RECO – Recherche opérationnelle

Airport operations management assistance Project

Attachment Document

BONATI Michelle – MONTANARI Filippo

All the technical aspects of the four solved problems are explained in detail in this document. For each solution we added all the information that seemed necessary in order to understand the implemented features.

Summary

1. Planning the schedule of check-in agents

VBA Code

The code of problem 1 mainly concerns the visual formatting of data.
NB: to integrate the solver in a VBA code, the solver reference must be added to the excel VBA section. Otherwise, the VBA compiler won't recognize the solver commands.

```
Solution_problème_1ab() – Solution_problème_1c()
```

These algorithms copy the input data into the solver table, then launch the solver and return the results in the results table.

```
diagramme_1ab() – diagramme_1c()
```

These algorithms save all the meaningful solver results into a matrix. After that, they can format the cells that are going to host the diagram and trace the graphic results inside them.
There is no risk of overwriting the upper cells, since the formatting is sensitive to input data, and the algorithms can calculate the number of cells that are going to be needed.

```
effacer_diagramme_1ab() – effacer_diagramme_1c()
```

These algorithms recognize the space occupied by the diagram and clear and resize all the in-volved cells.

Solver

Supporting table for the solver in problem 1a:

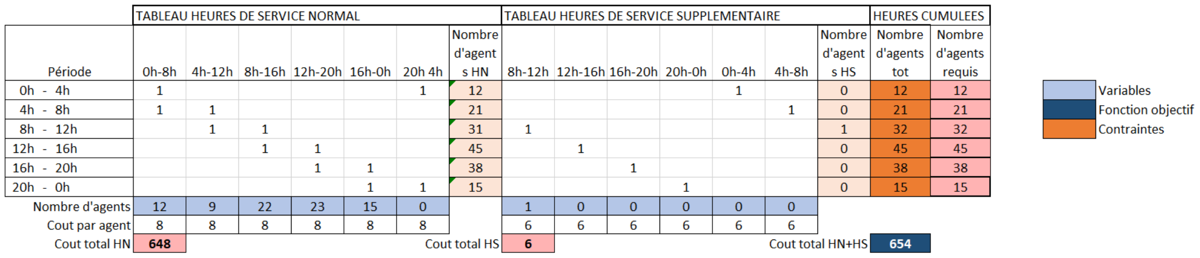


	TABLEAU HEURES DE SERVICE NORMAL							TABLEAU HEURES DE SERVICE SUPPLEMENTAIRE							HEURES CUMULEES	
Période	0h-8h	4h-12h	8h-16h	12h-20h	16h-0h	20h 4h	Nombre d'agents HN	8h-12h	12h-16h	16h-20h	20h-0h	0h-4h	4h-8h	Nombre d'agents HS	Nombre d'agents tot	Nombre d'agents requis
0h - 4h	1					1	12					1		0	12	12
4h - 8h	1	1					21						1	0	21	21
8h - 12h		1	1				31	1						1	32	32
12h - 16h			1	1			45		1					0	45	45
16h - 20h				1	1		38			1				0	38	38
20h - 0h					1	1	15				1			0	15	15
Nombre d'agents	12	9	22	23	15	0		1	0	0	0	0	0			
Cout par agent	8	8	8	8	8	8		6	6	6	6	6	6			
Cout total HN	648						Cout total HS	6						Cout total HN+HS	654	

🟦	Variables
🟦	Fonction objectif
🟧	Contraintes

Supporting table for the solver in problem 1b:

