11



Figure 2.11: Simulation and Backpropagation - I3

This is the fourth iteration *I*4 represented on Figure 2.12:



Figure 2.12: Selection - Simulation - Backpropagation - I4

The MCTS algoirthm can either be stopped because you are running out of time or because you have no more available actions. For instance, if we were to stop at this stage of the algorithm, the best action to undertake is *a*2 because it has the higher average value: .

**Chapter 3**

# Problem Description

## 3.1 Overview

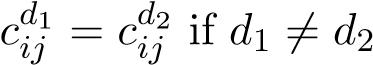
Kiwi’s traveler wants to travel in *N* different areas in *N* days, let’s denote *A* the set of areas the traveler wants to visit:

*A* = {*A*1*,A*2*,...AN*}

where each *Aj* is a set of airports in area *j*:

*Aj* = {*aj,*1*,aj,*2*,...,aj,kj*}

where *aj,kj* being airports in area *j* and *kj* 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 *cdij* the cost to travel from *cityi* to *cityj* on day *d*. We do not necessarly have *cdij* = *cdji* neither. 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:

* A = {1*,*2*,...,N*}: Set of areas.

19

*Aj* = {*aj,*1*,aj,*2*,...,aj,kj*}: Set of airports in area *j* ∈ A.

* D = {1*,*2*,...,N*}: Set of days.
* *Ud* ⊆ A: Set of areas that have not been visited by the end of day *d*.

Parameters

* *cdij*: Cost to travel from airport *i* to airport *j* on day *d* ∈ D.

Variables

* *xdij*: Binary variable which is 1 if the traveler flies from airport *i* to airport *j* on day *d*, and 0 otherwise.
* *vjd*: 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:

|  |  |  |  |
| --- | --- | --- | --- |
|   *N*−1 minX    *d*=2 | X  *N*−1 | X  *N* |     *cdijxdij* + X *c*1*S*0*,jx*1*S*0*,j* + X X *cNij xNij*     *j*∈*A*1 *i*∈*AN j*∈*A*1  |
| S S | | |

*i*∈ *Ak j*∈ *Ak*

*k*=2 *k*=3

### Constraints

• Starting at the known starting airport *S*0 at take an existing flight connection:

#### X 1

*xS*0*,j* = 1

*j*∈*A*1



* Visit exactly one airport in each area each day:

X X *xdij* = 1 ∀*d* ∈ {1*,*2*,...,N* − 1}

*i*∈*Ad j*∈*Ad*+1

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

|  |  |  |
| --- | --- | --- |
| X *d* X *xik* =  *k*∈*Ad k*∈*Ad*−1 |  | ∀*i* ∈ [ *Aj,*∀*d* ∈ {2*,*3*,...,N*}  *j*=1 |

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

##### X X *N*

*xij* = 1

*i*∈*AN j*∈*A*1 ∀(*i,j*) ∈ *AN* × *A*1*,cNi,j* ∈ R+∗ • Ensure each area is visited exactly once:

X *vjd* = 1 ∀*j* ∈ A

*d*∈D

Update the unvisited list:



• 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*:

*N*

|  |  |
| --- | --- |
| *xdij* ≤ *cdij* · *vjd* | ∀*i,j* ∈ ([ *Aj*)2*,*∀*d* ∈ D  *j*=1 |
| *xdij* ≤ *vjd* • Binary variable constraints: | *N*  ∀*j* ∈ [ *Aj,*∀*d* ∈ D  *j*=1 |
| *xdij* ∈ {0*,*1} | *N*  ∀(*i,j*) ∈ ([ *Aj*)2*,*∀*d* ∈ D  *j*=1 |

*vjd* ∈ {0*,*1} ∀*j* ∈ A*,*∀*d* ∈ D

## 3.2 Instances

### 3.2.1 Description

We are given a set of 14 Instances *In* = {*I*1*,I*2*,...,I*13*,I*14} that we have to solve. Every instances has the same overall structure.

For example, the first few lines of *I*4 are:

13 GDN first

#### WRO DL1

second

BZG KJ1

third

BXP LB1

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

After all the information regarding the areas and the airports, we have the flight connections informations. In Table 3.1, few flights are displayed from *I*6 for illustrative purposes.



