Solving the Airline Crew Scheduling Problem

Marion Debon

University of Houston - Downtown

1. Introduction

In the airline industry, one of the largest administrative costs after aircraft costs and fuel expenses is crew costs. In comparison to other transportation industries, airline crews receive notably higher salaries than personnel in bus transit, rail and truck transportation, therefore lowering crew costs makes a huge difference in airline spending. Considering there is an increasing competition among low-cost flights, generating the most savings is crucial, which is why it is important for the airline industry to pay large attention to crew spending.

The Airline Crew Scheduling problem aims to minimize crew costs by finding the best assignment of crews to flights. It is a set partitioning problem with many constraints and restrictions.

Airline crew scheduling is a quite challenging problem to solve for multiple reasons. Firstly, there are many components to the problem, requiring it to be solved in two independent sub-problems. The first problem consists of partitioning the flight segments into possible crew pairings, which can then be used for the second step, which assigns specific crew to those pairings. Additionally, a large number of Federal Aviation Administration (FAA) restrictions and other government regulations prevent many possible solutions from being applicable, making airline crew scheduling much complicated.

In this paper, we define the airline crew scheduling problem and its formulation as an optimization problem, and examine different solution methods for solving it, including the Hungarian Algorithm and Binary Integer Programming. Then, we use Julia to model and solve an example of the airline crew scheduling problem.

2. Terminology

Crew pairing: A sequence of duty periods including one or more work days, separated by periods of rest.

Duty periods: Series of sequential flight legs forming a crew work day.

Home base: The airport located where a crew resides.

3. The Airline Crew Scheduling Problem

Airline crew scheduling is the process of assigning crew to flights. This problem is solved in two distinct phases:

1. The *Crew Pairing* problem: this first phase consists in generating crew pairings from the flight segments covered by the airline. Then, from these possible crew pairings, we select a set of pairings that minimizes the total crew cost.

2. The *Crew Assignment* problem: this phase consists in assigning the minimum cost set of pairings selected to specific crews.

3.1. The Crew Pairing Problem

The crew pairing problem is an optimization problem with an objective to minimize crew costs by finding the right set of feasible pairings that will cover all flight segments from the airline. There are two steps to this problem.

The first step consists in generating a set of pairings that can be feasible. The feasibility of a pairing is defined by airline restrictions, government regulations and labor negotiations. Examples of pairing restrictions include:

– Flight pairings must start and end at the crew's home base.
– The time away from base must not exceed an upper bound.
– There is a maximum number of duty periods allowed in a single pairing.
– There are appropriate rest periods between each duty.

Each pairing has a specific cost that depends on the number of flight segments and the duration of the flights included. The second step of the crew pairing problem is to select a minimum-cost set of pairings that will be used; it is modelled as a set partitioning problem. In this paper, we will focus on solving the second step of the crew pairing problem, along with the crew assignment problem.

3.2 The Crew Assignment Problem

Once we are done with the crew pairing phase and a set of minimum-cost pairings have been selected, the next problem is to assign specific crew to each of these pairings. Our goal is to find the best possible assignment of crew to flight pairings.

There are a few factors contributing to which crew members are able to cover a specific flight pairing. Firstly, although they work for the same airline, all crew members have a specific home base they depart from and need to return to after their last duty period. Airlines can be based in multiple distinct cities, but each of their crew can only be assigned to flight pairings that start and end at their own home bases; this already eliminates certain assignments. Secondly, availability must be considered when assigning crew members. While a crew may be qualified for certain pairings, they might not be available based on the duration of the duty periods, or other factors such as vacation periods. Therefore, we must only assign pairings to crew who are available for them and are from the same home base.

In addition to these mandatory factors, airlines may also consider their crew's preferences to certain flight pairings. An approach known as *bidline scheduling* is commonly used by airlines in the US. It allows crew members to bid on given schedules, and their bids are used to build assignments.

4. Solving the Airline Crew Problem

4.1 Solving the Crew Pairing Problem

The crew pairing problem is non-linear with constraints that reduce the feasibility of multiple solutions; therefore, it is modelled as a set partitioning problem.

We can model the crew pairing problem as follows:

$$\text{Minimize } \sum_{i=1}^{n} c_i x_i \tag{1}$$

Subject to:

$$\sum_{i:j \in F} x_i = 1 \quad \forall j \in F \tag{2}$$

$$x_i \in \{0, 1\} \quad \forall i \in P \tag{3}$$

Where:

$$x_i = \begin{cases} 1 \text{ if pairing } i \text{ is chosen} \\ 0 \text{ otherwise} \end{cases}$$

c_i = cost of pairing i
F = The set of all flights
P = The set of all feasible pairings
j = A flight from set F

Each pairing has a constant cost c_i that will define whether a specific pairing is chosen or not. The aim of our objective function (1) is to select a minimum cost set of pairings such that each flight segment is covered in at least one pairing in constraint (2). Our second constraint (3) makes sure only one pairing of each type is chosen.

A common solution method used for solving set partitioning problems such as this model is Binary Integer Programming, (BIP) which we will use to solve the crew pairing problem.

A Crew Pairing Example

Assume an airline covers five cities: A, B, C, D, and E, with city A and C as home bases. Let the pairings shown in table 4.1.1 be feasible pairings for the airline according to rules and regulations.

Table 4.1.1: Flight Pairings with Respective Cost

	Pairings	Flights	Cost
1	A → B → A	AB, BA	55
2	A → B → C → A	AB, BC, CA	85
3	A → B → C → D → E → A	AB, BC, CD, DE, EA	101
4	A → C → A	AC, CA	70
5	A → C → D → A	AC, CD, DA	83
6	A → C → D → E → A	AC, CD, DE, EA	91
7	A → D → A	AD, DA	73
8	A → D → E → A	AD, DE, EA	86
9	A → E → A	AE, EA	78
10	C → B → C	CB, BC	56
11	C → B → A → C	CB, BA, AC	87
12	C → B → A → D → C	CB, BA, AD, DC	96
13	C → B → A → D → E → C	CB, BA, AD, DE, EC	110
14	C → D → C	CD, DC	54
15	C → E → C	CE, EC	74
16	C → A → D → C	CA, AD, DC	89

From these pairings, we first define 16 decision variables x_i where $i = 1, 2, \ldots, 16$. Each pairing has its own fixed rate, and our goal is to minimize the total crew cost. Therefore, our objective function is

