# **3. METHODOLOGY**

The following sections are a description of the SAP solved in this project. This chapter introduces the data and constraints used to build the SAP, then outlines and justifies the process of formulating the SAP. Finally, the coding model created to solve the SAP is defined.

## 3.1 Initial Flight Data

Arup is a multinational consultancy company for various engineering industries and built environment. The Airport Planning department at Arup – the sponsors of this project – work on many developments at the Heathrow Airport. Therefore, this project focused on creating an optimised solution for an SAP at Heathrow Airport.

All the initial data used to solve the SAP on this project were of Heathrow Airport, from the year 2017.

### 3.1.1 *Flight Schedule Data*

When solving any SAP, the most important pieces of information required are the arrival and departure times of the flights. The secondary requirements are information about the fleet types, destinations of the flights, etc. All these essential data are included in the flight schedule.

The initial flight schedule dataset obtained had details of flights at every terminal for the whole year of 2017. This was an account of 47,1072 flights, entering and leaving the Heathrow airport.

Table-A1 in Appendix I displays the first 5 and last 5 records of the flight schedule. The list below outlines all the data elements included in it.

* "**Scheduled.Date**" – Date when the flight was scheduled to take place.
* "**Scheduled.Time**" – The scheduled ‘*In-block’* time for arriving flights and the scheduled *‘off-block*’ time for departure flights.
* "**Stand.Date**" – The actual day when the flight arrived or departed at the stand.
* "**Stand.Time**" – The actual time when the flight arrived or departed at the stand.
* "**Actual.Date**" – The actual day when the flight was on the runway.
* "**Actual.Time**" – The actual time when the flight was on the runway.
* "**A.D**" – An indicator if the flight was an arrival or departure. A meant arrival and D meant departure.
* “**Flight.No**" – The flight number.
* "**Airline**" – The airline associated with the flight.
* "**Terminal**" – The terminal at which the flight was occurring.
* "**O.D**" – The origin/destination of the flight denoted by the 3-letter IATA code of the airport.
* "**Aircraft.General**" – The general aircraft type
* "**Aircraft.Specific**" – The specific aircraft type
* "**Seats**" – The number of seats on the plane.
* "**Total.Pax**" – The total number of passengers.
* "**Terminal.Pax**" – The number of terminal passengers.
* "**Direct.Pax**" – The number of direct passengers.
* "**Transfer.Pax**" – The number of passengers connecting through the airport onto another flight.
* "**Transit.Pax**" – The number of passengers who do not leave the plane but connect to another destination.
* "**Load.Factor**" – The percentage ratio of total passengers to seats.
* "**Transfer.Share**" – The percentage of transfer passengers on the flight.
* "**Flight.Type** – An indicator of the status of the flight. 1 is for scheduled flights and 0 for otherwise.
* "**Reg.No**" – The aircraft registration number. This is a unique identifier of the aircraft.
* "**ICAO.Code**" – A letter from *‘A’* to *‘F’* that indicates the size of the aircraft depending on its wingspan.
* "**Actual.Timestamp**” – The *Actual Date* + *Actual Time*
* "**Scheduled.Timestamp**" – The *Scheduled Date* + *Scheduled Time*
* "**Stand.Timestamp**" – The *Stand Date* + *Stand Time*

### 3.1.2 *Airport Details*

It is important to know where a flight is arriving from or departing to. This is because the SAP requires the flights to be differentiated between ‘*Domestic’* and *‘International’* and this affects the flights' compatibility with some stands.

As seen in section 3.1.1, the flight schedule data did not have enough information to identify if a flight was domestic or international. To deal with this issue, a second dataset was added to the initial data. This was a dataset that included details of the airports corresponding to the 11,987 unique IATA codes.

The data elements included in this dataset were as follows:

* The IATA code of the airport
* The ICAO code of the airport
* The complete and shortened names of the airport
* The city and country of the airport
* The longitude and latitude positions of the airport.

A sample of the airport details dataset can be seen in Table-A2 in Appendix I.