For every instance *Ii*, we know what connections exist between two airports for a specific day and the associated cost. There might be in some instances flights connections at day 0, this means these connections exist for every day of the journey at the same price. Furthermore, we could have the same flight connections at a specific day but with different prices. Futhermore, we have to consider solely the more relevant connections i.e. the flight connection with the lowest fare.

### 3.2.2 General formulation

An instance *Ii* can be mathematically defined as follows:

*Ii* = (*Ni,Si*0*,Ai,Fi*)

where:

* **Number of Areas** *Ni*:

*Ni* ∈ N

The total number of distinct areas in instance *Ii*.

**Starting Airport** *Si*0:

*Si*0 ∈ Airports

The starting airport of the traveller.

* **Airports in Each Area**:

*Ai* = {*Ai,*1*,Ai,*2*,...Ai,Ni*}

where each *Ai,j* is a set of airports in area *j* for instance *i*:

*Ai,j* = {*ai,j,*1*,ai,j,*2*,...,ai,j,kj*}

with *ai,j,kj* being airports in area *j* and *kj* is the number of airports in area *j*.

* **Flight Connections**:

*Fi* = {*Fi,*0*,Fi,*1*,Fi,*2*,...,Fi,Ni*}

|  |  |  |
| --- | --- | --- |
| *adi,k,*1 *adi,k,*2  ... | *aai,k,*1 *aai,k,*2  ... |  *fi,k,*1 *fi,k,*2  ...    |

where each flight matrix *Fi,k* represents the flight information of instance *i* on day *k*:







*Fi,k* = 







* **Columns**:

∗ Departure Airport: *ad*

*adi,k,lk,i aai,k,lk,i fi,k,lk,i*

(Departure airport for the *j*-th flight on day *k*)

*i,k,j*

∗ Arrival Airport: *aai,k,j* (Arrival airport for the *j*-th flight on day *k*)

∗ Cost: *fi,k,j* (Cost of the *j*-th flight on day *k*), where *j* ∈ [1*,lk,i*]

* **Rows**: Each row corresponds to a specific flight on day *k*. The number of rows *lk,i* depends on the number of flights available on that day.

### 3.2.3 Kiwi’s rules

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

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

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.