Minimize $55x_1 + 85x_2 + 101x_3 + 70x_4 + 83x_5 + 91x_6 + 73x_7 + 86x_8 + 78x_9 + 56x_{10} + 87x_{11} + 96x_{12} + 110x_{13} + 54x_{14} + 74x_{15} + 89x_{16}$

This problem has the following constraints:

i. *Each flight must be covered by at least one pairing*. We must set constraints such that there are at least one pairing covering that flight. Flight AB is covered by pairing 1, 2, and 3. Thus, we add our first constraint:

$$x_1 + x_2 + x_3 \geq 1$$

We then repeat the same process for all other unique flights.

ii. *There can only be one of each type of pairing*. To make sure that no pairing is selected more than once, our decision variables must be binary; either a specific pairing is chosen, or it is not. We add our final constraint:

$x_i \in \{0, 1\}$

We now have a complete model for the problem:

Minimize $55x_1 + 85x_2 + 101x_3 + 70x_4 + 83x_5 + 91x_6 + 73x_7 + 86x_8 + 78x_9 + 56x_{10} + 87x_{11} + 96x_{12} + 110x_{13} + 54x_{14} + 74x_{15} + 89x_{16}$

Subject to

$x_1 + x_2 + x_3 \geq 1$	(Flight AB)
$x_1 + x_{11} + x_{12} + x_{13} \geq 1$	(Flight BA)
$x_2 + x_3 + x_{10} \geq 1$	(Flight BC)
$x_2 + x_4 + x_{16} \geq 1$	(Flight CA)
$x_3 + x_5 + x_6 + x_{14} \geq 1$	(Flight CD)
$x_3 + x_6 + x_8 + x_9 \geq 1$	(Flight EA)
$x_3 + x_6 + x_8 + x_{13} \geq 1$	(Flight DE)
$x_4 + x_5 + x_6 + x_{11} \geq 1$	(Flight AC)
$x_5 + x_7 \geq 1$	(Flight DA)
$x_7 + x_8 + x_{12} + x_{13} + x_{16} \geq 1$	(Flight AD)
$x_9 \geq 1$	(Flight AE)
$x_{10} + x_{11} + x_{12} + x_{13} \geq 1$	(Flight CB)
$x_{12} + x_{14} + x_{16} \geq 1$	(Flight DC)
$x_{13} + x_{15} \geq 1$	(Flight EC)
$x_{15} \geq 1$	(Flight CE)
$x_i \in \{0, 1\}$	

The optimal solution for this model is $x = \{0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0\}$ with value 484.
Table 4.1.2 below shows the selected pairings resulting from the model.

Table 4.1.2: Result Pairings

x_i	Pairings	Flights	Cost
2	A → B → C → A	AB, BC, CA	85
5	A → C → D → A	AC, CD, DA	83
9	A → E → A	AE, EA	78
13	C → B → A → D → E → C	CB, BA, AD, DE, EC	110
14	C → D → C	CD, DC	54
15	C → E → C	CE, EC	74
			484

4.2. Solving the Crew Assignment Problem

After solving the Crew Pairing problem, we must solve the second part of crew scheduling, which is the Crew Assignment problem. The crew assignment problem can be defined as follows:

Suppose we have m crew ($c_m = 1, 2, \ldots, m$) to assign to n pairings ($p_n = 1, 2, \ldots, n$). Depending on availability, each crew has a list of pairings they qualify for and rated with preference levels. Our objective is to assign a crew for every pairing, such that no crew is assigned to more than one pairing and each pairing is only given to a single crew. We can formulate a m-by-n matrix to illustrate all crew to pairing assignments.

$$\begin{array}{c c} & \begin{array}{ccc} p_1 & \ldots & p_n \end{array} \\ \begin{array}{c} c_1 \\ : \\ c_m \end{array} & \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \end{array}$$

Like crew pairing, this problem can also be modelled as a set partitioning problem and solved using Binary Integer Programming. However, since this step consists of a one-on-one assignment of crew to pairings, we can also use another solution method called the Hungarian Algorithm, or the Hungarian Method. In this subsection, we will demonstrate how to solve the problem using both solution methods.

Solving the crew assignment problem using Binary Integer Programming is similar to solving the crew pairing problem, with a slight difference. As opposed to the first part of crew scheduling where we assigned a minimum number of pairings to flights, in this phase we must assign every crew to a pairing. Therefore, in order to solve this problem using LIP, we focus on minimizing the number of crew-to-pairing assignments selected to find the best possible assignment. The LIP problem for Crew Assignment can be modeled as follows.

Given the set of possible assignments, we define binary decision variables a_{mn} such that

$$a_{mn} = \begin{cases} 1 \text{ if assignment is chosen} \\ 0 \text{ otherwise} \end{cases}$$

where *m* and *n* correspond to the row and column positions in the assignment matrix.

We formulate this as an optimization problem:

$$\text{Minimize} \sum_{m \in C} \sum_{n \in P} l_{mn} a_{mn} \qquad (1)$$

Subject to:

$$\sum_{n : m \in C} a_{mn} = 1 \quad \forall m \in C \qquad (2)$$

$$\sum_{m : n \in P} a_{mn} = 1 \quad \forall n \in P \qquad (3)$$

$$a_{mn} \in \{0, 1\} \quad \forall m \in C, \forall n \in P \qquad (4)$$

Where:

l_{mn} = Preference level in position (*m*, *n*)
C = The set of all crew
P = The set of all pairings
m = A crew in set *C*
n = A pairing in set *P*

Each valid assignment has a preference level l_{mn} that defines which assignments are most prioritized. The aim of our objective function (1) is to select the best possible assignment such that each crew is assigned to only one pairing (2), and each pairing is assigned to only one crew (3). Our last constraint (4) makes sure only one of each unique assignment is selected.

A Crew Assignment Example

For this example, we use the result pairings from our example in section 4.1. They are renumbered in table 4.2.1 to avoid confusion.

Table 4.2.1: Pairings to Assign

	Pairings
1	A → B → C → A
2	A → C → D → A
3	A → E → A
4	C → B → A → D → E → C
5	C → D → C
6	C → E → C

Suppose we have six crew to assign to these pairings, three from each home base. Since we can only assign crew to pairings that start from and end at their home base, we can solve two separate problems for this.

For home base A, suppose crew 1 is available for pairings 1 and 2, crew 2 is available for pairings 2 and 3, and crew 3 is available for pairings 1 and 3. Crew 1 favors pairing 2, and both crew 2 and 3 favor pairing 3. Let c_m (m = 1, 2, 3) and p_n (n = 1, 2, 3) denote the crew and pairings, respectively. The possible assignments are shown in table 4.2.2.