### 3.1.3 *Ethical Approval*

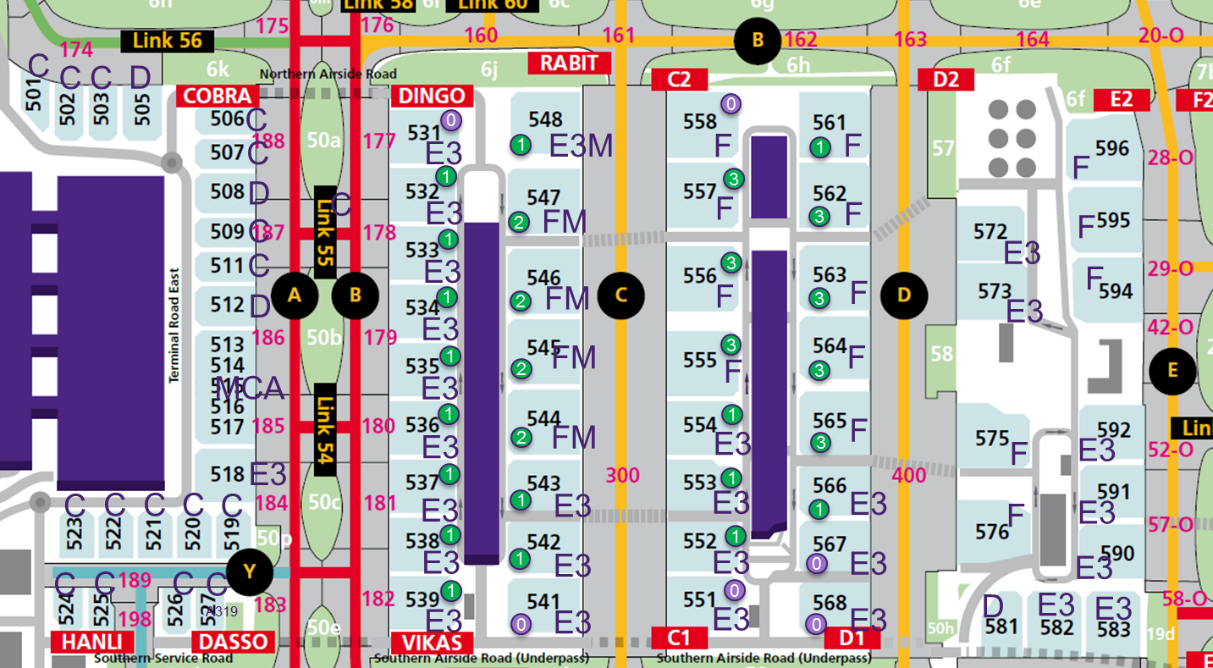
The CSV files of all the data used in the project were provided by Arup. These data are commercially sensitive hence a confidentiality agreement was signed to prevent sharing it. Consideration was given to ethics approval for this research and no application to ERGO was found necessary

## 3.2 Heathrow Airport – Terminal 5

The scope of this project was limited to Terminal 5 at Heathrow Airport. Here, the layout of the Terminal 5 apron is described along with the restrictions and assumptions on airport operations.

This information was a simplified version of the “Heathrow Stand Planning Assumptions (2017)” used in industry and was provided by Arup.

### 3.2.1 *Stand Layout*



**T5A**

**T5B0**

**T5C**

*Figure 3.2.1(i) shows a diagram of the layout of the stands at Terminal 5.*

Figure 3.2.1(i) shows a diagram of the layout of the stands at Terminal 5. Terminal 5 apron consisted of 3 sections: T5A, T5B and T5C. Each of these sections had restrictions on the types of aircraft that they could accommodate – these were based on the ICAO code of the aircraft.

The project was limited to only using flights with ICAO codes **C, D, E** and **F** were considered.

**T5A**

* All the stands in this section were code **C** and **D** stands and could accommodate *all aircraft in the code C and D fleets.*
* The stand labelled **MCA** could either be used to park ‘*1 D-code and 2 C-codes’* or ‘*2 E-code’* aircraft.

**T5B & T5C**

* The stands labelled E3 and EM were all code **E** stands and were able to accommodate *all aircraft in the code E fleet.*
* Thecode **F** stands could accommodate *all aircraft in the code F fleet*.
* All of the stands in these sections were large enough to be compatible with **C**-code and **D**-code aircraft.

**Remote Stands**

* The section with **570 – 581**, as seen on figure 3.2.1(i), were remote stands.
* All other stands were excluded from this project.

### 3.2.2 *Operational Assumptions*

Listed below are all the assumptions and restrictions that were applicable to the flights at Terminal 5.