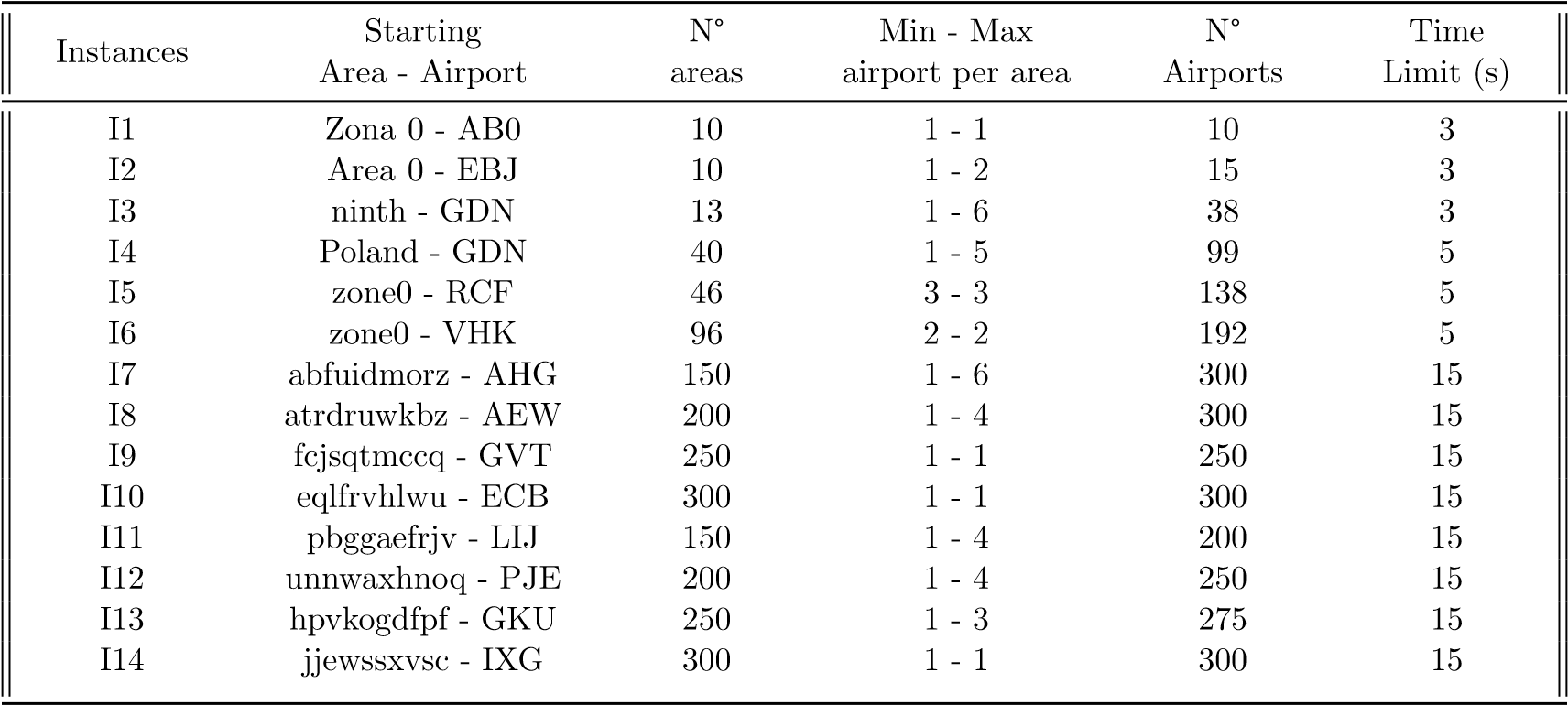


Table 3.3: Instances and their respective parameters

# 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.](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.](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,](https://www.oag.com/blog/unstoppable-lccs-growth-indicates-new-norm) 2023.
4. FranceTV Slash / Enquˆetes. Ryanair: Y-a-t-il un rh dans l’avion? enquˆete sur les conditions de travail du g´eant du low-cost, 2024. URL [https://www.youtube. com/watch?v=4TOsoX6aPiA.](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.](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/](https://www.cntraveler.com/story/the-complex-process-behind-your-flights-schedule#:~:text=Flight%20schedules%20are%20mapped%20out,affect%20departure%20and%20arrival%20times.)

[the-complex-process-behind-your-flights-schedule#:~:text=Flight%](https://www.cntraveler.com/story/the-complex-process-behind-your-flights-schedule#:~:text=Flight%20schedules%20are%20mapped%20out,affect%20departure%20and%20arrival%20times.)

[20schedules%20are%20mapped%20out,affect%20departure%20and%20arrival% 20times.](https://www.cntraveler.com/story/the-complex-process-behind-your-flights-schedule#:~:text=Flight%20schedules%20are%20mapped%20out,affect%20departure%20and%20arrival%20times.)

1. None. Advanced decision support for aviation disruption management. [https:](https://www.inform-software.com/en/lp/aviation-disruption-management#:~:text=Proper%20aviation%20disruption%20management%20means,the%20schedule%2C%20while%20minimizing%20costs.)

[//www.inform-software.com/en/lp/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.)

[~:text=Proper%20aviation%20disruption%20management%20means,the% 20schedule%2C%20while%20minimizing%20costs.,](https://www.inform-software.com/en/lp/aviation-disruption-management#:~:text=Proper%20aviation%20disruption%20management%20means,the%20schedule%2C%20while%20minimizing%20costs.) 2024.