Table 4.2.2: Possible Assignments for Home Base A with Preference Levels

	p1	**p2**	**p3**
c1	2	1	■
c2	■	2	1
c3	2	■	1

The lower the value, the higher the preference level is for the assignments; since we are minimizing, the smaller values will be prioritized.

We first show how to solve this problem using Binary Integer Programming. Our binary decision variables are as follows.

$$\begin{array}{c|ccc|} & p_1 & p_2 & p_3 \\ \hline c_1 & a_{11} & a_{12} & \\ c_2 & & a_{22} & a_{23} \\ c_3 & a_{31} & & a_{33} \end{array}$$

Since certain assignments such as (c_1, p_3) are not considered, we do not need decision variables for each position in the matrix. We focus on the feasible options and find the best possible assignment. Adding on the preference levels, our objective function is

Minimize $2a_{11} + a_{12} + 2a_{22} + a_{23} + 2a_{31} + a_{33}$

This problem has the following constraints:

i. *Only one assignment must be made for each crew.* No crew can be assigned to more than one pairing. Thus, we add our first constraint:

$$a_{11} + a_{12} = 1$$

We repeat this for the other two crews.

ii. *Only one assignment must be made for each pairing.* No pairing can be assigned to more than one crew. Thus, we add another constraint:

$$a_{11} + a_{31} = 1$$

We repeat this for the other two pairings.

iii. *There can only be one of each unique assignment chosen.* To make sure no assignment is chosen more than once, our decision variables must be binary. We have our final constraint:

$$a_{mn} \in \{0, 1\}$$

We now have a complete BIP model for the problem:

Minimize $2a_{11} + a_{12} + 2a_{22} + a_{23} + 2a_{31} + a_{33}$

Subject to

$$\begin{array}{ll}
a_{11} + a_{12} = 1 & \text{(Crew 1)} \\
a_{22} + a_{23} = 1 & \text{(Crew 2)} \\
a_{31} + a_{33} = 1 & \text{(Crew 3)} \\
a_{11} + a_{31} = 1 & \text{(Pairing 1)} \\
a_{12} + a_{22} = 1 & \text{(Pairing 2)} \\
a_{23} + a_{33} = 1 & \text{(Pairing 3)} \\
a_{mn} \in \{0, 1\} &
\end{array}$$

The optimal solution for this model is {0, 1, 0, 1, 1, 0}. Crew 1 is assigned to pairing 2, crew 3 is assigned to pairing 3, and crew 3 is assigned to pairing 1. Only crew 3 did not get assigned to a preferred pairing. The highlighted values in table 4.2.3 designate the resulting assignments.

Table 4.2.3: Optimal Assignments for Home Base A with Preference Levels

	p_1	p_2	p_3
c_1	2	1	■
c_2	■	2	1
c_3	2	■	1

Next, we can solve the same problem using the Hungarian Algorithm. For this solution method, we do not need variables; instead, we simply use a matrix containing the preference levels, and any impossible assignment holds an infinite value to prevent it from being selected.

However, unlike in the BIP model, the Hungarian Algorithm requires *m*, the set of crews, and *n*, the set of pairings, to be equal. Therefore, in cases where *m* and *n* do not have the same number of elements, we must add a “dummy” row or column to get a square matrix. Those added assignments have infinite values and will not be considered.

Once we have our assignment matrix, we can proceed with the steps of the Hungarian method. The steps to the Hungarian Algorithm are as follows:

Step (i). Subtract row minima. For each row in the matrix, we find the lowest value and subtract it from all values in that row.

Step (ii). Subtract column minima. For each column in the matrix, we find the lowest value and subtract it from all values in that column.

Step (iii). Draw minimum lines. Cover all zeros in the matrix using the least number of lines possible.

Step (iv). Check optimality. If *n* lines cover the matrix, then we have an optimal assignment. If the number of lines is less than *n*, we proceed to step v.

Step (v). Create additional zeros. Find the smallest element not covered by a line. Subtract that number from all uncovered values and add it to any value that is covered by two lines. Then, repeat step iv.

In our example, the number of crews is equal to the number of pairings, so we formulate our 3-by-3 matrix according to each crew’s availability and preference levels:

$$\begin{array}{c|ccc} & p_1 & p_2 & p_3 \\ \hline c_1 & 2 & 1 & \infty \\ c_2 & \infty & 2 & 1 \\ c_3 & 2 & \infty & 1 \end{array}$$

Our objective is to minimize the matrix so that the assignments with higher preference level get prioritized and we obtain a complete assignment. Since we are minimizing, the pairings with lower priority have larger values, and invalid assignments have infinite values to prevent them from getting selected.

We now proceed with the steps of the Hungarian Algorithm.

(i). We find the smallest value of each row and subtract it to all values in that row. For all rows, the value is 1, so we subtract 1 from all elements of the rows.

$$
\begin{array}{c c}
 & \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\
\begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} & \begin{bmatrix} 1 & 0 & \infty \\ \infty & 1 & 0 \\ 1 & \infty & 0 \end{bmatrix}
\end{array}
$$

(ii). Similarly, we find the smallest value of each column and subtract it to all values in that column. The first column's smallest element is 1, so we subtract it from all values in that column. Since columns 2 and 3 have zeros, they are unaffected.

$$
\begin{array}{c c}
 & \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\
\begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} & \begin{bmatrix} 0 & 0 & \infty \\ \infty & 1 & 0 \\ 0 & \infty & 0 \end{bmatrix}
\end{array}
$$

(iii). Next, we cover all 0's in the matrix with a minimum number of lines.

$$
\begin{array}{c c}
 & \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\
\begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} & \begin{bmatrix} \cancel{0} & \cancel{0} & \cancel{\infty} \\ \cancel{\infty} & \cancel{1} & \cancel{0} \\ \cancel{0} & \cancel{\infty} & \cancel{0} \end{bmatrix}
\end{array}
$$

(iv). We now check optimality for the problem. Since exactly 3 lines cover the 0's, we have an optimal assignment.

