RESEARCH ARTICLE

TITLE

Algorithm creation and optimization for the Crew Pairing Problem

AUTHOR

Jesús Manuel Puentes Gutiérrez.

OTHERS

Ángel A. Juan, Alexandre Viejo Galicia, Antonio Rodil Garrido. Respectively, author, director, consultant and tutor from Universitat Oberta de Catalunya.

ABSTRACT

In every airline, cost savings is essential to be able to compete with other airlines and even to survive in such aggressive sector as the world of aviation is. The most decisive factor, after fuel costs, is crew cost. For this reason, we're trying to get, on the one hand, cost savings and, on the other hand, to improve crew job conditions. More specifically, we try to obtain efficient flight pairings with minimal computer processing time, and always achieving all the airline or collective agreement constraints. We also pretend to stablish the basis to follow improving and optimizing crew scheduling, as crew rostering or avoiding delays and cancellations optimizing the entire system.

KEYWORDS

Crew Pairing Problem, optimization algorithm, Crew Scheduling, cost savings, leg, pairing, air routes, flight schedule.

INTRODUCTION

Airlines look for efficiently solving crew scheduling operation (The Crew Scheduling Problem) to decrease costs, specially staff ones as salaries, per diem expenses, etc. and so increase benefits. These costs are the second higher reason to worry about for airlines, after fuel expenses, just as we can see in the next general airline costs graphic:

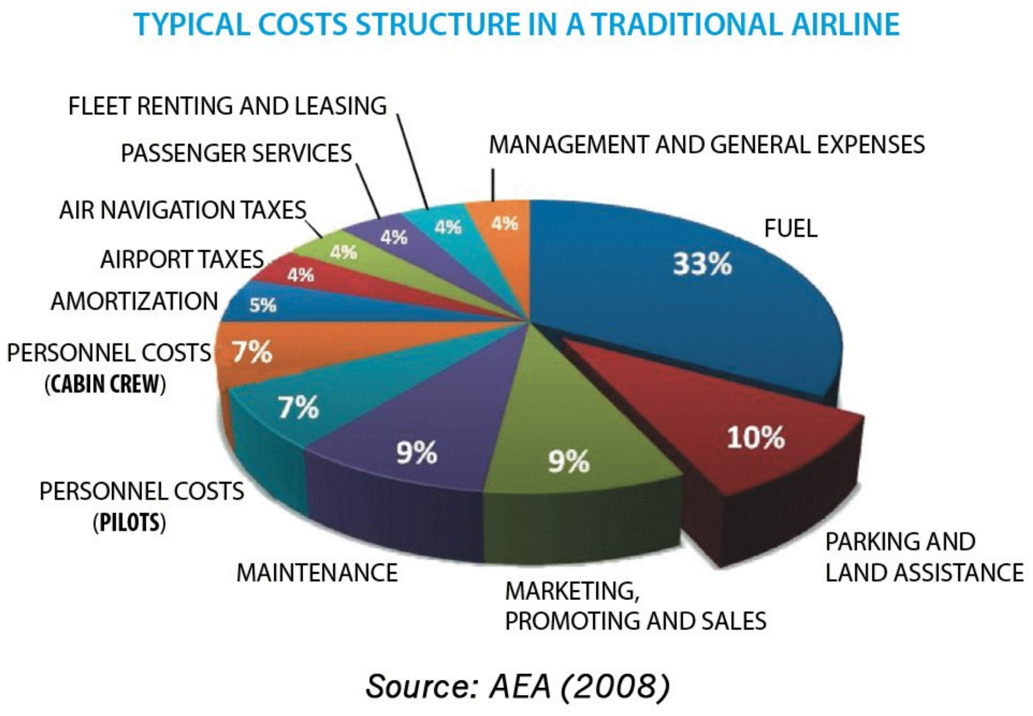


Illustration 1: Main expenses in an airline according to the Spanish government department "Ministerio de Fomento"

As we can observe, personnel expenses (pilots and cabin crew) are 14 % in an airline, which suppose the second higher cost after fuel expense (33 %). This data can be found in the Spanish government department "Ministerio de Fomento":

http://www.fomento.gob.es/NR/rdonlyres/77B84967-D3E5-4EB9-87B8-5CC1E5855641/71842/ANALISISCOMPARATIVO1.pdf

Therefore, saving costs in crew scheduling is also profitable. Even it can be the difference between coming through or not.

Owing to the importance of solving this trouble, in this research we are going to approach the problem subdividing it into: the crew pairing and the crew rostering. We specifically choose the first sub-problem because both complement each other and because solving it can solve the next one.

PROBLEM DESCRIPTION

In the following graphic, we can observe the crew scheduling elaboration process:

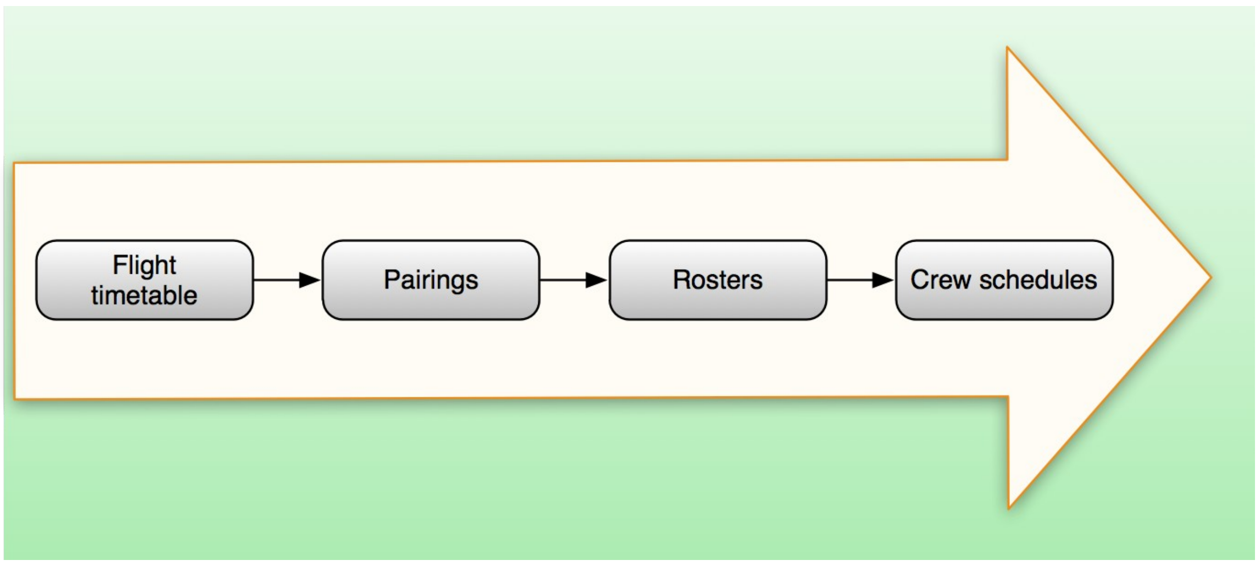


Illustration 2: Detail in the crew scheduling elaboration process

It is one of the most difficult combinatorial optimization issues. It's due to the difficulty it supposes assigning personnel (even with different skills or personal conditions) to different working days (flight pairings), depending on the city they're placed and achieving several constraints (as a rest period prior to a working day, maximum daily or monthly flight hours, maximum number of flights, etc.). That's why all possible combinations provide us a limited number of variables, obtaining a hardly soluble problem. Therefore, it's better dividing it into two phases: The Crew Pairing and The Crew Rostering.

The Crew Pairing Problem is a sequence of flight legs which begins and ends in the same base, whenever it is possible and always satisfying all the imposed regulations and constraints. It can be solved as shown in the following graphic:

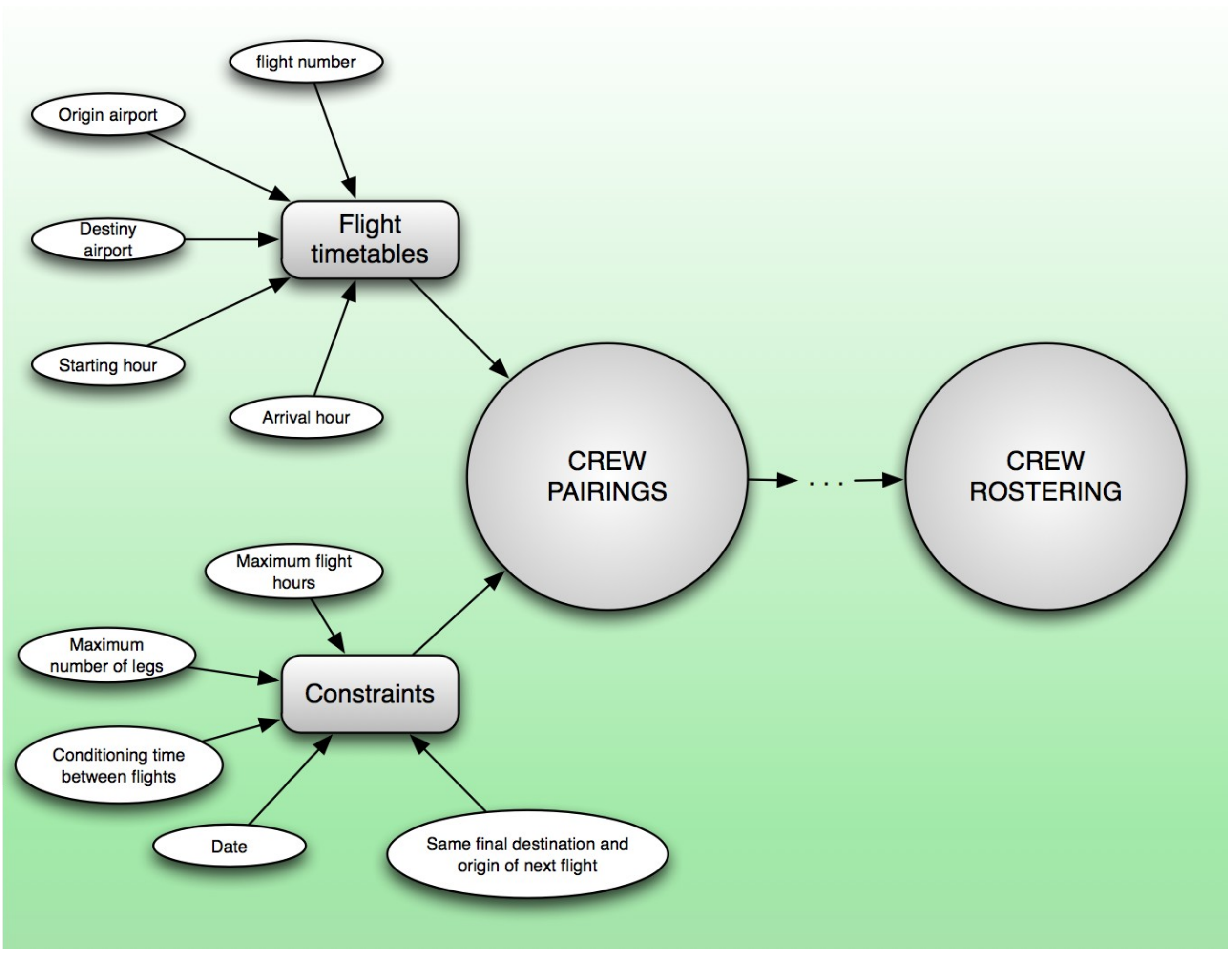
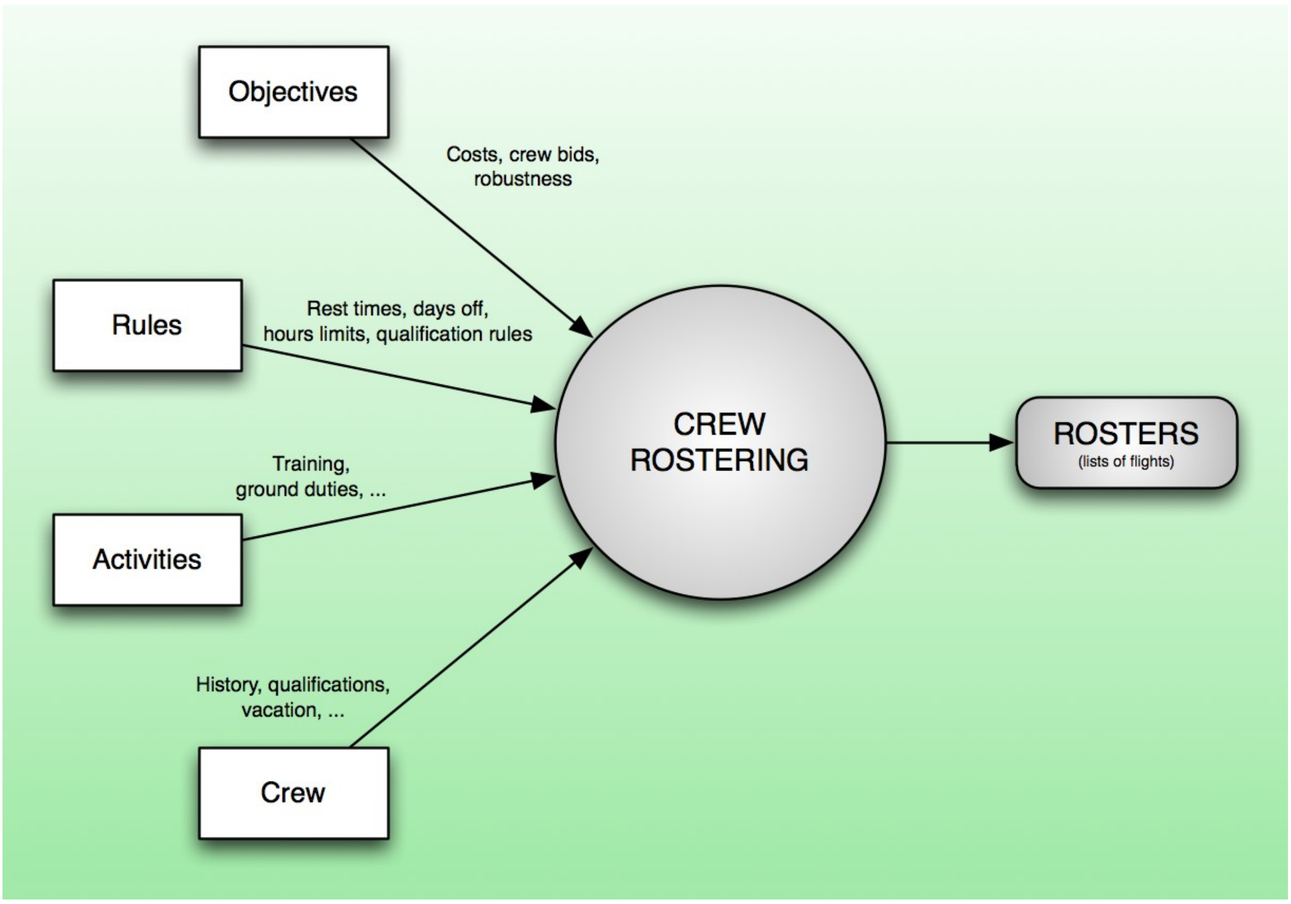


Illustration 3: Crew pairings obtaining process

A crew pairing sequence may span from one to eight days, depending on the airline, the flight type (domestic or transatlantic flights) and the government or contractual regulations. The objective of crew pairing is to find a set of pairings that covers all flights and minimizes the total crew cost. The final crew pairing includes dates and times for each week, but we reduce or remove frequency of flights during that period of time.

After obtaining final crew pairings, they're not addressed to individual crew members because they are part of the next phase: the "Crew Rostering Problem". It represents the process of assigning individual crew members to crew pairings (which have been obtained in the previous phase) on a monthly basis. Therefore, the objective is to assign crew pairings to particular crew members and so to maximize human resources. Graphically we can observe it in the following diagram:


Objectives
Costs, crew bids, robustness
Rules
Rest times, days off, hours limits, qualification rules
Training, ground duties, ...
Activities
History, qualifications, vacation, ...
Crew
CREW ROSTERING
ROSTERS
(lists of flights)
Illustration 4: Crew Rostering obtaining process

The use of crews also depends on airlines, training courses, resting and working periods and so on. Besides, we must take into account several constraints (weekly considered):

- At least one day off between pairings
- Two parings per week (depending on the number of days off that must be assigned in a month)
- About 20 flight hours per week (depending on the total number of flight hours in a month)

All these constraints are approaches and they always depend on union, government and contractual regulations.

LITERATURE REVIEW

Crew Scheduling Problem can also be used in other aspects of daily life which have similar troubles. We're talking about a problem where airlines must fill every flight leg with their corresponding crew members consistently with all needed constraints as Preston (1992) states in his article. There, methods for solving the set partitioning and covering problem are investigated. Besides, a test example illustrates the problem and the use of heuristics. It also suggests a computer software (The Air Crew Scheduler) to develop and modify crew pairings, and create and optimize crew schedules.

This issue has been studied for several decades and, even though it's a problem with a hardly solution, where many techniques have been applied with different results. We also must take into consideration that in the long term, airlines and that problem have been growing hugely. And practical and mathematical procedures are needed to solve the problem effectively. In the same way, Gopalakrishnan and Johnson (2005) do a proper study based on the different techniques used to deal with the matter when the article was being created. They also make a proposal to future investigations about this matter. More specifically, in this study the optimal allocation of crews to flights is treated. They try to obtain minimal crew costs, and flight attendant and cockpit crews are considered in the same way because of their similar structure and treatment. Yet, cockpit crews are a little more focus due to bigger crew costs.

On the other hand, (Graves et al., 1993) state the same thing in their article. In that article, they focus the problem applying different rules depending on each crew category and each aircraft type. That way, they can divide the problem making it more treatable, dividing it into different assignment pairing troubles. In spite of being a daily, weekly or monthly issue, Gopalakrishnan and Johnson (2005) focus it daily because it's the most common trouble solving it as a partition problem depending on costs or constraints. They also use a matrix formula to verify if a flight leg can be made.

The problem we're dealing with had been treated by airlines long time ago. They had to solve and optimize it properly for saving costs and to improve resources. Airline companies were claiming studies, as Arabeyre et al., (1969), where they tried to optimize crew flights adaptation. In that research an approach was made by integer programming with 0-1 variables and with matrix of coefficients. It covered the generations, costing, reduction and optimization of such matrices.

But while airlines and crew increased, the problem grew with 10^4 columns and between 500 – 1000 rows matrices, becoming a hard solution problem. (Rubin 1973) survey thereby appeared, where many other surveys were based in subsequently. In that survey, he used a set covering algorithm repeatedly on much smaller matrices extracted from the overall problem, generating columns as needed and dealing with the main problem as many sub-problems.

Dividing a NP-hard problem into several easier sub-problems is a widely used technique. Moudani et al. (2000) work on it tackling disturbances which leads to modifications for the monthly personalized assignment of the crew staff. In particular, they deal with the crew staff assignment problem (the crew rostering problem) while minimizing changes in the monthly crew scheduling. They propose a new systematic approach based in a repeatedly crew staff assignment problem resolution, dividing the original one into easier sub-ones. In spite of not producing an exact solution in pure mathematical terms, they obtain an almost optimal solution quite adapted to the operational context of airlines.

This kind of NP-hard problem about crew scheduling is also studied by Ozdemir and Mohan (2001), but based in genetic algorithms. In particular, their goal is to obtain solutions but respecting crew needs, minimizing number of crews, maximizing crew time utilization and balancing workloads. They get all these objectives using a genetic algorithm applied to a flight graph representation where several uncovered flights are represented and several constraints are kept in mind.