1. 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,](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 de- |
| mand. *McKinsey*, January 8, 2024. | URL [https://www.mckinsey.](https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/how-airlines-can-handle-busier-summers-and-comparatively-quiet-winters#/) |

[com/industries/travel-logistics-and-infrastructure/our-insights/ how-airlines-can-handle-busier-summers-and-comparatively-quiet-winter](https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/how-airlines-can-handle-busier-summers-and-comparatively-quiet-winters#/)s#

[/.](https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/how-airlines-can-handle-busier-summers-and-comparatively-quiet-winters#/)

1. None. What is acmi leasing? ACC Aviation, 2024.
2. Lark Editorial Team. Np hard definition of np hardness. *Lark*, 26 December, 2023.
3. Hennie de Harder. Np-what? complexity types of optimization problems explained. *Towards Data Science*, August 17, 2023.
4. Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983. ISSN 01676377. doi: https://doi.org/10.1016/0167-6377(83)90048-2. URL [https://www. sciencedirect.com/science/article/pii/0167637783900482.](https://www.sciencedirect.com/science/article/pii/0167637783900482)
5. 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.](https://www.sciencedirect.com/science/article/pii/S0305048304001550)
6. Sneˇzana Mitrovi´c-Mini´c 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.](https://www.sciencedirect.com/science/article/pii/S0167637705000295)
7. 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.](https://www.sciencedirect.com/science/article/pii/S0377221710002973)
8. 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/](https://www.sciencedirect.com/science/article/pii/S219243762030087X)

[pii/S219243762030087X.](https://www.sciencedirect.com/science/article/pii/S219243762030087X)

1. Aviv Adler. The traveling salesman problem under dynamic constraints. *Massachusetts Institute of Technology*, Feb 2023.
2. Petric˘a 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.](https://www.sciencedirect.com/science/article/pii/S0377221723005581)
3. 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.
4. 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.
5. 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.](https://www.sciencedirect.com/science/article/pii/0377221785901511)
6. Not specified. Travelling salesman problem using dynamic programming. *Geeksforgeeks*, 19 April, 2023.
7. 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.
8. 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.
9. 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.
10. 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.](https://www.sciencedirect.com/science/article/pii/S1568494621003628)
11. Wikipedia. Havannah (board game) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Havannah%20(board% 20game)&oldid=1240631485,](http://en.wikipedia.org/w/index.php?title=Havannah%20(board%20game)&oldid=1240631485) 2024. [Online; accessed 18-August-2024].
12. Wikipedia. Game of the Amazons — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Game%20of%20the%20Amazons& oldid=1235225698,](http://en.wikipedia.org/w/index.php?title=Game%20of%20the%20Amazons&oldid=1235225698) 2024. [Online; accessed 18-August-2024].
13. Wikipedia. Lines of Action — Wikipedia, the free encyclopedia. [http://en. wikipedia.org/w/index.php?title=Lines%20of%20Action&oldid=1198717858,](http://en.wikipedia.org/w/index.php?title=Lines%20of%20Action&oldid=1198717858) 2024. [Online; accessed 18-August-2024].
14. Wikipedia. Shogi — Wikipedia, the free encyclopedia. [http://en.wikipedia. org/w/index.php?title=Shogi&oldid=1240175752,](http://en.wikipedia.org/w/index.php?title=Shogi&oldid=1240175752) 2024. [Online; accessed 18August-2024].
15. Joris Dugu´ep´eroux, 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.
16. 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.

1. Mark Winands, Yngvi Bj¨ornsson, 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.
2. Wikipedia. Go (game) — Wikipedia, the free encyclopedia. [http://en.wikipedia. org/w/index.php?title=Go%20(game)&oldid=1239511822,](http://en.wikipedia.org/w/index.php?title=Go%20(game)&oldid=1239511822) 2024. [Online; accessed 18-July-2024].
3. Wikipedia. Lee Sedol — Wikipedia, the free encyclopedia. [http://en.wikipedia. org/w/index.php?title=Lee%20Sedol&oldid=1234296689,](http://en.wikipedia.org/w/index.php?title=Lee%20Sedol&oldid=1234296689) 2024. [Online; accessed 11-August-2024].
4. Google DeepMind. Alphago - the movie / full award-winning documentary. Youtube, 2020.
5. Not mentionned. Explain the role of monte carlo tree search (mcts) in alphago and how it integrates with policy and value networks. EITCA, 2024.
6. 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.
7. Hendrik Baier and Peter D. Drake. The power of forgetting: Improving the lastgood-reply policy in monte carlo go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:303–309, 2010. URL [https://api.semanticscholar.](https://api.semanticscholar.org/CorpusID:13578069)

[org/CorpusID:13578069.](https://api.semanticscholar.org/CorpusID:13578069)

1. 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.