Now that the steps of the Hungarian Algorithm are complete, we can now determine the solution. We observe that row c_2 only has one zero, therefore (c_2, p_3) is our first assignment. Since p_3 is already assigned, we then get (c3, p1) as row c_3 can only have p_1 as its pair. Our last assignment can only be (c_1, p_2).

$$
\begin{array}{c c}
 & \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\
\begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array} & \begin{bmatrix} 0 & 0 & \infty \\ \infty & 1 & 0 \\ 0 & \infty & 0 \end{bmatrix}
\end{array}
$$

These assignments match the results from our linear programming model.

Next, we must assign the crew from home base C to the last three pairings. We also have 3 crews to assign to these pairings. Suppose crew 4 is available for pairing 1 and 2, crew 5 is available for pairings 2 and 3, and crew 6 is available for all three pairings. Crew 4 favors pairing 2 and crew 5 favors pairing 3. Crew 6 favors pairing 5, 6 and 4 in that order.

Let c$_n$ and p$_n$ (n = 4, 5, 6) denote the crew and pairings, respectively.

Table 4.2.4: Possible Assignments for Home Base C with Preference Levels

	p_4	p_5	p_6
c_4	2	1	■
c_5	■	2	1
c_6	3	1	2

Our matrix is as follows:

$$\begin{array}{c} \\ c_4 \\ c_5 \\ c_6 \end{array} \begin{array}{c} \begin{array}{ccc} p_4 & p_5 & p_6 \end{array} \\ \begin{bmatrix} 2 & 1 & \infty \\ \infty & 2 & 1 \\ 3 & 1 & 2 \end{bmatrix} \end{array}$$

After applying the steps of the Hungarian Algorithm, we get the following assignment.

$$\begin{array}{c} \\ c_4 \\ c_5 \\ c_6 \end{array} \begin{array}{c} \begin{array}{ccc} p_4 & p_5 & p_6 \end{array} \\ \begin{bmatrix} 0 & 0 & \infty \\ \infty & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \end{array}$$

Thus, crew 4 is assigned to pairing 4, crew 5 is assigned to pairing 6, and crew 6 is assigned to pairing 5. All pairing assignments are shown in table 4.2.5.

Table 4.2.5: Assigned Pairings

	Pairings	**Operating Crew**
1	A → B → C → A	c_2
2	A → C → D → A	c_1
3	A → E → A	c_3
4	C → B → A → D → E → C	c_4
5	C → D → C	c_6
6	C → E → C	c_5

5. Solving the Airline Crew Problem with Julia

In the previous section, we examined solution methods to solve the airline crew problem, the main one being Binary Integer Programming. Due to the number of decision variables and constraints involved, BIP problems are not easily solved by hand. Real life airline crew problems (especially for popular airlines) involve many cities, and a high number of pairings are possible;

making this problem difficult to solve. Therefore, it is favorable to use computer programs to evaluate the solution. In this section, we will use the Julia programming language to solve a larger instance of the airline crew scheduling problem.

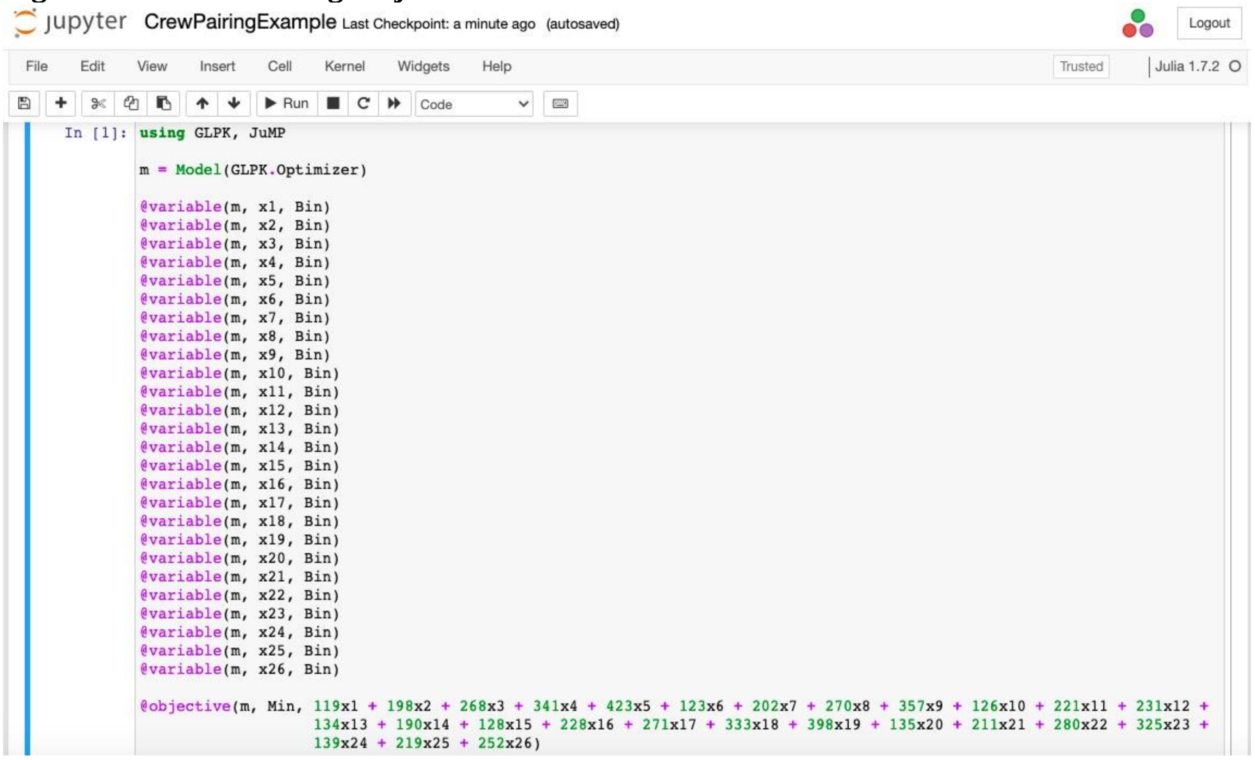
Assume we have the possible pairings in table 5.1 to select from. We wish to minimize the total cost such that all flights are covered.