**Domestic and CTA flights**

Flights considered as domestic were those arriving from and departing to, the United Kingdom. CTA flights were flights that were travelling to or from Dublin, Ireland. The service to these flights was limited to specific stands within Terminal 5:

* **Domestic** **arrivals** could only be served by stands **501-503** and **505-507**.
* **Domestic** **departures** were only compatible with the stands **501-503, 505-511, 519, 522** and **523**.
* **CTA arrivals** could only be parked at stand **523**.

**Contact and Remote Stands**

* Since the domestic flights and CTA arrival flights were restricted to be served by the stands mentioned above, an allocation to **any other stand** was *treated as a remote arrival/departure.*
* All other flights were treated as *pier-served flights.*
* However, there were some remote stands within sections T5A, T5B and T5C. **Any** departures or arrivals from these stands were *treated as remote:*

**T5A**: 524 – 527

**T5B**: 531, 541 and 548

**T5C**: 551, 558, 568 and 567

* The stands **572 – 581**, as mentioned before, were remote stands and *all flights occurring at stands were remote*.

**Buffer Times**

The buffer times used were derived from the British Airways (BA) rules in 2017. There were 2 sets of buffer times for the smaller aircraft and the larger aircraft.

* **C-code** and **D-code** aircraft:
  + **25 minutes** between flights
* **E-code** and **F-code** aircraft:
  + **30 minutes** between flights

**Towing Rules and Considerations**

Similar to the buffer times, the rules for towing operations were also dependent on the size of the aircraft. Listed below are the durations for enplaning/deplaning before and after towing the aircraft.

* **C-code** and **D-code** aircraft:
  + **Only overnight** flights – i.e. the aircraft that were parked at the airport through the night – were towed.
  + C-code: **45 minutes** before tow-off | **45 minutes** after tow-back.
  + D-code: **60 minutes** before tow-off | **60 minutes** after tow-back.
* **E-code** and **F-code** aircraft:
  + Flights with turnarounds greater than **8 hours** were towed off.
  + E-code: **90 minutes** before tow-off | **100 minutes** after tow-back.
  + D-code: **100 minutes** before tow-off | **120 minutes** after tow-back.
* The *time taken for towing* was assumed to be **0 minutes.**

### 3.2.3 *Training Data*

The training data used to test the feasibility of the SAP model was the flight schedule from the date: **21st July 2017**.

## 3.3 Defining the Problem

There were 2 primary objectives for the project. The first was a goal for the data manipulation stage – to create a set of useful data for the SAP. The second was the objective for the SAP. These 2 objectives were as follows:

1. *Automate the processing of the flight schedule data to calculate the flight turnaround times using Python.*
2. *Define and implement a heuristic to solve an SAP that maximised the number of passengers passing through pier-served stands.*

The project also aimed to address these questions:

* Was the Python model, for the data manipulation, feasible when dealing with flight data for a year?
* Had all the flights been allocated to a stand?
* Were the allocated stands compatible with the aircraft?
* Had potential delays or early arrivals of flights been accounted for? (Buffer times)

## 3.4 Data Manipulation

Once all the initial data were received, it was important to clean the data and process them into a format that was useful for the SAP.

### 3.4.1 *Data Cleaning*

Data cleaning is the process of detecting any incomplete, inaccurate or irrelevant data, in a dataset, and removing or correcting them.

For this project, only the **scheduled flight times** were considered. This meant that the actual times when the flights were at the stand or runway were ignored. In addition to that, the initial flight schedule dataset had data elements with irrelevant or repeated information. Therefore, the data elements removed from the **flight schedule dataset** were:

* The actual times and dates when the flight was at the stand

(*'Stand.Date', 'Stand.Time', 'Stand.Timestamp’*)

* The actual times and dates when the flight was at the runway (*'Actual.Date', 'Actual.Time', 'Actual.Timestamp'*)
* The indicator of the flights’ status (*'Flight.Type'*) due to irrelevance.
* The repeated information about the scheduled flight times (*'Scheduled.Date', 'Scheduled.Time'*)

Once these data were cleaned, the information of when the flight arrivals and departures occurred was only given by the ***'Scheduled.Timestamp'****.*

As introduced in section 3.1.2, the **airport details dataset** included various pieces of information about the airports, however, not all of this information was needed to help solve the SAP. The important data elements extracted from this dataset were:

* The IATA code of the airport (*‘AIRPORT\_IATA\_CODE'*)
* The city and country of the airport

(*'AIRPORT\_CITY', 'AIRPORT\_COUNTRY’*)

Lastly, it was important the training data – the schedule for 21st July 2017 – was extracted from the dataset.