The problem we're studying is important, but previously we must take into account another one: aircraft availability and routing, same as crew members. Although it's a similar trouble, it must be taken into account because aircrafts are needed to make flights. This is the point given in Mercier et al. (2005) survey, where they focus in determining a minimum cost set of aircraft routes and crew pairings such that each flight leg is covered by one aircraft and one crew, and side constraints are satisfied.

Most airlines use a sequential procedure to plain their operations, after creating a schedule that defines origin and destination cities as well as departure and arrival times for each flight leg. That procedure assigns an aircraft type to each flight leg to maximize anticipated profits so as to cover each leg exactly once for each aircraft individually. After setting aircraft routes and side constraints, airlines build crew scheduling solving the scheduling problem. So Mercier et al. (2005) methodology combines Benders decomposition and column generation. Benders decomposition is explained and developed in this article and it is used to reformulate the problem so as to reduce the number of variables at the expense of an increase in the number of constraints. The Benders master problem solves the crew scheduling problem whereas the aircraft routing issue is dealt with by the Benders sub-problem.

Other authors, instead, as Medard y Sawhney (2007), propose creating and assigning crew working plans in just one step, instead of several ones. These authors provide solution techniques based on simple tree search and more sophisticated column generation and shortest-path algorithms. This way, when making modifications, you must undo part of the job done during crew scheduling creation phase. This situation is due to a flight cancellation or delay, making the original pairing impossible for the crew, but it exists a crew pairing recovery software program which belongs to Yu et al. (2003) and Wei et al. (1997), where broken pairings are repaired and new ones are built to cover unforeseen flights.

(Medard and Sawhney 2007) integrates the pairing and rostering models used in planning, and applies a generate and optimize technique that incorporates both pairing construction and pairing assignment in one step. This direct approach is made possible by a new look at the mathematical models and because limiting the recovery time period reduces the size of the original problem.

In that way, Weide et al. (2010) consider the aircraft routing problem together with both crew scheduling problem, one for technical crew and another one for cabin crew. Instead of solving the integrated model, they propose to solve both original problems iteratively. They start with a minimal cost crew pairing solution without taking aircraft routings into account. Then, in each iteration, they solve the individual aircraft routing problem first, taking into account the current crew pairing solution. Then, once they have finished with iterations, both troubles are solved.

Normally, when you solve crew scheduling problems, you use exact methodologies like linear programming, dynamic programming or branch and bound algorithms, as defined in (Ángel A. Juan et al. 2010) survey. But even though decent levels of performance for these methods are reached, in some cases, with greater than 100 elements, the success rate is variable. For this reason, starting with one of these heuristics algorithms, as CWS (Clark and Wright Savings heuristics algorithm) and applying it to our problem (Crew Pairing Problem), we can apply a randomization process with statistical distributions to improve classic algorithms efficiency and effectiveness.

Also in that way, if we use the CWS algorithm combined with the use of Monte Carlo simulation (MCS), as done in (Ángel A. Juan et al. 2011) survey, we can improve the results with numerical solutions, specially for complex problems that can't be efficiently solved using analytic approximations. The algorithm used in that survey can be defined as a group of techniques that use random numbers and statistical distributions to solve several deterministic or stochastic problems. Thanks to that method a random process is introduced to improve feasible solutions space search process, covering all possibilities just once. Instead of choosing the optimal solution (the one used in CWS algorithm), they choose a quasi-optimal one with greater probability, in a higher probability list with better savings. This selection is done without introducing too many parameters to avoid algorithm complication. They use different geometric statistical distributions during the randomized CWS solution-construction process to obtain objectives set.

Other method used to solve airline crew scheduling problem is Deng and Lin (2011) one. It's based in Ant Colony Optimization algorithm. They attempt to search shortest path from the flight graph such as TSP (travelling salesman problem) does. Results indicate that "ant colony optimization" is an effective and efficient heuristic algorithm to minimize the total crew costs by effectively scheduling flights to reduce overnight stay in hotels, deadhead times and sitting times.

After solving the dealt problem in this survey, the crew pairing problem (which is the first phase in the crew scheduling problem) can be used as starting-point to solve the crew rostering one. This second phase is also studied in other surveys as (Ernst et al. 2004), where they review methods, models and software or in (Maenhout and Vanhoucke 2010) survey, where they use a scatter search algorithm for the airline crew rostering problem assigning a personalized roster to each crew member, minimizing the overall operational costs.

Finally, this survey can also be used to strengthen generated schedules. For instance, a time window can be added to build the schedule up, avoiding delays propagation as Dunbar et al. (2010) suggest in their survey to minimize the overall propagated delay and produce a robust solution for both aircraft and crew. Also AhmadBeygi et al. (2008) use a propagation tree to minimize delay propagation due to flights and crew in a particular point in the schedule, by redistributing existing slack in the planning process. Also Chiraphadhanakul and Barnhart (2011) look for re-allocate existing slacks including delay propagation and passengers delays in an easier way, limiting complexity.

SOLUTION METHODOLOGY

For solving the crew pairing problem, we create an algorithm that calculates pairings for crew members, beginning from existing flights. Besides, these flights must be covered linking each other and being part of pairings according to necessary constraints.

Once we get the algorithm, we compare actual ones with ours and compare processing times to upgrade those times obtained, if possible. We'll also make an evaluation with the obtained results.

The algorithm we've used to get pairings for flights that achieve constraints is:

```
procedure generateRoutesOfLegs ()
    legs = getInputs();
    sortListOfLegs(legs);
    while (sol is not null) →
        clearRoutes(route);
        if we need a leg (anotherLeg is true) →
            leg = selectLeg(legs);
            route = addLeg(leg);
            removeFromList(leg);
        end if
        while (legs is not empty) →
            sol = searchSolution(legs);
            if sol is not null →
                addRoute(sol, route);
                removeFromList(sol);
            else
                saveRouteSolution(route);
                showRouteSolution(route);
                anotherLeg = true;
            end if
        end while  // legs is not empty
    end while  // sol is not null
end
```

Fig. 1: Article's main algorithm

As we can see in the second row, we read algorithm's entry data, which is the starting legs we are going to create. This generated data is read from a text file (test.txt). In particular, it corresponds to:

- Pairing or flight number (unique number)
- Starting airport for a specific leg
- Ending airport for a specific leg
- Starting time for the aforesaid leg
- Landing time for the mentioned leg
- Leg's date

After reading data, we shape an object through a Java class which contains our flight leg according to the form BCN – MAD (for example) with taking off and landing times (10:00 – 11:00) corresponding to “X” flight in the “Y” date. In particular, this created and shaped class would be the “Leg” class which is into “Input” class as we can see in the Java program appendix:

```
    // Read inputs files and construct the inputs object
    Inputs inputs = InputsManager.getInputs(testsFilePath);
```

Fig. 2: Java code to read entry code and model it in a class

Once the data is available, we sort it (row number 3 in the algorithm) according to taking off time from each leg, making more sense to pairings.

After that, thanks to a loop where we select (row 6) and use all the available legs (row 4), we will apply all constraints to obtain our final solutions. In each loop iteration, we'll select a leg (the first one in the remaining unused and sorted legs) which is going to be the beginning of our partial solution pairing. This choice, done in row 7, is added to the partial solution (row 8) and deleted from the available legs list, because it has already been used.

Later, thanks to another new loop (row 11), we begin to look for a leg from the remaining legs list that we have to link with other ones where all constraints are obeyed (row 12). In particular, these constraints are the following:

- The first and obvious one is that final destination airport in a leg must be initial airport in the next linking leg. And that is why the aircraft and the crew must be in the airport where next leg begins to be able to continue with the assigned pairing.
- Minimum time to be able to prepare aircraft, embark and disembark passengers, get crew ready themselves, prepare documentation, etc., will be one hour between legs.
- Pairings will be prepared for the same date. Even though they are daily pairings, they can be later joined to get several days pairings.
- Maximum number of legs in a pairing will be 5.
- Maximum number of flight hours in a day made by crew is 8.

The main reason to choose these constraints is that pairings creation is based in author's personal experience. The author in this research has been working in an airline for 22 years as a crew member. Besides, these parameters are the most common ones used while developing flight schedules. In particular, airlines use one hour as stopover between each air route as it is the most common procedure. Also we use it for being an easier way of calculating data, although, domestic flights stopover time is 45 minutes. Also 5 legs in a day is the usual maximum value for these kind of flights.

First, those constraints will be firstly used, then we can add others or change the existing ones, though increasing processing time and complexity. To make this labour easier, in Java code we add several constants at the beginning of the main class “Inicio”, where anyone can modify the values, as followed:

```
    final static int TIEMPO_ESCALA_MIN = 60;     // Stopover minimum time
                                                          in minutes
    final static int NUM_MAX_SALTOS = 5;
    final static int HORAS_MAX_VUELO = 800;      // Equivalent to 8
                                                    flight hours as
                                                    maximum time
```

Fig. 3: Constants to modify constraints easily

The constraints check is done in row 12 in the main algorithm through a procedure call with the following algorithm:

```
1    procedure searchSolution (legs)
2         solution = null;
3         while list contain legs →
4              if leg meets all constraints →
5                   solution = actualLeg();
6              end if
7         end while
8         return solution;
9    end
```

Fig. 4: Algorithm to apply needed constraints in each solution.

In this procedure we cover all available remaining legs (row 3) given as input data (row 1: legs). We check that the dealt leg achieves all given constraints (row 4) and if we find a solution, we give it as output data in the procedure, adding it to the pairing we're building as a final solution part. If we can't find any solution, the "null" value (row 2) is given as output data indicating there's no solution.