Table 5.1: Flight Pairings with Respective Cost

	Pairings	**Flights**	**Cost**
1	A → B → A	AB, BA	119
2	A → B → C → A	AB, BC, CA	198
3	A → B → C → D → A	AB, BC, CD, DA	268
4	A → B → C → D → E → A	AB, BC, CD, DE, EA	341
5	A → B → C → D → E → F → A	AB, BC, CD, DE, EF, FA	423
6	A → C → A	AC, CA	123
7	A → C → D → A	AC, CD, DA	202
8	A → C → D → E → A	AC, CD, DE, EA	270
9	A → C → D → E → F → A	AC, CD, DE, EF, FA	357
10	A → D → A	AD, DA	126
11	A → D → E → A	AD, DE, EA	221
12	A → D → E → F → A	AD, DE, EF, FA	231
13	A → E → A	AE, EA	134
14	A → E → F → A	AE, EF, FA	190
15	F → E → F	FE, EF	128
16	F → E → D → F	FE, ED, DF	228
17	F → E → D → C → F	FE, ED, DC, CF	271
18	F → E → D → C → B → F	FE, ED, DC, CB, BF	333
19	F → E → D → C → B → A → F	FE, ED, DC, CB, BA, AF	398
20	F → D → F	FD, DF	135
21	F → D → C → F	FD, DC, CF	211
22	F → D → C → B → F	FD, DC, CB, BF	280
23	F → D → C → B → A → F	FD, DC, CB, BA, AF	325
24	F → C → F	FC, CF	139
25	F → C → B → F	FC, CB, BF	219
26	F → C → B → A → F	FC, CB, BA, AF	252

In this problem, we have 26 decision variables and 23 unique flights to cover. This gives us a very large objective function and many constraints. In order to solve this problem, we use Linear Binary Programming with Julia to optimize the cost.

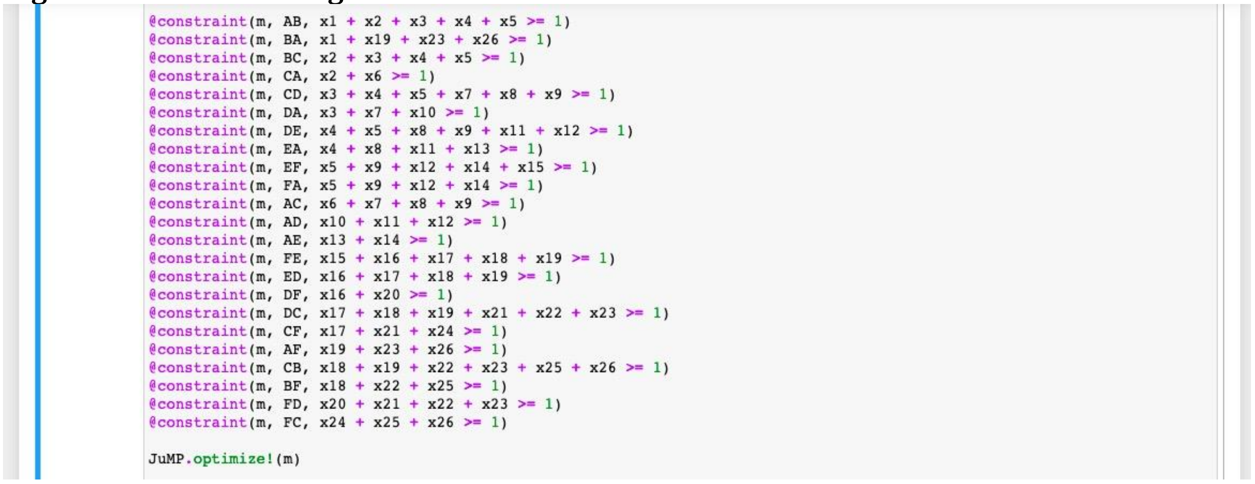
Figure 5.1: Crew Pairing Objective in Julia


jupyter CrewPairingExample Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Julia 1.7.2
Run
Code

```
In [1]: using GLPK, JuMP

        m = Model(GLPK.Optimizer)

        @variable(m, x1, Bin)
        @variable(m, x2, Bin)
        @variable(m, x3, Bin)
        @variable(m, x4, Bin)
        @variable(m, x5, Bin)
        @variable(m, x6, Bin)
        @variable(m, x7, Bin)
        @variable(m, x8, Bin)
        @variable(m, x9, Bin)
        @variable(m, x10, Bin)
        @variable(m, x11, Bin)
        @variable(m, x12, Bin)
        @variable(m, x13, Bin)
        @variable(m, x14, Bin)
        @variable(m, x15, Bin)
        @variable(m, x16, Bin)
        @variable(m, x17, Bin)
        @variable(m, x18, Bin)
        @variable(m, x19, Bin)
        @variable(m, x20, Bin)
        @variable(m, x21, Bin)
        @variable(m, x22, Bin)
        @variable(m, x23, Bin)
        @variable(m, x24, Bin)
        @variable(m, x25, Bin)
        @variable(m, x26, Bin)

        @objective(m, Min, 119x1 + 198x2 + 268x3 + 341x4 + 423x5 + 123x6 + 202x7 + 270x8 + 357x9 + 126x10 + 221x11 + 231x12 +
                           134x13 + 190x14 + 128x15 + 228x16 + 271x17 + 333x18 + 398x19 + 135x20 + 211x21 + 280x22 + 325x23 +
                           139x24 + 219x25 + 252x26)
```

As shown in figure 5.1, we create a new model called *m* and declare our 26 decision variables to denote each pairing. Since the value of a pairing can only be 1 or 0, all decision variables are set to binary. We then add our objective function that includes the cost of each pairing.