This produced the final versions of the data: (1) a daily flight schedule for the 21st and (2) the suitable airport details. These were then further processed to create input data for the SAP.

### 3.4.2 *Calculating Flight Turnarounds*

The turnaround of a flight refers to the total time an aircraft spends at the airport, between its arrival and departure. The equation for calculating the turnaround time of a flight is:

**Eq1:**

The final flight schedule dataset was a table that listed the arrival and departure flight details for each aircraft in separate rows. This dataset was used to create a “**Turnarounds**” table. Unlike the flight schedule, the “Turnarounds” table displayed the information of the arrival and departure flights, of each aircraft, in the same row.

A key feature of this table was that it accounted for the towing operations that needed to take place during the 21st of July. When the flight turnaround times were calculated, those with very large turnarounds were split. This splitting process is further described in section 2.1. The turnarounds table allowed the 3 towing operations (enplaning, waiting and deplaning) to be viewed separately. It had 2 empty columns that were used to record the arrival and departure stands for each aircraft. It also included information about the city and country of the arrivals/departures.

The turnarounds table is the most important dataset for airport planners. This is because this table holds all the data required for solving SAPs, or other airport procedures, in one place. This was also the case here as one of the main objectives of the project was to automate the creation of the flight turnarounds dataset.

### 3.4.3 *Modelling the Data Manipulation*

Cleaning the initial data and creating the turnarounds dataset were carried out through an automated model. This model was built using the programming language **Python**. The decision to use Python was made following the request from Arup. The Python module **“Pandas”** and **“NumPy”** were used to conduct the mathematical processing.

#### **3.4.3a – Model for Data Cleaning**

There were 2 functions in the Python model that dealt with the cleaning of the initial data.

The first was **“Load\_File(***Schedule\_file, Airport\_file***)”**. This function took 2 inputs – the file names of the flight schedule and airport details datasets – and read these files onto 2 *‘dataframes’* in Python. This was also the function that carried out the removing of irrelevant data and extracted the useful ones. It was followed by the execution of the **“Select\_Data()”** function. This was to select the training data.

The Python scripts of these 2 functions are shown in Section-A1 and Section-A2 of Appendix II.

#### **3.4.3b – Model for Flight Turnarounds**

To create the “Turnarounds” dataframe, the first step was to generate 2 datasets from the final flight schedule dataframe – one for arrival flights and the other for departures. The registration numbers were used to uniquely identify the aircraft. They were then used to find and match up details of the arrival and departure flights, for each aircraft. These data were recorded in the “Turnarounds” dataframe. The functions **“Sort\_cols()”** and **“Create\_Turns()”**, in the Python model, were responsible for this and are displayed in Section-B1 of Appendix II.

Next, the **“Turnarounds()”** function was executed in the model. This added 7 new columns to the turnarounds dataframe to include the origin city/country, destination city/country, turnaround times and the arrival/departure stands for each aircraft. The columns for the stands were left empty. The turnaround times were calculated using Eq1 and data from within the “Turnarounds” dataframe. To add the origin and destination data, the IATA codes of the arrival and departure airports were used as the unique identifier. These were matched with the IATA codes in the airport details dataframe and the details of the cities and countries were mapped onto the “Turnarounds” dataframe.

**“Turnarounds()”** function also dealt with formatting the data into a more readable initial table. Therefore, the Python script in Section-B2 in Appendix II, shows the snippet of the function where new data are calculated and added.

**Additional Data Cleaning**

Since the data of the day, 21st July, the arrival/departure times of the overnight flights were blank. The was because these occurred during the previous or next day.

The **“Splitting()”** function performed additional data cleaning to replace these missing data with the time ‘*12am*'. This indicated that, at 12 am of 21st July, the overnight flights departing in the morning were at the Terminal 5 apron. It also indicated that the overnight flights that arrived on the 21st stayed on the apron till 12 am of 22nd July.

The main purpose of the **“Splitting()”** function was to conduct the splitting process on the flight turnarounds. To do this the initial turnarounds dataframe was separated into 5 parts:

1. Aircraft that arrived during the day/night and stayed overnight
2. Aircraft that parked overnight and departed in the morning
3. Aircraft in fleet F where
4. Aircraft in fleet E where
5. Aircraft that didn’t require towing.