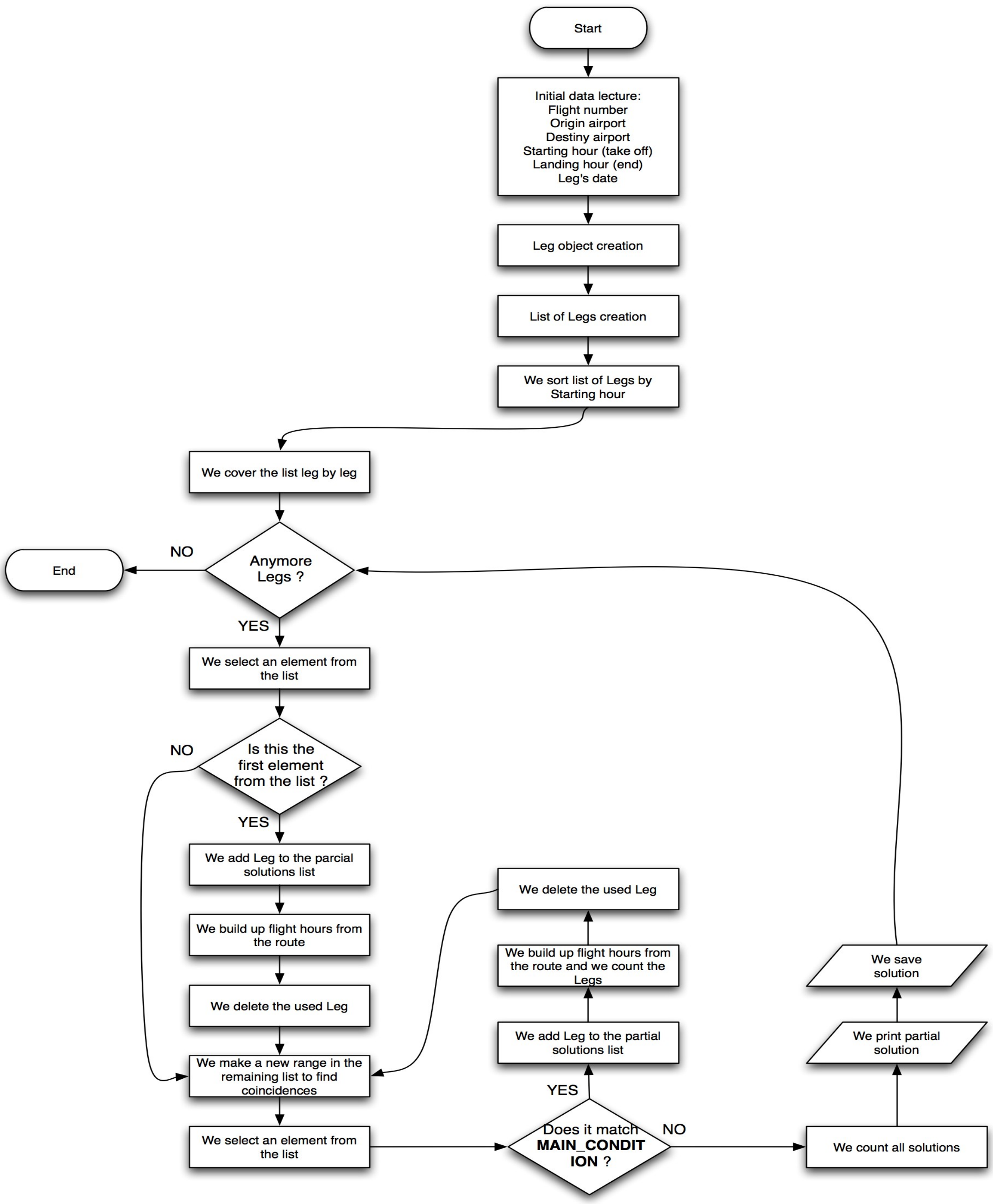
Java code that corresponds to this procedure where a valid leg is searched, is the following:

```java
/**
 * Function that looks for a feasible solution achieving with
 * constraints
 *
 * @param legs
 * @param destino
 * @param horaInicio
 * @param horaFin
 * @param fecha
 * @param numSaltos
 * @param numHorasVuelo
 * @return Returns a Leg that achieves with constraints
 */
public static Leg searchSol(ArrayList<Leg> legs, String destino,
int horaInicio, int horaFin, String fecha, int numSaltos, int
numHorasVuelo){
      Leg sol = null;
      boolean found = false;
      int horas = 0;
      int suma = 0;

      for (Leg i: legs)
            if ((i.getFecha().equals(fecha)) &&
                  (i.getOrigen().equals(destino)) &&
                  ((calcDifMin(horaFin, i.getHoraInicio())) >=
                  TIEMPO_ESCALA_MIN) && (i.getHoraInicio() >
                  horaInicio) && (numSaltos < NUM_MAX_SALTOS) &&
                  (numHorasVuelo <= HORAS_MAX_VUELO)) {
                  horas = calcFlightHours(i.getHoraInicio(),
                            i.getHoraFin());
                  suma = suma + horas;
                  suma = addFlightHours(suma, numHorasVuelo);
                  if ((found == false) && (suma <=
                            HORAS_MAX_VUELO)) {
                       sol = i;
                       found = true;
                  }
             }

      return sol;
}
```

Fig. 5: Algorithm's Java code which looks for solutions applying constraints

Following with the main algorithm (procedure generateRoutesOfLegs()), if we find a leg achieving all constraints (row 13), we add it as a partial solution pairing and we delete this used leg (row 15). Therefore this leg won't be available any more. If we don't find a valid leg (row 16), we can't continue creating or extending our pairing and we finish our dealt pairing which is one of the final solutions. So we save this pairing as part of the solution (row 17) and we continue with remaining legs (row 19).

Once we've finished loop iterations and we've used available legs, we'll dispose all possible pairings achieving with constraints in a file and in our computer screen. So we've got a solution to the crew pairing problem more or less efficient. In the main, the working model diagram of our algorithm is showed with a flowchart in the next page:


Start
Initial data lecture:
Flight number
Origin airport
Destiny airport
Starting hour (take off)
Landing hour (end)
Leg's date
Leg object creation
List of Legs creation
We sort list of Legs by Starting hour
We cover the list leg by leg
Anymore Legs ?
NO
End
YES
We select an element from the list
Is this the first element from the list ?
NO
YES
We add Leg to the parcial solutions list
We build up flight hours from the route
We delete the used Leg
We make a new range in the remaining list to find coincidences
We select an element from the list
Does it match MAIN_CONDITION ?
YES
NO
We add Leg to the partial solutions list
We build up flight hours from the route and we count the Legs
We delete the used Leg
We count all solutions
We print partial solution
We save solution

MAIN_CONDITION = ((Fecha$_{destino\_Leg\_seleccionado}$ = Fecha$_{origen\_Leg\_lista}$) &
(Aeropuerto$_{destino\_Leg\_seleccionado}$ = Aeropuerto$_{origen\_Leg\_lista}$) &
(TiempoEscala >= TiempoEscalaMínima) &
(HoraInicio$_{destino\_Leg\_seleccionado}$ <> HoraInicio$_{origen\_Leg\_lista}$) &
(NúmeroSaltos < NúmeroMáximoSaltos) &
(NúmeroHorasVuelo <= NúmeroMáximoHorasVuelo))

Ilustration 5: Algorithm's flowchart

Java code section shows main algorithm operation as seen in the flowchart above:

```
      do{
            // We clear route to start a new stage
            ruta.clear();
            // We reset the counter
            numSaltos = 1;


            if (primero == true){
                  // We select needed data from the first
                  element in the list
                  destino = legs.get(0).getDestino();
                  horaInicio = legs.get(0).getHoraInicio();
                  horaFin = legs.get(0).getHoraFin();
                  fecha = legs.get(0).getFecha();
                  // We add the first element of this stage (if it is
                  the first used one)
                  ruta.add(legs.get(0));
                  // We calculate flight hours of this stage
                  numHorasVuelo = calcFlightHours (horaInicio, horaFin);
                  // We add total flight hours we've got
                  totalHorasVuelo = addFlightHours(totalHorasVuelo,
                        numHorasVuelo);
                  // We delete the used element (if it is the first
                  used one)
                  legs.remove(0);
                  primero = false;
            }


            do {
                  // We search a Leg solution
                        l = searchSol(legs, destino, horaInicio, horaFin,
                              fecha, numSaltos, totalHorasVuelo);

                  if (l != null){
                        // We add the Leg to the route
                        ruta.add(l);
                        // We count it
                        numSaltos++;

                        destino = l.getDestino();
                        horaInicio = l.getHoraInicio();
                        horaFin = l.getHoraFin();
                        fecha = l.getFecha();


                        // We calculate stage flight hours
                        numHorasVuelo = calcFlightHours (horaInicio,
                              horaFin);
                        // We add total flight hours we've got
```

```
                    totalHorasVuelo = addFlightHours(totalHorasVuelo,
                         numHorasVuelo);

                    // We delete used Leg
                    int indice = legs.indexOf(l);
                    legs.remove(indice);
               } else {
                    // We show solution in the screen
                    numSols++;
                    solution = showSol(ruta, numSols, totalHorasVuelo);

                    // We save solution in a text file (out.txt)
                    saveSolution(solution, outputFilePath);

                    primero=true;
                    totalHorasVuelo = 0;

                    }

          } while (l != null);        // until there's no coincidences
                                      of Legs
        } while (legs.size() > 0);    // until Leg's list is empty
```
Fig. 6: Main algorithm's Java code that shows crew pairings creation.

In this Java code we can see both loops working and building legs solutions (with “do...while” loops), making a call to “searchSol” function which checks that all constraints are achieved.