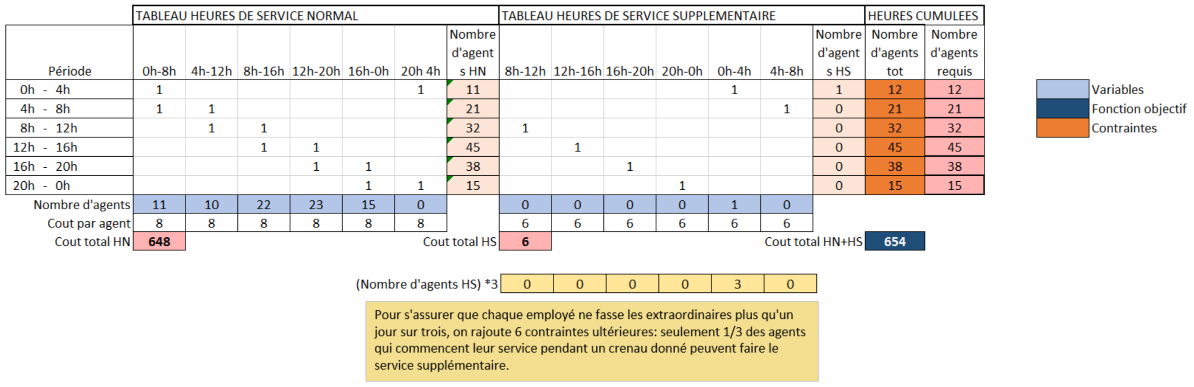
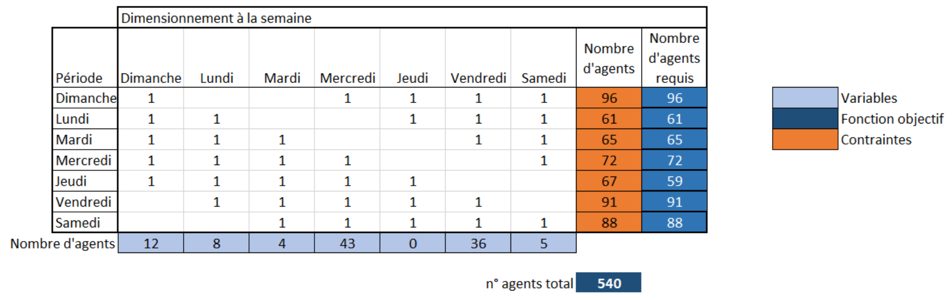


	TABLEAU HEURES DE SERVICE NORMAL							TABLEAU HEURES DE SERVICE SUPPLEMENTAIRE							HEURES CUMULEES	
Période	0h-8h	4h-12h	8h-16h	12h-20h	16h-0h	20h 4h	Nombre d'agents HN	8h-12h	12h-16h	16h-20h	20h-0h	0h-4h	4h-8h	Nombre d'agents HS	Nombre d'agents tot	Nombre d'agents requis
0h - 4h	1					1	11					1		1	12	12
4h - 8h	1	1					21						1	0	21	21
8h - 12h		1	1				32	1						0	32	32
12h - 16h			1	1			45		1					0	45	45
16h - 20h				1	1		38			1				0	38	38
20h - 0h					1	1	15				1			0	15	15
Nombre d'agents	11	10	22	23	15	0		0	0	0	0	1	0			
Cout par agent	8	8	8	8	8	8		6	6	6	6	6	6			
Cout total HN	648						Cout total HS	6						Cout total HN+HS	654	
							(Nombre d'agents HS) *3	0	0	0	0	3	0			

🟦	Variables
🟦	Fonction objectif
🟧	Contraintes

Pour s'assurer que chaque employé ne fasse les extraordinaires plus qu'un jour sur trois, on rajoute 6 contraintes ultérieures: seulement 1/3 des agents qui commencent leur service pendant un crenau donné peuvent faire le service supplémentaire.

Supporting table for the solver in problem 1c:



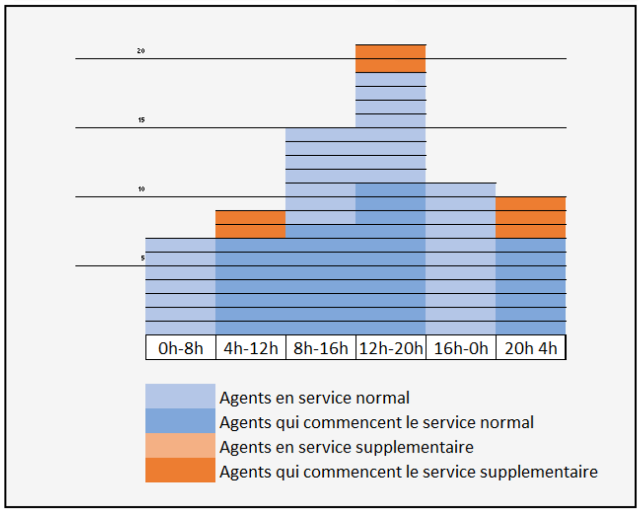
	Dimensionnement à la semaine								
Période	Dimanche	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Nombre d'agents	Nombre d'agents requis
Dimanche	1			1	1	1	1	96	96
Lundi	1	1			1	1	1	61	61
Mardi	1	1	1			1	1	65	65
Mercredi	1	1	1	1			1	72	72
Jeudi	1	1	1	1	1			67	59
Vendredi		1	1	1	1	1		91	91
Samedi			1	1	1	1	1	88	88
Nombre d'agents	12	8	4	43	0	36	5		