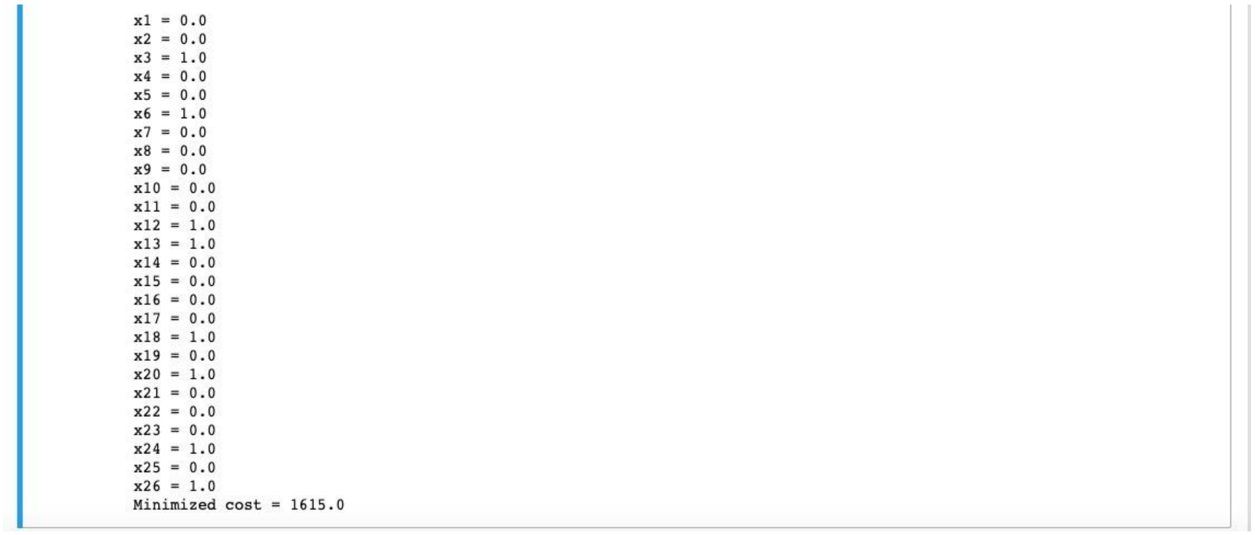
Figure 5.2: Crew Pairing Constraints in Julia



```
@constraint(m, AB, x1 + x2 + x3 + x4 + x5 >= 1)
@constraint(m, BA, x1 + x19 + x23 + x26 >= 1)
@constraint(m, BC, x2 + x3 + x4 + x5 >= 1)
@constraint(m, CA, x2 + x6 >= 1)
@constraint(m, CD, x3 + x4 + x5 + x7 + x8 + x9 >= 1)
@constraint(m, DA, x3 + x7 + x10 >= 1)
@constraint(m, DE, x4 + x5 + x8 + x9 + x11 + x12 >= 1)
@constraint(m, EA, x4 + x8 + x11 + x13 >= 1)
@constraint(m, EF, x5 + x9 + x12 + x14 + x15 >= 1)
@constraint(m, FA, x5 + x9 + x12 + x14 >= 1)
@constraint(m, AC, x6 + x7 + x8 + x9 >= 1)
@constraint(m, AD, x10 + x11 + x12 >= 1)
@constraint(m, AE, x13 + x14 >= 1)
@constraint(m, FE, x15 + x16 + x17 + x18 + x19 >= 1)
@constraint(m, ED, x16 + x17 + x18 + x19 >= 1)
@constraint(m, DF, x16 + x20 >= 1)
@constraint(m, DC, x17 + x18 + x19 + x21 + x22 + x23 >= 1)
@constraint(m, CF, x17 + x21 + x24 >= 1)
@constraint(m, AF, x19 + x23 + x26 >= 1)
@constraint(m, CB, x18 + x19 + x22 + x23 + x25 + x26 >= 1)
@constraint(m, BF, x18 + x22 + x25 >= 1)
@constraint(m, FD, x20 + x21 + x22 + x23 >= 1)
@constraint(m, FC, x24 + x25 + x26 >= 1)

JuMP.optimize!(m)
```

For each unique flight, we add a constraint to make sure at least one of the pairings selected covers that flight. We then use the optimize method to solve the objective under these constraints. Figure 5.2 shows all of the constraints needed for our problem. After executing and printing the results, we get the following solution shown in figure 5.3.

Figure 5.3: Crew Pairing Result



```
x1 = 0.0
x2 = 0.0
x3 = 1.0
x4 = 0.0
x5 = 0.0
x6 = 1.0
x7 = 0.0
x8 = 0.0
x9 = 0.0
x10 = 0.0
x11 = 0.0
x12 = 1.0
x13 = 1.0
x14 = 0.0
x15 = 0.0
x16 = 0.0
x17 = 0.0
x18 = 1.0
x19 = 0.0
x20 = 1.0
x21 = 0.0
x22 = 0.0
x23 = 0.0
x24 = 1.0
x25 = 0.0
x26 = 1.0
Minimized cost = 1615.0
```

The pairings with a value of 1 were the ones selected from our original 26. Table 5.2 shows the selected pairings resulting from the model.

Table 5.2: Result Pairings

x_i	**Pairings**	**Flights**	**Cost**
3	A → B → C → D → A	AB, BC, CD, DA	268
6	A → C → A	AC, CA	123
12	A → D → E → F → A	AD, DE, EF, FA	231
13	A → E → A	AE, EA	134
18	F → E → D → C → B → F	FE, ED, DC, CB, BF	333
20	F → D → F	FD, DF	135
24	F → C → F	FC, CF	139
26	F → C → B → A → F	FC, CB, BA, AF	252
			1615

Those eight pairings cover every flight mentioned in Table 5.1.

Now that we have our minimal cost set of pairings, we assign these pairings to crew. These pairings have two home bases, A and F. Suppose the airline has 8 crew (4 from each home base) to assign to the pairings.

Let c_m (m = 1, 2, …, 8) and p_n (n = 1, 2, …, 8) denote the crew and pairings, respectively. Table 5.3 below illustrates the possible assignments of crew to each pairing, with their preference levels.