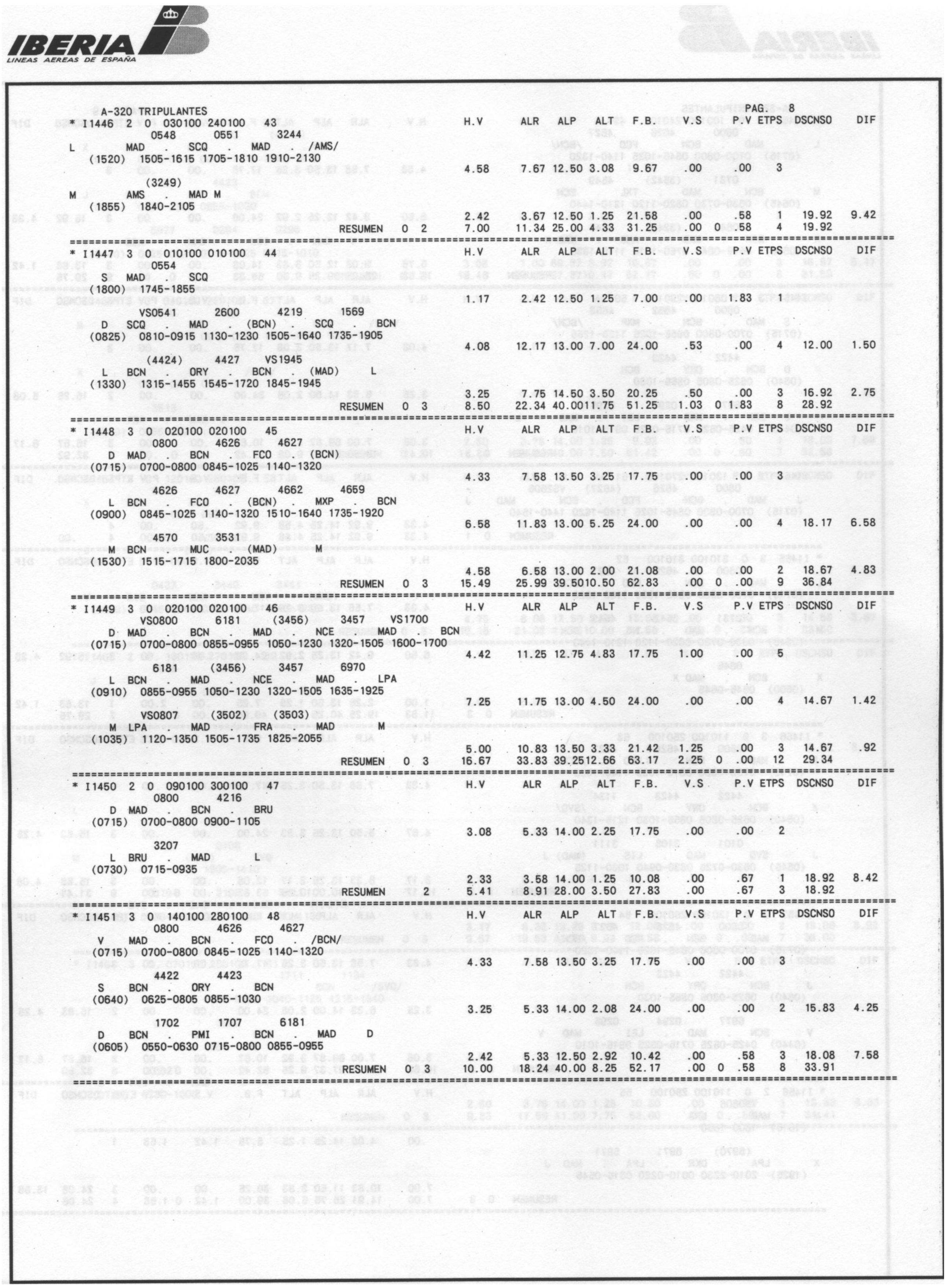
We get algorithm's processing time, using a Java class with its own libraries in this language. First, we calculate system time in the application's starting. Then we have to measure it again when finishing the application. And finally, we're able to obtain the difference of time between both of them. The result gives us application's processing time in hours, minutes and seconds.

Originally, the algorithm and its constraints will be applied to domestic flights and daily pairings. This concept is the basis to solve crew pairing problem. Transoceanic flights or longer flights have other constraints as higher flight hours (they will be greater than 8 hours) and have lower number of legs. That's why the algorithm must be different though the basis is the same. So we should use another one in a new research or adapt it with new parameters.

COMPUTATIONAL EXPERIMENTS

Once we've developed our algorithm as a Java application, we're going to test it with a set of experiments, checking its efficiency.

If it works correctly, we put it to several tests with 50, 100, 150 and 500 daily flights, then we choose a group of real flights as a small token. Flights which belong to Iberia L. A. E. airline are according to our algorithm and represented in the next page to compare results obtained. Iberia's solutions are:



```
      A-320 TRIPULANTES                                                                        PAG.   8
   * I1446  2  0  030100 240100   43                      H.V     ALR   ALP   ALT   F.B.   V.S   P.V ETPS  DSCNSO    DIF
              0548        0551        3244
     L         MAD    .    SCQ    .    MAD    .  /AMS/
       (1520)  1505-1615 1705-1810 1910-2130
                                                          4.58    7.67 12.50 3.08   9.67   .00   .00   3
              (3249)
     M         AMS    .    MAD M
       (1855)  1840-2105
                                                          2.42    3.67 12.50 1.25  21.58   .00   .58   1   19.92    9.42
                                          RESUMEN   0  2  7.00   11.34 25.00 4.33  31.25   .00 0 .58   4   19.92
==========================================================================================================================
   * I1447  3  0  010100 010100   44                      H.V     ALR   ALP   ALT   F.B.   V.S   P.V ETPS  DSCNSO    DIF
              0554
     S   MAD      SCQ
       (1800)  1745-1855                                  1.17    2.42 12.50 1.25   7.00   .00  1.83   1
              VS0541        2600        4219        1569
       D   SCQ    .    MAD    .   (BCN)   .    SCQ    .   BCN
       (0825)  0810-0915 1130-1230 1505-1640 1735-1905
                                                          4.08   12.17 13.00 7.00  24.00   .53   .00   4   12.00    1.50
              (4424)       4427     VS1945
       L   BCN    .    ORY    .    BCN    .   (MAD)     L
       (1330)  1315-1455 1545-1720 1845-1945
                                                          3.25    7.75 14.50 3.50  20.25   .50   .00   3   16.92    2.75
                                          RESUMEN   0  3  8.50   22.34 40.0011.75  51.251.03 0 1.83   8   28.92
==========================================================================================================================
   * I1448  3  0  020100 020100   45                      H.V     ALR   ALP   ALT   F.B.   V.S   P.V ETPS  DSCNSO    DIF
              0800       4626        4627
       D  MAD    .   BCN    .  FCO   .  (BCN)
       (0715)  0700-0800 0845-1025 1140-1320
                                                          4.33    7.58 13.50 3.25  17.75   .00   .00   3
              4626       4627        4662        4659
       L  BCN    .   FCO    .  (BCN)   .   MXP    .   BCN
       (0900)  0845-1025 1140-1320 1510-1640 1735-1920
                                                          6.58   11.83 13.00 5.25  24.00   .00   .00   4   18.17    6.58
              4570       3531
       M  BCN    .   MUC    .  (MAD)       M
       (1530)  1515-1715 1800-2035
                                                          4.58    6.58 13.00 2.00  21.08   .00   .00   2   18.67    4.83
                                          RESUMEN   0  3 15.49   25.99 39.5010.50  62.83   .00 0 .00   9   36.84
==========================================================================================================================
   * I1449  3  0  020100 020100   46                      H.V     ALR   ALP   ALT   F.B.   V.S   P.V ETPS  DSCNSO    DIF
              VS0800        6181      (3456)        3457      VS1700
       D  MAD    .    BCN    .    MAD    .    NCE    .    MAD    .   BCN
       (0715)  0700-0800 0855-0955 1050-1230 1320-1505 1600-1700
                                                          4.42   11.25 12.75 4.83  17.75  1.00   .00   5
              6181       (3456)       3457        6970
       L  BCN    .    MAD    .    NCE    .    MAD    .    LPA
       (0910)  0855-0955 1050-1230 1320-1505 1635-1925
                                                          7.25   11.75 13.00 4.50  24.00   .00   .00   4   14.67    1.42
              VS0807   (3502)    (3503)
       M  LPA    .    MAD    .    FRA    .    MAD      M
       (1035)  1120-1350 1505-1735 1825-2055
                                                          5.00   10.83 13.50 3.33  21.42  1.25   .00   3   14.67     .92
                                          RESUMEN   0  3 16.67   33.83 39.2512.66  63.17  2.25 0 .00  12   29.34
==========================================================================================================================
   * I1450  2  0  090100 300100   47                      H.V     ALR   ALP   ALT   F.B.   V.S   P.V ETPS  DSCNSO    DIF
              0800        4216
       D  MAD    .    BCN    .    BRU
       (0715)  0700-0800 0900-1105
                                                          3.08    5.33 14.00 2.25  17.75   .00   .00   2
              3207
       L  BRU    .    MAD        L
       (0730)  0715-0935
                                                          2.33    3.58 14.00 1.25  10.08   .00   .67   1   18.92    8.42
                                          RESUMEN     2   5.41    8.91 28.00 3.50  27.83   .00   .67   3   18.92
==========================================================================================================================
   * I1451  3  0  140100 280100   48                      H.V     ALR   ALP   ALT   F.B.   V.S   P.V ETPS  DSCNSO    DIF
              0800        4626        4627
     V    MAD    .    BCN    .    FCO    .   /BCN/
       (0715)  0700-0800 0845-1025 1140-1320
                                                          4.33    7.58 13.50 3.25  17.75   .00   .00   3
              4422        4423
     S    BCN    .    ORY    .    BCN
       (0640)  0625-0805 0855-1030
                                                          3.25    5.33 14.00 2.08  24.00   .00   .00   2   15.83    4.25
              1702        1707        6181
     D    BCN    .    PMI    .    BCN    .    MAD      D
       (0605)  0550-0630 0715-0800 0855-0955
                                                          2.42    5.33 12.50 2.92  10.42   .00   .58   3   18.08    7.58
                                          RESUMEN   0  3 10.00   18.24 40.00 8.25  52.17   .00 0 .58   8   33.91
==========================================================================================================================
```

Ilustration 6: Real Pairings table from a real airline

For the two beginning pairings, input data we're going to use as an example are:

```
0548  MAD   SCQ   1505  1615  03/09/2012
0551  SCQ   MAD   1705  1810  03/09/2012
3244  MAD   AMS   1910  2130  03/09/2012
3249  AMS   MAD   1840  2105  04/09/2012
…
```

Fig. 7: Entry data example used in the comparative with Iberia

We continue adding data until completing all testing Iberia flights. We must take into account that Iberia airline uses a 45 minutes stopover time as minimum one in domestic flights. We have to use it as our first constant TIEMPO_ESCALA_MIN. This airline also takes a maximum of 5 air routes in a working day (constant NUM_MAX_SALTOS). Finally we'll suppose that maximum number of flight hours in a working day will be 8 (constant HORAS_MAX_VUELO).