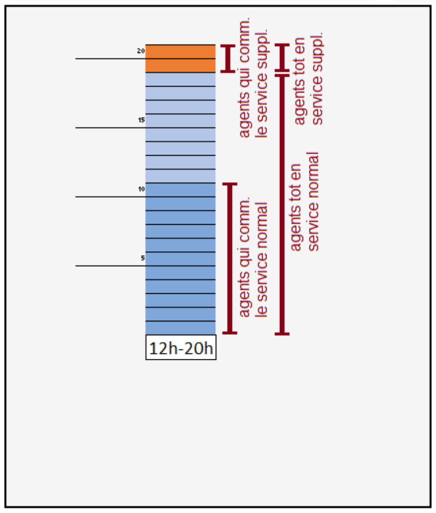
n° agents total 540

🟦	Variables
⬛	Fonction objectif
🟧	Contraintes

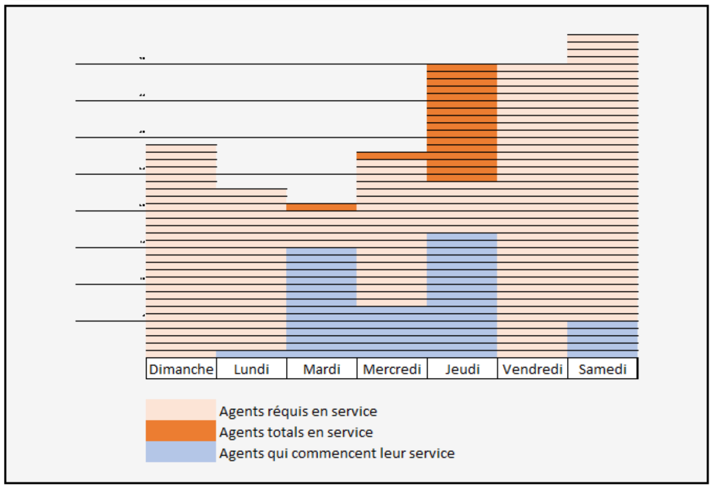
Notes

The meaning of the diagrams of problems 1a and 1b (how to read them) is shown below:




20
15
10
5
12h-20h
agents qui comm. le service suppl.
agents tot en service suppl.
agents qui comm. le service normal
agents tot en service normal

The meaning of the diagram of problem 1c (how to read it) is shown below:



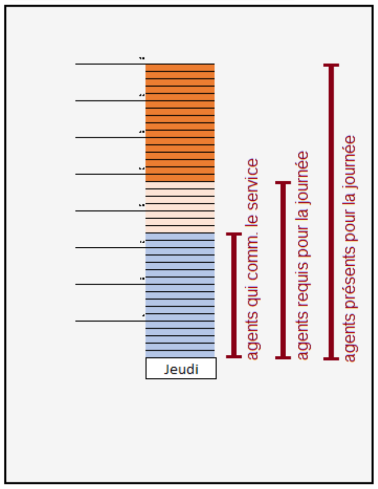

Jeudi
agents qui comm. le service
agents requis pour la journée
agents présents pour la journée

Fig.5 How to read the diagram of the number of agents in problem 1c

2. Assignment of crews to flights

VBA code

For problem 2, only the most meaningful codes will be listed and explained.

`Affectation_des_equipages()`

This code runs all the different pieces of code in sequence, so as to create the Gannt chart of the flights, write the adjacency matrix, find the flight sequences, trace them in their Gantt chart, and finally launch the solver.
Finally, the results of the solver are saved and converted into a visual output on the user interface. The formatting of the results table is sensitive to the number of crews found by the solver.