Once the final data cleaning was completed, 2 duplicates of each row in the first 4 dataframes were created so that each aircraft had 3 rows that separately represented the 3 towing operations. The arrival and departure times on these rows were then recalculated using the enplaning and deplaning times stated in part 3.2.2. The waiting operations were labelled with an **“R”** on their arrival stand column – this was to ensure that these were definitely allocated to remote stands.

The **“Splitting()”** function uses a combination of ‘*for-loops’* and *‘if-statements’* to perform the splitting process. A sample of code for one set of overnight flights and for fleet F flights are displayed in Section-B3 of Appendix II. The structure of the code was similar for feet E data. The last step was to execute the **“final\_Turnarounds()”** function which merged the 5 parts to create the final “Turnarounds” table. The Python script for this is in Section-B4 of Appendix II and a part of the “Turnarounds” table is shown in Table-A3 in Appendix I.

## 3.5 Formulating the SAP

Discussed below is the process of identifying the variables of the SAP and deriving the heuristic to solve it.

### 3.5.1 *Identifying Variables*

The variables used in the formulation of the SAP were based on the 2nd primary objective stated in section 3.3 – maximise the number of passengers passing through pier-served stands. There was also an aim to ensure that flights were allocated to compatible stands only.

Since the objectives were strongly focused on the size/type and compatibility of the stands, the first step was to use the concept partitioning to create sets of feasible stands for aircraft in each fleet type. The partitioning of the stands was conducted using the restrictions and assumptions described in section 3.2. The stand layout was already segmented (T5A, T5B and T5C) and the operational assumptions suggested that majority of the domestic flights were to be allocated to T5A while the international flights to T5B and T5C. As for the MCA stand in T5A, it was assumed to always hold ‘*1 D-code (513) and 2 C-codes (515, 517)*’ aircraft.

The derived sets of stands for each fleet type are listed below.

***Remote Stands***

= {524, 525, 526, 527}

= {581}

= {531, 541, 548, 551, 567, 568, 572, 573}

= {558, 575, 576}

***Pier-served Stands***

= {501, 502, 503, 506, 507, 509, 511, 519, 522}

= {523}

= {515, 517, 520, 521}

= {505, 508}

= {512, 513, 505, 508}

= {518, 532, 533, 534, 535, 536, 537, 538, 539, 542, 543, 552, 553,

554, 566}

= {555, 556, 557, 561, 562, 563, 564, 565, 544, 545, 546, 547}

The final list of variables used in the SAP was:

= Set of Flights in a day

= Sets of compatible Stands

= Set of Buffer times for fleets

= List of Domestic Flights in a day

= List of International Flights in a day

= List of flights for remote stands

= List of flights for pier-served stands.

= Set of flight Arrival times

= Set of flight Departure times

= Set of Total Passengers in each flight

### 3.5.2 *The Stand Allocation Heuristic*

The approach of the *Stand Allocation* heuristic was a combination of concepts from the Greedy heuristic and the BLS heuristic.

It allocated flights following the “first come, first assigned” policy hence the flights with the earliest arrival times were chosen greedily to be allocated first. When allocating flights, if more than one compatible pier-served stand was available, the heuristic chose to allocate to the stand that had served the highest number of passengers.

Once the *Stand Allocation* heuristic produced an initial schedule, it was checked to see if the flights in each stand adhered to the buffer times restriction. If any overlapping existed, these flights were removed from the schedule and reallocated to other feasible stands. This technique was adopted from the BLS heuristic; as discussed in chapter 2.

The procedure of the *Stand Allocation* heuristic segregated the flight data by fleet type (F, E, D, C) and made allocations one fleet at a time. This was because the compatibility of stands depended strongly on the size of the aircraft. The *Stand Allocation* heuristic was applied to flights in decreasing order of aircraft size - the largest aircraft (F) first and the smallest (C) last. This decision was made because stands that could facilitate larger aircraft were big enough to hold smaller aircraft but not vice versa. Therefore, when the heuristic was making reallocations, flights of smaller aircraft that didn’t have room in their own stands could be parked at larger stands.

In industry, airport planners create the stand allocation schedules on a daily basis due to the high probability of changes in the flight schedule. For this reason, the *Stand Allocation* heuristic was also designed to work with daily flight data.