Table 5.3: Possible Pairings with Preference Levels

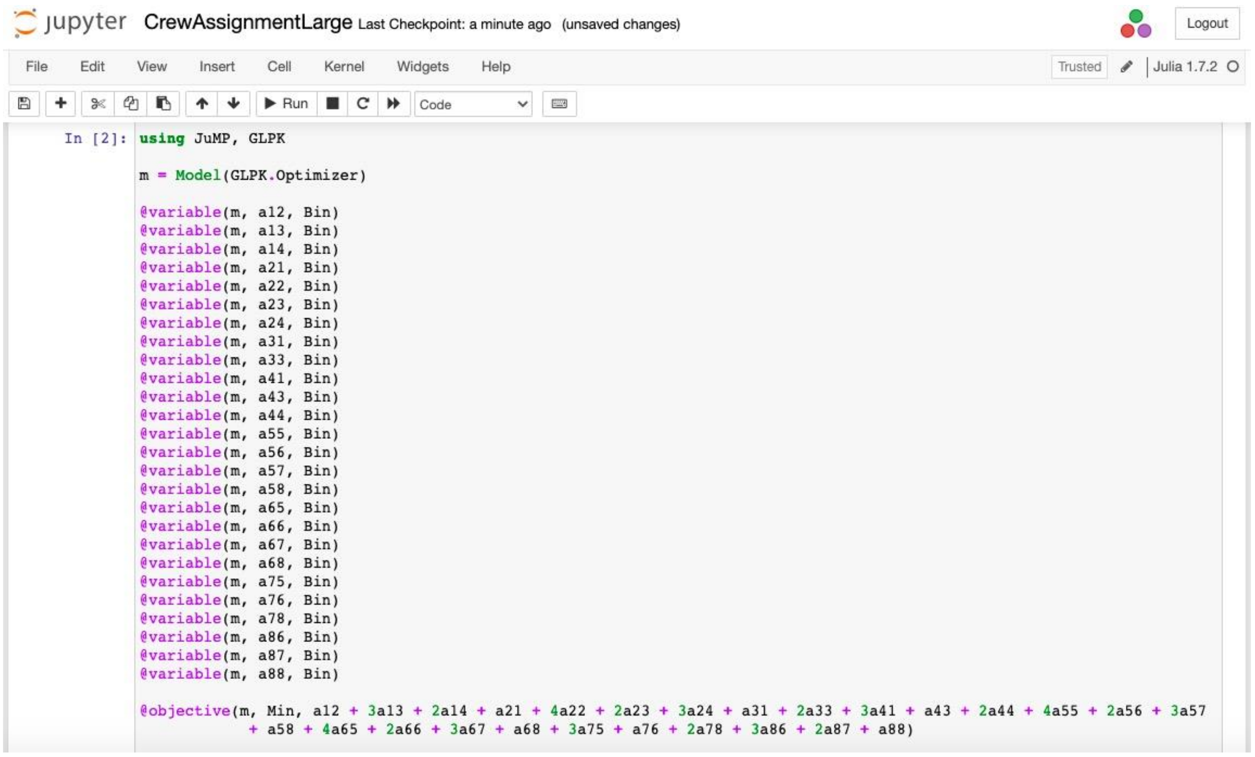
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
c_1	■	1	3	2	■	■	■	■
c_2	1	4	2	3	■	■	■	■
c_3	1	■	2	■	■	■	■	■
c_4	3	■	1	2	■	■	■	■
c_5	■	■	■	■	4	2	3	1
c_6	■	■	■	■	4	2	3	1
c_7	■	■	■	■	3	1	■	2
c_8	■	■	■	■	■	3	2	1

From the table, we build a matrix with the assignment variables.

$$\begin{array}{c|cccccccc|}
 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \\
c_1 & & a_{12} & a_{13} & a_{14} & & & & \\
c_2 & a_{21} & a_{22} & a_{23} & a_{24} & & & & \\
c_3 & a_{31} & & a_{33} & & & & & \\
c_4 & a_{41} & & a_{43} & a_{44} & & & & \\
c_5 & & & & & a_{55} & a_{56} & a_{57} & a_{58} \\
c_6 & & & & & a_{65} & a_{66} & a_{67} & a_{68} \\
c_7 & & & & & a_{75} & a_{76} & & a_{78} \\
c_8 & & & & & & a_{86} & a_{87} & a_{88}
\end{array}$$

Using the preference levels and assignment variables, we can now build our model in Julia.

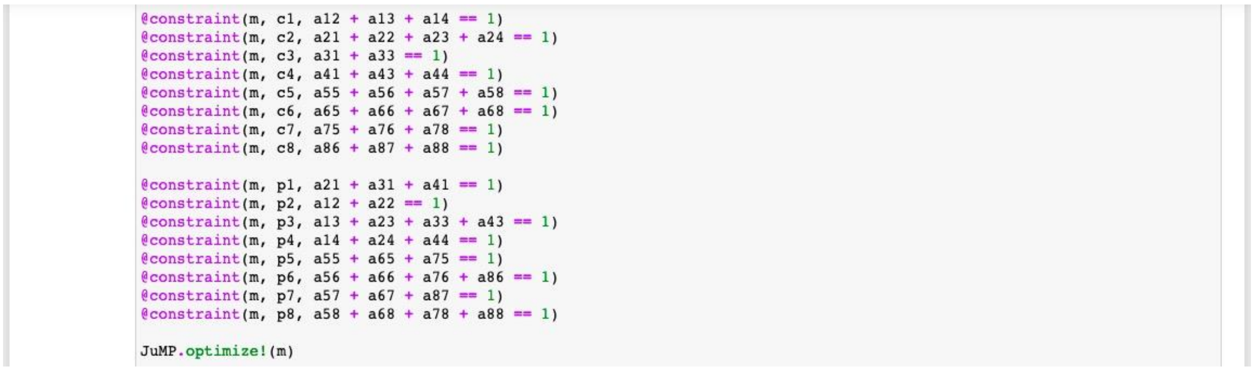
Figure 5.4: Crew Assignment Objective in Julia



```
In [2]: using JuMP, GLPK

        m = Model(GLPK.Optimizer)

        @variable(m, a12, Bin)
        @variable(m, a13, Bin)
        @variable(m, a14, Bin)
        @variable(m, a21, Bin)
        @variable(m, a22, Bin)
        @variable(m, a23, Bin)
        @variable(m, a24, Bin)
        @variable(m, a31, Bin)
        @variable(m, a33, Bin)
        @variable(m, a41, Bin)
        @variable(m, a43, Bin)
        @variable(m, a44, Bin)
        @variable(m, a55, Bin)
        @variable(m, a56, Bin)
        @variable(m, a57, Bin)
        @variable(m, a58, Bin)
        @variable(m, a65, Bin)
        @variable(m, a66, Bin)
        @variable(m, a67, Bin)
        @variable(m, a68, Bin)
        @variable(m, a75, Bin)
        @variable(m, a76, Bin)
        @variable(m, a78, Bin)
        @variable(m, a86, Bin)
        @variable(m, a87, Bin)
        @variable(m, a88, Bin)

        @objective(m, Min, a12 + 3a13 + 2a14 + a21 + 4a22 + 2a23 + 3a24 + a31 + 2a33 + 3a41 + a43 + 2a44 + 4a55 + 2a56 + 3a57
                    + a58 + 4a65 + 2a66 + 3a67 + a68 + 3a75 + a76 + 2a78 + 3a86 + 2a87 + a88)
```

Like in the first phase, we create a model in Julia and declare our assignment variables above as binary. We then add our objective function with the preference values. Figure 5.4 shows the set up in Julia.