Other associated pieces of code:

- `Reset_tableau_des_resultats_dans_le_feuille_dinput()`

`GanttVols()`

Given the flights data, the algorithm traces the Gantt chart of the flights. This code can deal with any number of input flights.
This code presented a problem when it was asked to make comparisons that involved the hour 11:00. in fact, in a first version, it didn't correctly trace the first Gantt cell of the flights that started at 11:00. This was probably due to conversion problems, and to solve it we added a `±0.001` to the comparisons:

```
If Gantt hour + 0.001 >= departure time And
Gantt hour < arrival time - 0.001 Then
   Color the cell
End If
```

Pseudo-code:

```
Count the number of flights
For each flight
   For each time cell of the Gantt
      If [the time on the GANTT chart is later or equal to the departure
      time in the input table] And [the time on the GANTT chart is earlier
      than the arrival time in the input table] Then
         Mark the Gantt cell
      End If
   Next
Next
```

Other associated pieces of code:

- Code integrated in the sheet, that adjusts the size of the input table whether the user changes the number of flights that he/she wishes to insert.
- `Copier_les_donnees_dans_la_feuille_de_diagramme_Gantt()`
- `EffacerDonnes()`
- `EffacerGantt()`

`MatriceAdjacence()`

Given the flights data, the algorithm traces the adjacency matrix of the flights. This code can deal with any number of input flights.

Pseudo-code:

```
Count the number of flights
For each flight i
   For each possible pairing flight j
      If [the pairing flight j is different from flight i] And [the
      departure airport of flight j is equal to the arrival airport of
      the flight i] And [the departure time of the flight j is later than
      the arrival time of flight i] Then
         Write 1 into the cell
      End if
   Next
Next
```

Other associated pieces of code:

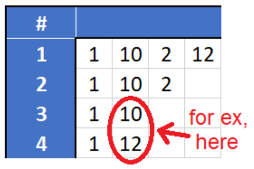
- Code integrated in the sheet, that adjusts the size of the input table whether the user changes the number of flights that he/she wishes to insert.
- `Copier_les_donnees_dans_la_feuille_de_la_matrice_dadjacence()`
- `EffacerDonnes()`
- `EffacerMatrice()`

`tableau_séquences()`

Given the flights data and the adjacency matrix, the algorithm finds all the possible sequences and adds them to the dedicated sequence table. The code can potentially deal with any number of flights and sequences, but the formatting only suits 105 sequences and a maximum of 10 flights in any sequence.
Moreover, the algorithm uses a 'graphical method' to close the explored path and continuously jumps from a sheet to the other. This results in a rather slow execution.
A better version of the algorithm would save the adjacency matrix in a variable matrix, drastically reducing the execution time.

Pseudo-code:

```
Count the number of flights n
line = 1 'line of the adjacency matrix = departure flight
While flight line <= n + 1
   While there are still non-explored path starting from the current line
      i = line
      j = 1 'column of the adjacency matrix = arrival flight
      While the column index j is lesser than n + 1
         If the current cell i,j contains 1 And it has not a grey back-
         ground Then
            Save the column index j in an array (provisional sequence)
            i = j 'fundamental step: we set the line index i equal to the
            column index j of the column in which we found 1
            If in the previous sequence there isn't the
            same flight number j that we just added to
            the current one in the same position Then
               Unlock the closed cells that start with
               the same flight number of the current one
         End If
         Else : j = j + 1 'next column = next arrival flight
         End If
      'when j = n + 1 we exit the loop, because that means that we found
      a void line and there are no flight starting from the current one
      Wend
```



#				
1	1	10	2	12
2	1	10	2	
3	1	10		
4	1	12		

```
      Close the last visited cell -> grey background
      Write the provisional sequence in the sequence table
      Check if there are other paths to follow starting from the current
      line
   Wend
   line = line + 1 'next line = next first sequence flight
   Unlock all the closed path -> bleach all the cells
Wend
```

Other associated pieces of code:

- `EffacerTableauSequences()`

`Gantt_sequences()`

Given the flights data and the sequences table, this algorithm traces the Gantt chart of each sequence. In this case, the input table is saved in a 50x7 matrix, thus making unnecessary to continuous sheet jump. By doing this, we observed that the code is dramatically accelerated (from more than 2 minute, to less than 2 seconds).
The code can deal with 50 flights due to matrix dimension, and with 105 sequences due to formatting reasons.