The detailed steps of the *Stand Allocation* heuristic, for assigning flights in each fleet type, are described below.

The procedure for the flights of F-code aircraft.

From , filter the flights of F-code aircraft.

**For F-code aircraft:**

* **Get** the list of flights for remote stands
* **Get** the list of flight for pier-served stands

**Allocate** the 1st flight in to 1st stand in

**For** the rest of flights in :

* **Sort** the stands in in descending order of the total passengers served
* **For** with the largest , **If** the is **true:**

( sees if is free for 30minutes before the arrival of flight . i.e. )

* **Allocate**  to
* **Else: go to** next
* Repeat until a feasible is found and is allocated
* If no feasible is found, leave unallocated

**Then**

**Allocate** the 1st flight in to 1st stand in

**For** the rest of the flights in :

* **Sort** the stands in in descending order of
* **For**  with the largest , **If** the is **true:**
* **Allocate**  to
* **Else: go to** next
* Repeat until a feasible is found and is allocated
* If no feasible is found, leave unallocated

The procedure for the flights of E-code aircraft.

From , filter the flights of E-code aircraft.

**For E-code aircraft:**

* **Get** the lists and

**Allocate** the 1st flight in to 1st stand in

**For** the rest of flights in :

* **Sort** the stands in in descending order of the total passengers served
* **For** with the largest , **If** the is **true:**

( sees if is free for 30minutes before the arrival of flight . i.e. )

* **Allocate**  to
* **Else: go to** next
* Repeat until a feasible is found and is allocated
* **If** no feasible is found, leave **unallocated**

**Detect** unallocated in :

* **Reallocate** to stands in

**Then**

**Allocate** the 1st flight in to 1st stand in

**For** the rest of flights in :

* **Sort** the stands in in descending order of
* **For**  with the largest , **If** the is **true:**
* **Allocate**  to
* **Else: go to** next
* Repeat until a feasible is found and 𝑒 is allocated
* **If** no feasible is found, leave **unallocated**

**Detect** unallocated in :

* **Reallocate** to:
  + Stands in
  + Stands in
  + Stands in

The procedure for the flights of D-code aircraft.

From , filter the flights of D-code aircraft.

**For D-code aircraft:**

* **Get** the lists and

1. **Allocate** the 1st flight in to 1st stand in

**For** the rest of flights in :

* **Sort** the stands in in descending order of the total passengers served
* **For** with the largest , **If** the is **true:**

( sees if is free for 30minutes before the arrival of flight . i.e. )

* **Allocate**  to
* **Else: go to** next
* Repeat until a feasible is found and is allocated
* **If** no feasible is found, leave **unallocated**

1. **Detect** unallocated and **remove** infeasibly allocated in :

* **Reallocate** those to:
  + Stands in
  + Stands in

**Then**

1. **For**  in

**Follow** the allocation process in step 1 to:

* **Allocate** domestic to in
* **Allocate** international to in

1. **Detect** unallocated and **remove** infeasibly allocated in :

* **Reallocate** those to:
  + Stands in
  + Stands in
  + Stands in

The procedure for the flights of C-code aircraft.

From , filter the flights of C-code aircraft.

**For C-code aircraft:**

* **Get** the lists and

1. **Allocate** the 1st flight in to 1st stand in

**For** the rest of flights in :

* **Sort** the stands in in descending order of the total passengers served
* **For** with the largest , **If** the is **true:**

( sees if is free for 30minutes before the arrival of flight . i.e. )

* **Allocate**  to
* **Else: go to** next
* Repeat until a feasible is found and is allocated
* **If** no feasible is found, leave **unallocated**

1. **Detect** unallocated and **remove** infeasibly allocated in :

* **Reallocate** those to:
  + Stand in all pier-served C-code stands
  + Stand in
  + Stand in

**Then**

1. **For**  in

**Follow** the allocation process in step 1 to:

* **Allocate** CTA arrival to in
* **Allocate** domestic to in
* **Allocate** international c to in

1. **Detect** unallocated and **remove** infeasibly allocated in :

* **Reallocate** those to:
  + Stand in
  + Stand in
  + Stand in
  + Stand in

To get the stand allocation schedule for the flights of a whole day, the *Stand Allocation* heuristic performed all the fleet-specific procedures. For the monthly schedule was achieved by creating the daily schedule, on loop, for every day of the month.