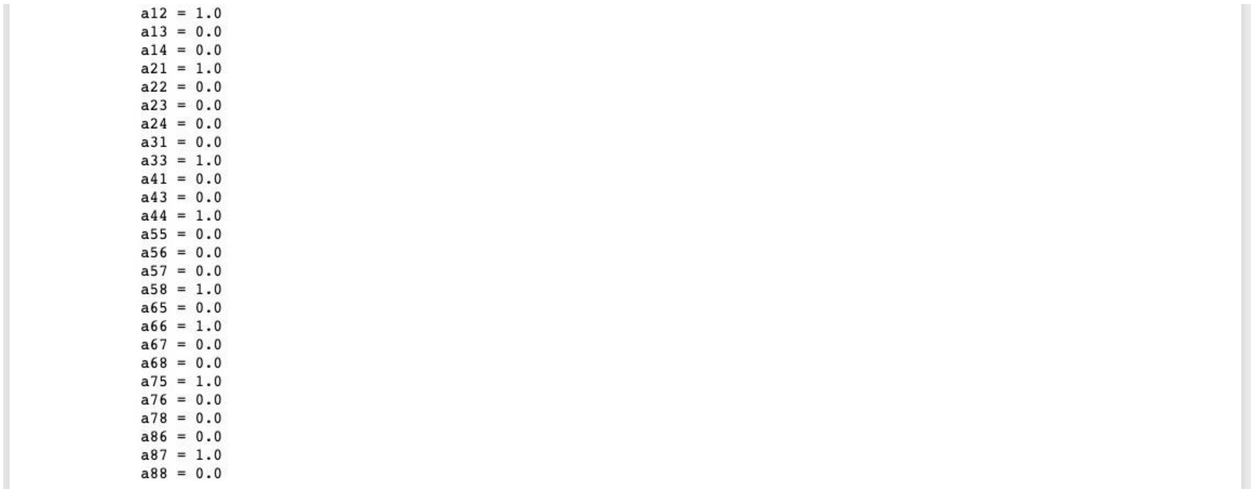
Figure 5.5: Crew Assignment Constraints in Julia



```
@constraint(m, c1, a12 + a13 + a14 == 1)
@constraint(m, c2, a21 + a22 + a23 + a24 == 1)
@constraint(m, c3, a31 + a33 == 1)
@constraint(m, c4, a41 + a43 + a44 == 1)
@constraint(m, c5, a55 + a56 + a57 + a58 == 1)
@constraint(m, c6, a65 + a66 + a67 + a68 == 1)
@constraint(m, c7, a75 + a76 + a78 == 1)
@constraint(m, c8, a86 + a87 + a88 == 1)

@constraint(m, p1, a21 + a31 + a41 == 1)
@constraint(m, p2, a12 + a22 == 1)
@constraint(m, p3, a13 + a23 + a33 + a43 == 1)
@constraint(m, p4, a14 + a24 + a44 == 1)
@constraint(m, p5, a55 + a65 + a75 == 1)
@constraint(m, p6, a56 + a66 + a76 + a86 == 1)
@constraint(m, p7, a57 + a67 + a87 == 1)
@constraint(m, p8, a58 + a68 + a78 + a88 == 1)

JuMP.optimize!(m)
```

Then, we add our constraints. For each crew, only one assignment must be chosen, so we set constraints such that each crew is only given one assignment. We also do the same for each pairing, as each pairing is only covered by one crew. Finally, we use the optimize method to optimize the problem under our constraints. Figure 5.5 shows the constraints we use for the problem.

Figure 5.6: Crew Assignment Result



```
a12 = 1.0
a13 = 0.0
a14 = 0.0
a21 = 1.0
a22 = 0.0
a23 = 0.0
a24 = 0.0
a31 = 0.0
a33 = 1.0
a41 = 0.0
a43 = 0.0
a44 = 1.0
a55 = 0.0
a56 = 0.0
a57 = 0.0
a58 = 1.0
a65 = 0.0
a66 = 1.0
a67 = 0.0
a68 = 0.0
a75 = 1.0
a76 = 0.0
a78 = 0.0
a86 = 0.0
a87 = 1.0
a88 = 0.0
```

After executing and printing the results, we get the resulting values for our assignment variables in figure 5.6. Assignments with a value of 1 were the ones selected by the model. Table 5.4 shows the optimal solution for the assignments.

Table 5.4: Optimal Assignments with Preference Levels

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
c_1	■	1	3	2	■	■	■	■
c_2	1	4	2	3	■	■	■	■
c_3	1	■	2	■	■	■	■	■
c_4	3	■	1	2	■	■	■	■
c_5	■	■	■	■	4	2	3	1
c_6	■	■	■	■	4	2	3	1
c_7	■	■	■	■	3	1	■	2
c_8	■	■	■	■	■	3	2	1

We now have assigned crew for each pairing. The following table 5.5 shows the original selected pairings from the crew pairing phase along with the crew assigned to each pairing.

Table 5.5: Assigned Pairings

	Pairings	Flights	Operating Crew
1	A → B → C → D → A	AB, BC, CD, DA	c_2
2	A → C → A	AC, CA	c_1
3	A → D → E → F → A	AD, DE, EF, FA	c_3
4	A → E → A	AE, EA	c_4
5	F → E → D → C → B → F	FE, ED, DC, CB, BF	c_7
6	F → D → F	FD, DF	c_6
7	F → C → F	FC, CF	c_8
8	F → C → B → A → F	FC, CB, BA, AF	c_5

6. Conclusion

In this paper, we defined the steps to the airline crew scheduling problem and demonstrated how to solve it using Binary Integer Programming. We also introduced the Hungarian Algorithm as a second solution method for the crew assignment phase of the problem and compared the results. Finally, we used the Julia programming language to model and solve a larger example of the airline crew scheduling problem.

References

AhmadBeygi, Shervin, et al. *An Integer Programming Approach to Generating Airline Crew Pairings*. 2008. University of Michigan.

Barnhart, Cynthia, et al. “Airline Crew Scheduling.” *Handbook of Transportation Science*, edited by Randolph W. Hall, Springer, 2003, pp. 517-560.

Kasirzadeh, Atoosa, et al. *Airline crew scheduling: models, algorithms, and data sets*. Springer, 2015.

Patel, Ronakkumar R, et al. *Scheduling of Jobs based on Hungarian Method in Cloud Computing.* International Conference on Inventive Communication and Computational Technologies, 2017. *IEEE Xplore.*

Rauf, K, et al. *An Airline Crew Scheduling for Optimality.* International Journal of Mathematics and Computer Science, 2016.

Zeghal, F.M., and M. Minoux. *Modeling and solving a Crew Assignment Problem in air transportation.* European Journal of Operational Research, 2006. *ScienceDirect.*