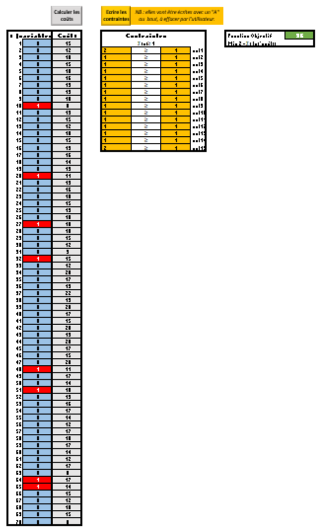
Pseudo-code:

```
Count the number of flights n
Save the input table in a matrix
Copy-paste the input table and associate to each flight a different color
Count the number of sequences m
For each sequence
   j = 1 'index of exploration of the current sequence
   While we have flights inside the sequence
      For each flight ii inside the input matrix
         If the element j of the sequence is equal to the flight number
         ii inside the input matrix Then
            For each time cell (1 to 24)
               If the departure time of the flight ii is greater or equal
               to the current time cell And the arrival time of the flight
               ii is smaller than the current time cell Then
                  Add the flight ID to the current cell and color its
                  background with the respective color
               End If
            Next
         End If
      Next
   j = j + 1 'next sequence element
   Wend
Next
```

Other associated pieces of code:

- `Vider_Gantt_Sequences()`

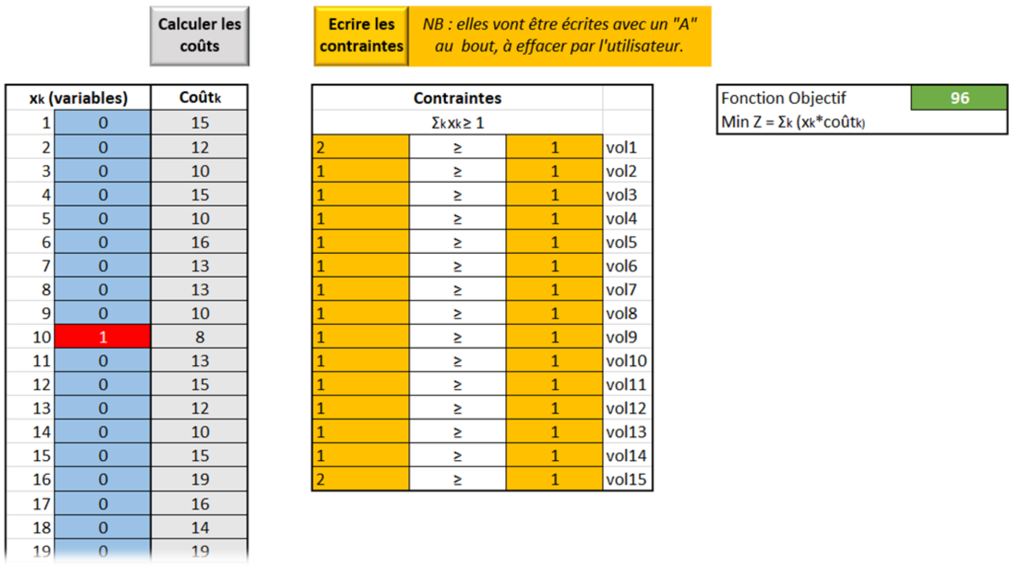
Solver



The supporting table for the solver in problem 2 is shown aside. This sheet also features:

- A VBA algorithm to calculate the cost of each sequence (5h + arrival hour of the last flight – departure time of the first flight):
  `Calculer_cout_sequences()`
- A VBA algorithm to automatically write the constraints:
  `Ecrire_les_contraintes()`
  This algorithm writes the constraint formula with a A on top, since we couldn't write into a cell in a sequential way starting with a '='. Therefore, after running the code the user will have to manually remove the A on top of each constraint formula.

The solver sheet doesn't automatically adapt to problems with a different number of flights and sequences.