Now, we're going to make several tests just as showed in Iberia pairings. In other words, one date for each pairing as shown. Results obtained with our algorithm are:

```
****************************************************************************
**********************   S O L U C I Ó N   1  ***************************
****************************************************************************

 fecha =                            18/09/2012
 horas de vuelo realizadas =        2 horas y 25 minutos (2.41)

 numLineas =        1702       1707       6181
              BCN ------ PMI ------ BCN ------ MAD
                0550-0630  0715-0800  0855-0955

****************************************************************************
**********************   S O L U C I Ó N   2  ***************************
****************************************************************************

 fecha =                            17/09/2012
 horas de vuelo realizadas =        3 horas y 15 minutos (3.25)

 numLineas =        4422       4423
              BCN ------ ORY ------ BCN
                0625-0805  0855-1030

****************************************************************************
**********************   S O L U C I Ó N   3  ***************************
****************************************************************************

 fecha =                            08/09/2012
 horas de vuelo realizadas =        4 horas y 20 minutos (4.33)

 numLineas =        800       4626       4627
              MAD ------ BCN ------ FCO ------ BCN
                0700-0800  0845-1025  1140-1320
```

```
**************************************************************************
**********************   S O L U C I Ó N   4   ***************************
**************************************************************************

 fecha =                             11/09/2012
 horas de vuelo realizadas =         6 horas y 25 minutos (6.41)

 numLineas =        800       6181       3456       3457       1700
              MAD ------ BCN ------ MAD ------ NCE ------ MAD ------ BCN
                0700-0800  0855-0955  1050-1230  1320-1505  1600-1700

**************************************************************************
**********************   S O L U C I Ó N   5   ***************************
**************************************************************************

 fecha =                             14/09/2012
 horas de vuelo realizadas =         3 horas y 5 minutos (3.08)

 numLineas =        800       4216
              MAD ------ BCN ------ BRU
                0700-0800  0900-1105

**************************************************************************
**********************   S O L U C I Ó N   6   ***************************
**************************************************************************

 fecha =                             16/09/2012
 horas de vuelo realizadas =         4 horas y 20 minutos (4.33)

 numLineas =        800       4626       4627
              MAD ------ BCN ------ FCO ------ BCN
                0700-0800  0845-1025  1140-1320

**************************************************************************
**********************   S O L U C I Ó N   7   ***************************
**************************************************************************

 fecha =                             15/09/2012
 horas de vuelo realizadas =         2 horas y 20 minutos (2.33)

 numLineas =        3207
              BRU ------ MAD
                0715-0935

**************************************************************************
**********************   S O L U C I Ó N   8   ***************************
**************************************************************************

 fecha =                             06/09/2012
 horas de vuelo realizadas =         5 horas y 10 minutos (5.16)

 numLineas =        541       2600       4219       1569
              SCQ ------ MAD ------ BCN ------ SCQ ------ BCN
                0810-0915  1130-1230  1505-1640  1735-1905

**************************************************************************
**********************   S O L U C I Ó N   9   ***************************
**************************************************************************

 fecha =                             09/09/2012
 horas de vuelo realizadas =         6 horas y 35 minutos (6.58)

 numLineas =        4626       4627       4662       4659
              BCN ------ FCO ------ BCN ------ MXP ------ BCN
                0845-1025  1140-1320  1510-1640  1735-1920
```

```
***************************************************************************
**********************    S O L U C I Ó N   10  ***************************
***************************************************************************

 fecha =                             12/09/2012
 horas de vuelo realizadas =         7 horas y 15 minutos (7.25)

 numLineas =        6181       3456       3457       6970
               BCN ------ MAD ------ NCE ------ MAD ------ LPA
                 0855-0955  1050-1230  1320-1505  1635-1925

***************************************************************************
**********************    S O L U C I Ó N   11  ***************************
***************************************************************************

 fecha =                             13/09/2012
 horas de vuelo realizadas =         7 horas y 30 minutos (7.50)

 numLineas =        807       3502       3503
               LPA ------ MAD ------ FRA ------ MAD
                 1120-1350  1505-1735  1825-2055

***************************************************************************
**********************    S O L U C I Ó N   12  ***************************
***************************************************************************

 fecha =                             07/09/2012
 horas de vuelo realizadas =         4 horas y 15 minutos (4.25)

 numLineas =        4424       4427       1945
               BCN ------ ORY ------ BCN ------ MAD
                 1315-1455  1545-1720  1845-1945

***************************************************************************
**********************    S O L U C I Ó N   13  ***************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         4 horas y 35 minutos (4.58)

 numLineas =        548       551       3244
               MAD ------ SCQ ------ MAD ------ AMS
                 1505-1615  1705-1810  1910-2130

***************************************************************************
**********************    S O L U C I Ó N   14  ***************************
***************************************************************************

 fecha =                             10/09/2012
 horas de vuelo realizadas =         4 horas y 35 minutos (4.58)

 numLineas =        4570       3531
               BCN ------ MUC ------ MAD
                 1515-1715  1800-2035

***************************************************************************
**********************    S O L U C I Ó N   15  ***************************
***************************************************************************

 fecha =                             05/09/2012
 horas de vuelo realizadas =         1 horas y 10 minutos (1.16)

 numLineas =        554
               MAD ------ SCQ
                 1745-1855
```

```
*****************************************************************************
**********************   S O L U C I Ó N   16  *************************
*****************************************************************************

 fecha =                            04/09/2012
 horas de vuelo realizadas =        2 horas y 25 minutos (2.41)

 numLineas =        3249
              AMS ------ MAD
                1840-2105
```

Fig. 8: Results obtained with our application when we use same data as Iberia airline.

All algorithm results match exactly with Iberia sample, but ordered by flight hours. In our results in "horas de vuelo realizadas =", we've parenthesized the percent of flight hours done, so we can compare them with Iberia's ones in "H. V" column, made in the same format. As we can note, all data are the same except rounding digits and flights whose number begins with VS (dead head). In that situation, flight hours only count half of real hours, because the airline pays only for "working hours". We haven't taken it into account because it depends on each airline

Second tests set has been made taking into consideration the main objective of our algorithm. In other words, we want to get maximum efficiency for several flights in the same date and with the same parameters. Output data obtained with our algorithm is:

```
*****************************************************************************
**********************   S O L U C I Ó N   1  **************************
*****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        6 horas y 20 minutos (6.33)

 numLineas =        1702       1707       4626       4627       4219
              BCN ------ PMI ------ BCN ------ FCO ------ BCN ------ SCQ
                0550-0630  0715-0800  0845-1025  1140-1320  1505-1640

*****************************************************************************
**********************   S O L U C I Ó N   2  **************************
*****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        7 horas y 30 minutos (7.50)

 numLineas =        4422       4423       4424       4427       1945
              BCN ------ ORY ------ BCN ------ ORY ------ BCN ------ MAD
                0625-0805  0855-1030  1315-1455  1545-1720  1845-1945
```

```
****************************************************************************
**********************   S O L U C I Ó N   3  ******************************
****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        7 horas y 35 minutos (7.58)

 numLineas =        800       4626       4627       4662       4659
              MAD ------ BCN ------ FCO ------ BCN ------ MXP ------ BCN
                0700-0800  0845-1025  1140-1320  1510-1640  1735-1920

****************************************************************************
**********************   S O L U C I Ó N   4  ******************************
****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        6 horas y 20 minutos (6.33)

 numLineas =        800       4626       4627       4570
              MAD ------ BCN ------ FCO ------ BCN ------ MUC
                0700-0800  0845-1025  1140-1320  1515-1715

****************************************************************************
**********************   S O L U C I Ó N   5  ******************************
****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        6 horas y 25 minutos (6.41)

 numLineas =        800       6181       3456       3457       1700
              MAD ------ BCN ------ MAD ------ NCE ------ MAD ------ BCN
                0700-0800  0855-0955  1050-1230  1320-1505  1600-1700

****************************************************************************
**********************   S O L U C I Ó N   6  ******************************
****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        5 horas y 25 minutos (5.41)

 numLineas =        800       6181       3456       3457
              MAD ------ BCN ------ MAD ------ NCE ------ MAD
                0700-0800  0855-0955  1050-1230  1320-1505

****************************************************************************
**********************   S O L U C I Ó N   7  ******************************
****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        3 horas y 20 minutos (3.33)

 numLineas =        3207      2600
              BRU ------ MAD ------ BCN
                0715-0935  1130-1230

****************************************************************************
**********************   S O L U C I Ó N   8  ******************************
****************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        5 horas y 40 minutos (5.66)

 numLineas =        541       548       551       3244
              SCQ ------ MAD ------ SCQ ------ MAD ------ AMS
                0810-0915  1505-1615  1705-1810  1910-2130
```

```
***************************************************************************
**********************   S O L U C I Ó N   9  ***************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         6 horas y 0 minutos (6.00)

 numLineas =        6181       3502       3503
              BCN ------ MAD ------ FRA ------ MAD
                0855-0955  1505-1735  1825-2055

***************************************************************************
**********************   S O L U C I Ó N   10  **************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         2 horas y 5 minutos (2.08)

 numLineas =        4216
              BCN ------ BRU
                0900-1105

***************************************************************************
**********************   S O L U C I Ó N   11  **************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         5 horas y 20 minutos (5.33)

 numLineas =        807       6970
              LPA ------ MAD ------ LPA
                1120-1350  1635-1925

***************************************************************************
**********************   S O L U C I Ó N   12  **************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         1 horas y 30 minutos (1.50)

 numLineas =        1569
              SCQ ------ BCN
                1735-1905

***************************************************************************
**********************   S O L U C I Ó N   13  **************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         1 horas y 10 minutos (1.16)

 numLineas =        554
              MAD ------ SCQ
                1745-1855

***************************************************************************
**********************   S O L U C I Ó N   14  **************************
***************************************************************************

 fecha =                             03/09/2012
 horas de vuelo realizadas =         2 horas y 35 minutos (2.58)

 numLineas =        3531
              MUC ------ MAD
                1800-2035
```

```
************************************************************************
**********************   S O L U C I Ó N   15  **************************
************************************************************************

 fecha =                            03/09/2012
 horas de vuelo realizadas =        2 horas y 25 minutos (2.41)

 numLineas =        3249
              AMS ------ MAD
                1840-2105
```

Fig. 9: Results obtained with our application when we use same data as Iberia but with different date.