The details of these procedures were as follows:

**The Daily Stand Allocation Heuristic**

**For** the daily flight turnarounds:

1. Perform the allocation procedure for **F-code** aircraft
2. Perform the allocation procedure for **E-code** aircraft
3. Perform the allocation procedure for **D-code** aircraft
4. Perform the allocation procedure for **C-code** aircraft

**The Monthly Stand Allocation Heuristic**

**For each day** of the monthly flight schedule:

* **Create** the daily flight turnarounds
* **Perform** the Daily Stand Allocation heuristic

### 3.5.3 *Modelling the Stand Allocation Heuristic*

The model of the Stand Allocation Heuristic was created using Python. The programming language, Python, was used, again, following the request made by Arup. The created model had 5 main classes – 4 classes that had functions to allocate flights from each fleet type and 1 class that had functions to create the final daily or monthly stand allocation schedules.

**Class 1: F-Code Flights**

The flights using F-code aircraft were allocated to stands by the function **Schedule\_F.** This function ran these 3 sub-functions:

1. **F\_fl** – Responsible for extracting flight turnarounds data for F-code aircraft and defining the sets of stands compatible with the code F fleet.
2. **Remote\_F** – All the overnight parking was allocated to the remote F-stands, first. This ensured that the pier-served stands were free for the daily stand operations.
3. **Pier\_F** – Assigned the rest of the arrival/departure stand operations to the pier-served F-stands. Any of the operations that didn’t fit the schedule, were assigned to remote F-stands.

The details of the annotated Python scripts, for these functions, are given in Section-C1 of Appendix II.

**Class 2: E-Code Flights**

**Schedule\_E** was the function for allocating all the E-code aircraft and it ran the following sub-functions:

1. **E\_fl** – Responsible for extracting flight turnarounds data for E-code aircraft and defining the sets of stands compatible with the code E fleet.
2. **allo\_checker** – This function checked if there were any infeasible allocations being made, and if there were, those allocations were removed from the schedule. This function was used for the allocation of both E- and D- code flights.
3. **Remote\_E** – All the overnight parking was allocated to the remote E-stands to that the pier-served E-stands were free for the daily stand operations.
4. **Remote\_Eleft** – Checks for overnight parking that had not been allocated to the remote stands and assigns them to a pier-served E- or F-code stand.
5. **Pier\_F** – Assigned the rest of the arrival/departure stand operations to the pier-served E-stands.
6. **E\_to\_FPier** & **E\_to\_ERemote** & **E\_to\_FRemote** – Uses the allo\_checker function to find infeasible allocations at the pier-served E-stands and any stand operations that were not assigned any stand. Then reassigns them to any feasible E- or F-code stands.

The annotated Python scripts for fleet E allocation functions are given in Section-C2 of Appendix II.

**Class 3: D-Code Flights**

**Schedule\_D** assigned all the D-code aircraft to feasible D-, E- and F-code stands. This was achieved by the following sub-functions:

1. **D\_fl** – Responsible for extracting flight turnarounds data for D-code aircraft and defining the sets of stands compatible with the code D fleet.
2. **Remote\_D** – All the overnight parking was allocated to the remote D-stands.
3. **RemoteD\_to\_FPier** & **RemoteD\_to\_EPier** – Checks for overnight parking that had not been allocated to the remote stands and assigns them to a pier-served E- and F-code stand.
4. **UKArr\_D** – The stand operations arriving from a domestic origin were assigned to only stands accommodating the parking of domestic arrival flights.
5. **UKDep\_D** – The stand operations departing to domestic destinations were assigned to the stands that were able to accommodate the domestic departures.
6. **Int\_D** – Assigned all the international flights to any of the pier-served D-code stands. (except those restricting international arrivals)
7. **PierD\_to\_EPier** & **PierD\_to\_FRemote** – Uses the allo\_checker to find infeasible or incomplete allocations made to any pier-served D-stand. They were removed and reassigned to any available, feasible E- and F- code stands.

Section-C3 of Appendix II shows all the annotated Python scripts for these functions.