Calculer les coûts

Ecrire les contraintes

NB : elles vont être écrites avec un "A" au bout, à effacer par l'utilisateur.

	x_k (variables)	Coût$_k$
1	0	15
2	0	12
3	0	10
4	0	15
5	0	10
6	0	16
7	0	13
8	0	13
9	0	10
10	1	8
11	0	13
12	0	15
13	0	12
14	0	10
15	0	15
16	0	19
17	0	16
18	0	14
19	0	19

Contraintes			
	$\Sigma_k x_k \geq 1$		
2	≥	1	vol1
1	≥	1	vol2
1	≥	1	vol3
1	≥	1	vol4
1	≥	1	vol5
1	≥	1	vol6
1	≥	1	vol7
1	≥	1	vol8
1	≥	1	vol9
1	≥	1	vol10
1	≥	1	vol11
1	≥	1	vol12
1	≥	1	vol13
1	≥	1	vol14
2	≥	1	vol15

Fonction Objectif	96
Min Z = Σ_k (x_k*coût$_k$)	

3. Ground operations management - passengers transport

VBA Code

The code of problem 3 mainly concerns the visual formatting of data.

```
Calculer_les_temps_de_passage()
```

This algorithm copies the input data into the solver table, then launches the solver and returns the results in the results table.
Moreover, it returns information about the delays of the bus.

Other associated pieces of code:

- `Effacer_les_donnees()`
- `Effacer_les_resultats()`

```
Tracer_la_sequence_de_passage()
```

Starting from the results of the results table, this algorithm graphically indicates the sequence that the bus should follow, together with the estimated arrival time and the delay for each stop.

Pseudo-code:

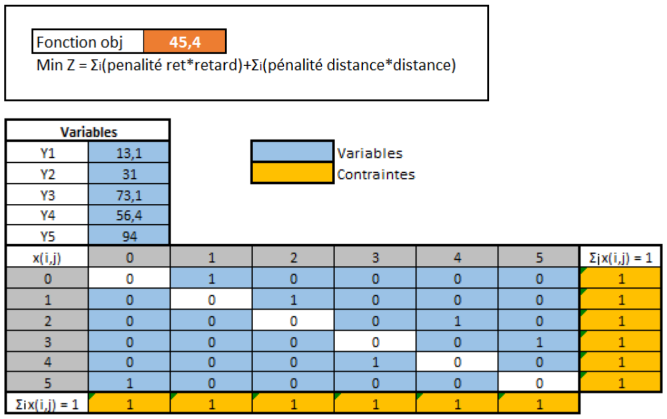
```
For each stop (plane/terminal)
   Find the stop with the lowest arrival time among those which are left
   Add the information about the chosen stop in the table of the sequence
Next
```

Solver

The supporting table for the solver in problem 3 is shown here.

In this modelling, a mistake to avoid consists in writing the time constraints also for the travels direct toward 0 (the deposit).
In fact, in that case the problem has, of course, no solution, since it would be impossible for the bus to make its last stop (Y5) before the departure time (Y0).

The lines corresponding to these travels are colored in black in the constraints' table beside.



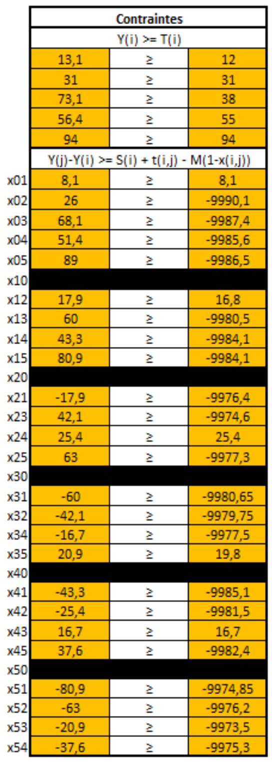
Fonction obj	45,4

Min Z = Σi(penalité ret*retard)+Σi(pénalité distance*distance)

Variables	
Y1	13,1
Y2	31
Y3	73,1
Y4	56,4
Y5	94

🟦	Variables
🟨	Contraintes

x(i,j)	0	1	2	3	4	5	Σjx(i,j) = 1
0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	1
2	0	0	0	0	1	0	1
3	0	0	0	0	0	1	1
4	0	0	0	1	0	0	1
5	1	0	0	0	0	0	1
Σix(i,j) = 1	1	1	1	1	1	1	