Here with our algorithm, we fit crews in a better way reordering air routes.

ARGUMENTING RESULTS

As we can see, when we use the same conditions as any other airline, we obtain the same results (assigning same date to an air routes group). Even though this tests group has no practical value, it enables us to check algorithm's effectiveness. So we can confirm it makes exactly what it should make.

The other columns in Iberia sample (distinct from "H. V"), have no interest because data is calculated depending on flight hours and on legal terms. So we get the same result. For example, let's take ALR (personal work made) which adds several hours to flight hours due to stopovers; ALP (maximum work allowed) which is the maximum number of hours permitted depending on timetables and on the number of air routes made, etc.

Instead, in the second tests set, we get different solutions. As we can see, we can improve original results:

	Legs	R. F. H	O. F. H	F. H. W	N. O. P	M. V. L	N. P. M	M. V. F. H	F. H. A
Iberia	44	69.61	63.07	3.29	16	3	1	4.33	4.35
First tests group	44	69.61	69.61	__	16	3	1	4.33	4.35
Second tests group	44	69.61	69.61	__	15	3	4	5.41	4.64
Total flight hours = 69 hours 36 minutes 36 seconds Number of air routes = 44									

Table 1: Table showing the difference between Iberia and our algorithm results

Caption:

- R. F. H = Real flight hours
- O. F. H = Obtained flight hours
- F. H. W = Flight hours without working
- N. O. P = Number of pairings
- M. V. L = Mean value of legs per pairing
- N. P. M = Number of pairings with maximum number of legs (5 legs)
- M. V. F. H = Mean value of flight hours per pairing
- F. H. A = Flight hours average

In spite of not being a wide test (44 flight legs), we have made some good progress compared with Iberia airline. We have succeed saving one crew (one pairing less) with the same air routes average in each pairing, and just growing up a little flight hours average. We also increase the mean value of hours per pairing obtaining a bigger adjustment. This way we get more air routes per pairing but with the same air routes average. So, we can have a balanced distribution too.

Also we see it exists a difference between Iberia obtained flight hours and the ones in my research. That is due to that Iberia counts dead heads half. So, if we multiply "F. H. W" with 2 and we add "O. F. H", we get 69,65 flight hours, instead of 69,61 due to rounded numbers. As we said previously, our algorithm doesn't consider this scenario.

CONCLUSION

Thanks to Java programming and to the use of Free Software (as Eclipse or Java), we finally get a solution to Crew Pairing Problem developing our own algorithm. Airlines get this target (link different legs to obtain pairings that achieve all constraints needed) thanks to pairings manual development with expensive software and several human resources costs Other software companies as CARMEN which was bought by Jeppesen company (http://ww1.jeppesen.com/index.jsp) in 2006, belongs to Boeing company. In particular, they work with all aspects of scheduling as Jeppesen Crew Pairing (http://ww1.jeppesen.com/industry-solutions/aviation/commercial/carmen-crew-pairing.jsp). When we develope our own algorithm we get same target with lower costs and with low computing times, even with wide number of daily flights in big airlines.

Also we can adjust results only modifying parameters. For example, if we want to get pairings with more legs per crew member, we only have to increase the number of flight hours per day. On the other hand, if we want to balance the number of legs per crew, we only have to reduce the maximum number of legs' parameter. This way, we obtain better balanced pairings. We must take into account that all those modifications have to operate within the law.

Anyway, we let the possibility to other people to improve this algorithm through other mathematical techniques, as could be applying random data to input data, or applying statistic techniques to ameliorate algorithm's efficiency. In the future, it would be a good to continue with my study to enhance it and use this research as basement of others as could be the Crew Rostering Problem.

REFERENCES

– *AhmadBeygi, Shervin; Cohn, Amy and Lapp, Marcial, "Decreasing airline delay propagation by re-allocating scheduled slack". IEE Transactions. Vol. 42, no. 7, pages 478-489. 2010*

– *Angel A. Juan, Javier Faulin, Rubén Ruiz, Barry Barrios and Santi Caballé, "The SR-GCWS hybrid algorithm for solving the capacited vehicle routing problem". Applied Soft Computing 10, pages 215-224. 2010*

– *Angel A. Juan, Javier Faulin, J. Jorba, D. Riera, D. Masip and Barry Barrios, "On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics". Journal of the Operational Research Society 62, pages 1085-1097. 2011*

– *Bazargan Massoud, "Airline Operations and Scheduling 2nd Edition". Ashgate Publishing Limited, Embry-Riddle Aeronautical University, USA. 2010*

– *Chiraphadhanakul, Virot and Barnhart, Cynthia, "Robust flight schedules through slack re-allocation". Operations Research Center, Massachusetts Institute of Technology. 2011*

– *Deng, Guang Fen and Lin, Woo-Tsong, "Ant Colony optimization-based algorithm for airline crew scheduling problem". Expert Systems with Applications. Vol. 38, no. 5, pages 5787-5793. 2011*

– *Dunbar, Michelle and Froyland, Gary, "Robust Airline schedule planning: Minimizing propagated delay in an integrated routing and crewing framework". School of Mathematics and Statistics, University of New South Wales. 2010*

– *El Moudani, W; Brochado MR; Handou, M and Mora-Camino, F, "A fuzzy reactive approach for the crew rostering problem". Current Advances in Mechanical Design and Production VII. Vol. 7, pages 611-619. 2000*

– *Gopalakrishnan, Balaji and Johnson, Ellis J. "Airline Crew Scheduling: state-of-the-Art", Annals of Operations Research 140, pages 305-337. 2005*

– *Graves, G. W., R. D. McBride and Gershkoff. "Flight Crew Scheduling". Management Science 39(6), pages 736-745. 1993*

– *Medard, Claude P and Sawhney, Nidhi, "Airline Crew Scheduling from planning to operations". European Journal of Operational Research. Vol. 38, no. 3, pages 1013-1027. 2007*

– *Mercier, A; Cordeau, JF and Soumis, F, "A computacional study of Benders decomposition for the integrated aircraft routing and crew scheduling problem". Computers & Operations Research. Vol. 32, no. 6, pages 1451-1476. 2005*

– *Ozdemir, HT and Mohan, CK, "Flight graph based genetic algorithm for crew scheduling in airlines". Information Sciencies. Vol. 133, no. 3-4, pages 165-173. 2001*

– *Preston. "Airline Crew Scheduling". Preston Group Pty Ltd. 1992*

– *Wei, Guo; Yu, Gang and Song, Mark. "Optimization model and algorithm for crew management during irregular operations". Journal of Combinatorial Optimization. Vol. 1, pages 80-97. 1997*

– *Weide, Oliver; Ryan, David and Ehrgott, Matthias, "An iterative approach to robust and integrated aircraft routing and crew scheduling". Computers & Operations Research. Vol. 37, no. 5, pages 833-844. 2010*

– *Yen, JW and Birge, JR, "A Stochastic programming approach to the airline crew scheduling problem". Transportation Science. Vol. 40, no. 1, pages 3-14. 2006*

– *Yu, Gang; Arguello, Michael; Song, Gao; McCowan, Sandra and White, Anna. "A new era for crew recovery at Continental Airlines". Interfaces 33. Vol. 1, pages 5-22. 2003*