**Class 4: C-Code Flights**

**Schedule\_C** was the main function thatassigned all the C-code aircraft to feasible C-, D-, E- and F-code stands. This was achieved by the following sub-functions:

1. **C\_fl** – Extracted the flight turnarounds data for C-code aircraft and defined the sets of stands compatible with the code C fleet.
2. **C\_allo\_checker** – Like the ‘allo\_checker’ function, this checks for any infeasible or incomplete (failed) allocations that were made to any of the C-code stands. A checking function was made specifically for the code C fleet because it had the greatest number of flights and was compatible with stands of all sizes.
3. **Remote\_C** – All the overnight parking was allocated to both remote and pier-served C-code stands.
4. **RemoteC\_to\_FPier** & **RemoteC\_to\_EPier** – Identifies all the overnight parking that had not been allocated, or incorrectly allocated, to C-code stands. Then assigns them to a pier-served E- and F-code stand.
5. **CTA\_C** – The flights arriving from Dublin were restricted to only one pier-served stand. This function allocated the CTA arrivals to that stand with priority before non-CTA arrival/departure stand operations.
6. **UKArr\_C** – The stand operations arriving from UK origins were assigned to only C-code stands accommodating the parking of domestic arrival flights. The ‘C\_allo\_checker’ function was then used on the allocations made. The infeasible/incomplete allocations were then allocated to both remote and pier-served F-, E- and D-code stands – all of these were considered remote.
7. **UKDep\_C** – Had the same functionality as ‘UKArr\_C’, except it was applied to the stand operations that were departing to a domestic destination.
8. **Int\_C** – Assigned all the international flights to any of the pier-served C-code stands. (except those restricting international arrivals). The ‘C\_allo\_checker’ function was applied to the allocations and the detected infeasible/incomplete allocations were reassigned to both remote and pier-served F-, E- and D-code stands but they were all

Section-C3 of Appendix II shows all the annotated Python scripts for these functions.

**Class 5: The Daily and Monthly Allocation Schedules**

There were 2 main functions that there defined in this class. The **Daily\_Sched** and **Monthly\_Sched** produced the daily and monthly allocation schedules for the flights, respectively.

The ‘Daily\_Sched’ function took the day of the date and a string input. It then ran the ‘Schedule\_F’, ‘Schedule\_E’, ‘Schedule\_D’ and ‘Schedule\_C’ functions, from the other classes, in this order. The largest aircraft were assigned first and smallest last. The completed flight turnarounds table, with the stand allocations, was then saved as a .CSV file.

The ‘Monthly\_Sched’ function took the numerical value of the month as an input and used it to extract the flight schedule data for that month. Then, the function looped over the dataset, to create a turnarounds table and apply the ‘Daily\_Sched’ function to every day of the month. The allocation schedules for each day of the month were then saved as .CSV files in a folder, labelled by the month.

The Python scripts for these 2 functions are shown in Section-D1 and Section-D2 in Appendix II.

The general code that is used for the allocations of each fleet type was the same. The parts that varied were the fleet specific constraints like the buffer times and the feasible sets of stands. Also, the specifics of which stands can accept reassigned flights – i.e. only small aircraft can be reassigned to larger stands, not vice versa. Described below are some of the key parts of the Python model that perform the allocation heuristic.

* Calculate the total number of passengers served at each used stand and sort the stands from largest to smallest total:

*#calculate the sum of all number of passengers served#*

df = Turnarounds.groupby('Stand\_Arrive').apply(lambda g:\

g.assign(col1\_sum=g.Total\_Pax\_Arrive.sum()))

df = df.sort\_values(by='Scheduled\_Timestamp\_Arrive',ascending= True)

*# create set of stands sorted from largest to smallest no.of passengers*

df1 = df.drop\_duplicates('Stand\_Arrive', keep = 'first')

df1 = df1.sort\_values(by='Scheduled\_Timestamp\_Depart',\

ascending=True).reset\_index()

* The greedy select the stand starting with the one that had largest number of passengers served:

for (j, rows) in df1.iterrows(): *#for each stand is set df1*

stand = df1['Stand\_Arrive'][j]

* Ensure that the buffer times are added:

*#for C- and D- code aircraft*

Buffer = Turnarounds.loc[i, 'Scheduled\_Timestamp\_Arrive'] \

- Turnarounds.loc[Turnarounds.index[f],'Scheduled\_Timestamp\_Depart']\

>=pd.to\_timedelta('25 minutes')

*#for E- and F- code aircraft*

Buffer = Turnarounds.loc[i, 'Scheduled\_Timestamp\_Arrive'] \

- Turnarounds.loc[Turnarounds.index[f],'Scheduled\_Timestamp\_Depart']\

>=pd.to\_timedelta('30 minutes')