	Contraintes		
	Y(i) >= T(i)		
	13,1	≥	12
	31	≥	31
	73,1	≥	38
	56,4	≥	55
	94	≥	94
	Y(j)-Y(i) >= S(i) + t(i,j) - M(1-x(i,j))		
x01	8,1	≥	8,1
x02	26	≥	-9990,1
x03	68,1	≥	-9987,4
x04	51,4	≥	-9985,6
x05	89	≥	-9986,5
x10			
x12	17,9	≥	16,8
x13	60	≥	-9980,5
x14	43,3	≥	-9984,1
x15	80,9	≥	-9984,1
x20			
x21	-17,9	≥	-9976,4
x23	42,1	≥	-9974,6
x24	25,4	≥	25,4
x25	63	≥	-9977,3
x30			
x31	-60	≥	-9980,65
x32	-42,1	≥	-9979,75
x34	-16,7	≥	-9977,5
x35	20,9	≥	19,8
x40			
x41	-43,3	≥	-9985,1
x42	-25,4	≥	-9981,5
x43	16,7	≥	16,7
x45	37,6	≥	-9982,4
x50			
x51	-80,9	≥	-9974,85
x52	-63	≥	-9976,2
x53	-20,9	≥	-9973,5
x54	-37,6	≥	-9975,3

Notes

This problem was treated as schedule problem 1|r$_i$|T, in which 1 machine (the bus) must perform 6 tasks (visiting the plane/terminals/deposit) respecting availability time constraints r$_i$ and minimizing the delay T.

4. Assignment of flights to boarding gates

Comment on the constraints listed at page 2 of the paper
Lim et al., 2005, Airport Gate Scheduling with Time Windows

OBJECTIVE FUNCTION:

$$MinZ = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{l=1}^{m} f_{ij}w_{kl}z_{ijkl} + \sum_{i=1}^{n} p_i(c_i - a_i)$$

We aim at minimizing the sum of the total distance walked by all passengers of all flights, and the sum of penalties due to empty gates.

(f_{ij} = number of passengers travelling from flight i to flight j)
(w_{ij} = distance to be travelled between flights $i$and$j$)
(p_i = unit delay penality of flight i)

CONSTRAINTS (1—9) from the original article:

1)

$$\sum_{k=1}^{m} x_{ik} = 1, 1 \le i \le n$$

This constraint prevents a flight to be assigned to more than one gate.

2)

$$z_{ijkl} \le x_{ik}, 1 \le i, j \le n, 1 \le k, l \le m$$

z_{ijkl} refers to a pair of flights in which flight i is assigned to gate k, while flight j is assigned to gate l. This constraint tells us that z_{ijkl} can be equal to 1 if and only if x_{ik} is equal to 1, 0 otherwise.

3)

$$z_{ijkl} \le x_{jl}, 1 \le i, j \le n, 1 \le k, l \le m$$

This constraint tells us that z_{ijkl} can be equal to 1 if and only if x_{jl} equals 1, 0 otherwise.
Thus, summing up the last two constraints, we can conclude that in order to have $z_{ijkl} = 1$, both x_{ik} and x_{jl} must be equal to 1.

4)

$$x_{ik} + x_{jl} - 1 \le z_{ijkl}, 1 \le i, j \le n, 1 \le k, l \le m$$

On the other hand, this constraint requires that, if $x_{ik} = x_{jl} = 1$, then z_{ijkl} will also be equal to 1.

5)

$$c_i \geq a_i, 1 \leq i \leq n$$

With this constraint it has been imposed that the time in which the flight i starts occupying a gate must be greater or equal to the time in which the gate returns available. This means that the flight cannot occupy the gate unless the gate is free at the flight arrival time.

6)

$$c_i \leq b_i - d_i, i \leq i \leq n$$

This constraint is necessary to ensure that flight i starts occupying the gate in time to allow boarding operations to be completed before the departure time (i.e. without exceeding the time window limit).

7)

$$(c_i + d_i) - c_j + y_{ij}M > 0, 1 \leq i, j \leq n$$

First of all, $y_{ij}M = 0$ if and only if there is overlap between flights i and j. In addition, $(c_i + d_i)$ is the time at which flight i frees its gate.
Thus, in case there is an overlap ($y_{ij}M = 0$) between services i and j, this constraint imposes that the end hour of service i is greater than the start hour of service j. → $(c_i + d_i) > c_j$

8)

$$(c_i + d_i) - c_j - (1 - y_{ij})M \leq 0, 1 \leq i, j \leq n$$

On the contrary, $-(1 - y_{ij})M = 0$ if and only if there is no overlap between flights i and j. Thus, in case there is no overlap ($(1 - y_{ij})M = 0$) between services i and j, this constraint requires that the end hour of service i is smaller than the start hour of service j. → $(c_i + d_i) \leq c_j$

9)

$$y_{ij} + y_{ji} \geq z_{ijkk}, 1 \leq i, j \leq n, i \neq j, 1 \leq k, l \leq m$$

$z_{ijkk} = 1$ if and only if the flight i and j both occupy the gate k, while $y_{ij} = 1$ if and only if there is no overlap between the two. Thanks to this constraint, we impose to flights i and j to not occupy the same gate unless there is no overlap between them.
The constraint n°9 has been modified from the original client's requirement (on Moodle).

VBA Code

The code of problem 4 concerns the visual formatting of data, the tracing of the Gantt charts and the application of the Greedy Algorithm. The developed application can deal with any problem of the same kind, with the same number of flights and the same number of gates.

```
Gantt_flights()
```

This algorithm traces the Gantt chart of the flights by using the information contained in the input table. If the information concerning the real arrival time of a plane exists, then the algorithm traces on the Gantt both the estimated and the real time window of each flight. Otherwise, it only traces the estimated time windows.

Pseudo-code:

```
Count the number of flights
For each flight i
   Save the estimated departure and arrival time in minutes, where the
   00:00 of day 1 corresponds to 0 minutes.
   Trace the estimated time window on the Gantt chart
   If the ‘arrival time’ cell is not empty Then
      Save the actual arrival time in minutes
      Calculate the actual departure time by adding the ‘gate duration’
      to the ‘arrival time’
      If a portion of the actual time window is contained into the esti-
      mated time window Then
         Trace it in blue on the Gantt chart
      End If
      If a portion of the actual time window is not contained into the
      estimated time window Then
         Trace it in red on the Gantt chart
      End If
   End If
Next
```

Other associated pieces of code:

- `Gantt_reset()`

`Greedy_algorithm()`

This code implements the Greedy Algorithm, a heuristic method that allocates the flights to the gates returning a possible solution. It is demonstrated that the solution returned by this algorithm is also the best one.
After that, the code graphically returns the results by adding the flight into the Gantt chart of the gates.

Pseudo-code:

```
Count the number of flights
Initialize the vector in which the last occupied minute of each gate is
stored g = {0, 0, 0, 0, 0}
For each flight
   If the actual arrival time of the flight is known Then
      Set the actual arrival time in minutes as start time
      Calculate the end time by adding the ‘gate duration’ to the start
      time
   End If
   If the actual arrival time of the flight is unknown Then
      Save the estimated departure and arrival time in minutes as start
      and end time
   End If
   Initialize the gate number to which the flight will be assigned to 6
   (apron) -> GateN = 6
   For each gate k = 1 To 5
      If the gate k is the one with the longest schedule And the start
      time of the flight is greater than the g(k) value Then
         GateN = k
      End If
   Next
   If GateN < 6 Then
      Trace the flight inside the GateN line
   Else : ad the flight number to the apron
   End If
Next
```

Other associated pieces of code:

- `Effacer_Greedy_Gantt()`

Code integrated in the sheet to dynamically update the Gantt charts

This code automatically launches the two previous codes whenever a change is detected in the 'arrival time' column of the input table. The auto-update option can be disabled by erasing the '*Oui* ' from the corresponding cell.

Pseudo-code:

```
If the content of any cell of the arrival time column changes Then
   If the auto-update option cell is equal to "Oui" Then
      Call the Gantt_flights procedure
      Call the Greedy_algorithm procedure
   End If
End If
```

Other pieces of code associated to the input table:

- `Trier_vols()`
- `reset_horaires_arrivee()`

Notes

The Gantt charts graphical resolution is of 5 minutes; therefore, it is impossible to reach a graphical precision higher than 5 minutes. However, all the time comparisons are done with the real time values in minutes, and not with the rounded ones.
The role of the 5 minutes rounding is making the algorithms quicker. In fact, in this way the length of all the loops that trace the Gantt charts is divided by 5.