APPENDIX

- **Application's Java code:**

```
package CrewPairingP;

/*****************************************************************************
* Project SimORouting -  ElapsedTime.java
* Given an elapsed time in seconds, the method printHMS() of this class returns
*  an string with the equivalent time in hours, minutes and seconds.
* Given an elapsed time in seconds, the method doPause() of this class forces
*  the code to do a pause for a time interval of that size.
* Date of last revision (YYMMDD): 101224
* (C) Angel A. Juan - ajuanp(@)gmail.com
```

```
*****************************************************************************/
public class ElapsedTime
{
    public ElapsedTime(){}

    public static long systemTime()
    {   long time = System.nanoTime();
        return time;
    }

    public static double calcElapsed(long start, long end)
    {   double elapsed = (end - start) / 1.0e+9;
        return elapsed;
    }

    public static String calcElapsedHMS(long start, long end)
    {   String s = "";
        double elapsed = (end - start) / 1.0e+9;
        s = s + calcHMS((int) Math.round(elapsed));
        return s;
    }

    public static String calcHMS(int timeInSeconds)
    {   String s = "";
        int hours, minutes, seconds;
        hours = timeInSeconds / 3600;
        timeInSeconds = timeInSeconds - (hours * 3600);
        minutes = timeInSeconds / 60;
        timeInSeconds = timeInSeconds - (minutes * 60);
        seconds = timeInSeconds;
        s = s + hours + "h " + minutes + "m " + seconds + "s";
        return s;
    }

    public static void doPause(int timeInSeconds)
    {   long t0, t1;
        t0 = System.currentTimeMillis();
        t1 = System.currentTimeMillis() + (timeInSeconds * 1000);
        do
        {   t0 = System.currentTimeMillis();
        } while (t0 < t1);
    }
}
package CrewPairingP;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

/*****************************************************************************
 *                                                                           *
 * Clase encargada de gestionar el programa. Lee los datos del fichero       *
 * test.txt de prueba, trata las condiciones necesarias para generar         *
 * las soluciones al generar los tramos de vuelos con los Legs y muestra     *
 * una salida formateada con los tramos solución                             *
 *                                                                           *
 *                                            CREW PAIRING PROBLEM           *
 *                                        Proyecto Fin de Master - U O C        *
*                                                                               *
 * @author Jesús Manuel Puentes Gutiérrez                                       *
 *                                                                              *
 * @director Angel A. Juan                                                      *
 *                                                                              *
 ******************************************************************************/


/**
 * @author shuli
 *
 */
public class Inicio {

    final static String inputFolder = "inputs";// + File.separator + "TSP";
    final static String fileNameTest = "test500.txt";//fichero de prueba
                                                       test.txt
    final static String outputFolder = "outputs";//directorio de salida
    final static String fileNameOutput = "out500.txt";//fichero con las
                                                             soluciones

    final static int TIEMPO_ESCALA_MIN = 60;   // Tiempo mínimo de escala
                                                       en minutos
    final static int NUM_MAX_SALTOS = 5;
    final static int HORAS_MAX_VUELO = 800;    // Equivale a 8 horas de
                                                       vuelo máximas

      /**
       * Método principal de la aplicación Crew Pairing Problem
       *
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
        System.out.println("****  WELCOME TO THIS PROGRAM  ****");
        long programStart = ElapsedTime.systemTime();

        String testsFilePath = inputFolder + File.separator + fileNameTest;
        String outputFilePath = outputFolder + File.separator + fileNameOutput;

        // Comprobamos si existe un archivo de solución anterior y lo borramos
        // si es así. De este modo se generan soluciones limpias cada vez
        String sFichero = outputFilePath;
        File fichero = new File(sFichero);

          if (fichero.exists()) {
            fichero.delete();
          }

        // Read inputs files and construct the inputs object
        Inputs inputs = InputsManager.getInputs(testsFilePath);

       // Mostramos un listado con todos los legs disponibles que hemos le�do de
       // nuestro archivo de datos
      System.out.println("\n--- Lista con Legs disponibles (sin ordenar)
                        ---\n");
      System.out.println(inputs.showLegList());
```

```
        ArrayList<Leg> legs = new ArrayList<Leg>();
        ArrayList<Leg> ruta = new ArrayList<Leg>();

        Leg l;
        boolean primero = true;
        String solution = "";
        int numSols = 0;
        int numSaltos = 0;
        int numHorasVuelo = 0;
        int totalHorasVuelo = 0;

        // Genero nueva lista
        for (int i=0; i < inputs.getLegList().length; i++)
            legs.add(inputs.getLegList()[i]);

        // Ordenamos la lista de Legs seg�n la hora de inicio del vuelo
            sortList(legs);

      // Seleccionamos los datos necesarios del primer elemento de la lista
      String destino = legs.get(0).getDestino();
      int horaInicio = legs.get(0).getHoraInicio();
      int horaFin = legs.get(0).getHoraFin();
      String fecha = legs.get(0).getFecha();

        do{
            // Vaciamos la ruta para empezar otro tramo
            ruta.clear();
            // Reiniciamos el contador de tramos
            numSaltos = 1;


            if (primero == true){
                  // Seleccionamos los datos necesarios del primer elemento de
                  la lista
                  destino = legs.get(0).getDestino();
                  horaInicio = legs.get(0).getHoraInicio();
                  horaFin = legs.get(0).getHoraFin();
                  fecha = legs.get(0).getFecha();
                  // Añadimos el primer elemento del tramo (si es el primero
                  utilizado)
                  ruta.add(legs.get(0));
                  // Calculamos las horas de vuelo del tramo
                  numHorasVuelo = calcFlightHours (horaInicio, horaFin);
                  // Sumamos al total de horas de vuelo que tenemos
                  totalHorasVuelo = addFlightHours(totalHorasVuelo,
                              numHorasVuelo);
                  // Borramos el elemento usado (si es el primero utilizado)
                  legs.remove(0);
                  primero = false;
            }


            do {
                  // Buscamos un Leg solucion
                        l = searchSol(legs, destino, horaInicio, horaFin, fecha,
                                    numSaltos, totalHorasVuelo);

                  if (l != null){
                        // Añadimos el Leg a la ruta
                        ruta.add(l);
                        // Contamos el salto
                        numSaltos++;

                        destino = l.getDestino();
                        horaInicio = l.getHoraInicio();
                        horaFin = l.getHoraFin();
                        fecha = l.getFecha();

                        // Calculamos las horas de vuelo del tramo
                        numHorasVuelo = calcFlightHours (horaInicio, horaFin);
                        // Sumamos al total de horas de vuelo que tenemos
                        totalHorasVuelo = addFlightHours(totalHorasVuelo,
                                          numHorasVuelo);

                        // Borramos el Leg utilizado
                        int indice = legs.indexOf(l);
                        legs.remove(indice);
                  } else {
                        // Imprimimos solución en pantalla
                        numSols++;
                        solution = showSol(ruta, numSols, totalHorasVuelo);

                        // Guardamos la solución en un archivo de texto
                        (out.txt)
                        saveSolution(solution, outputFilePath);

                        primero=true;
                        totalHorasVuelo = 0;

                        }

            } while (l != null);  // hasta que no existan coincidencias de Legs
        } while (legs.size() > 0); // hasta que se vacíe la lista de Legs


        /*******************************************************************
        * END OF PROGRAM
        *******************************************************************/
        System.out.println("\n****  END OF PROGRAM  ****");
            long programEnd = ElapsedTime.systemTime();
            System.out.println("Total elapsed time = "
                + ElapsedTime.calcElapsedHMS(programStart, programEnd));

      }

      /**
       * Método para ordenar la lista de Legs
       *
       * @param legs
       */
      @SuppressWarnings("unchecked")
      private static void sortList(ArrayList<Leg> legs) {
            Collections.sort(legs);
      }

      /**
       * Función que busca una solución factible con los condicionantes
       * requeridos
```

```
     *
     * @param legs
     * @param destino
     * @param horaInicio
     * @param horaFin
     * @param fecha
     * @param numSaltos
     * @param numHorasVuelo
     * @return Devuelve un Leg que cumple las condiciones impuestas
     */
    public static Leg searchSol(ArrayList<Leg> legs, String destino, int
          horaInicio, int horaFin, String fecha, int numSaltos, int
          numHorasVuelo){
          Leg sol = null;
          boolean found = false;
          int horas = 0;
          int suma = 0;

          for (Leg i: legs)
                if ((i.getFecha().equals(fecha)) &&
                      (i.getOrigen().equals(destino)) && ((calcDifMin(horaFin,
                      i.getHoraInicio())) >= TIEMPO_ESCALA_MIN) &&
                      (i.getHoraInicio() > horaInicio) && (numSaltos <
                      NUM_MAX_SALTOS) && (numHorasVuelo <= HORAS_MAX_VUELO)) {
                      horas = calcFlightHours(i.getHoraInicio(),
                                              i.getHoraFin());
                      suma = suma + horas;
                      suma = addFlightHours(suma, numHorasVuelo);
                      if ((found == false) && (suma <= HORAS_MAX_VUELO)) {
                            sol = i;
                            found = true;
                      }
                }

          return sol;
    }

    /**
     * Método que nos muestra la solución con un formato específico de algunas
     * compañías aéreas
     *
     * @param leg
     * @param numSol
     * @param hVuelo
     * @return Devuelve cadena con la solución
     */
    public static String showSol(ArrayList<Leg> leg, int numSol, int hVuelo) {
          String sol = "";
          String fechaL = " fecha =                             ";
          String f = "";
          String horasVuelo = " horas de vuelo realizadas =         ";
          String nLin = " numLineas =        ";
          String tramos = "";
          String horas = "";
          String horaInicio = "";
          String horaFin = "";

          sol +=
          "********************************************************************
```

```
            ********\n";
            if (((numSol / 10) > 0) && ((numSol / 10) < 10))
                  sol += "**********************   S O L U C I Ó N   " + numSol
                  + "  **************************\n";
            else if (((numSol / 10) > 9) && ((numSol / 10) < 100))
                  sol += "**********************   S O L U C I Ó N   " + numSol
                  + "  *************************\n";
            else if (((numSol / 10) > 99) && ((numSol / 10) < 1000))
                  sol += "**********************   S O L U C I Ó N   " + numSol
                  + "  ************************\n";
            else
                  sol += "**********************   S O L U C I Ó N   " + numSol
                  + "  ***************************\n";
            sol +=
            "*******************************************************************
            ********\n";


            for (int i = 0; i < leg.size(); i++){

                  if (f == "") {
                        f = leg.get(i).getFecha();
                        fechaL += f;
                  }

                  nLin += leg.get(i).getNumLinea() + "       ";

                  if (i == 0){
                        tramos += "               " + leg.get(i).getOrigen() + "
                        ------ " + leg.get(i).getDestino();
                        horaInicio = String.format("%04d",
                              leg.get(i).getHoraInicio());
                        horaFin = String.format("%04d",
                              leg.get(i).getHoraFin());
                        horas += "                 " + horaInicio + "-" +
                              horaFin;
                  } else {
                        tramos += " ------ " + leg.get(i).getDestino();
                        horaInicio = String.format("%04d",
                              leg.get(i).getHoraInicio());
                        horaFin = String.format("%04d",
                              leg.get(i).getHoraFin());
                        horas += "  " + horaInicio + "-" + horaFin;
                  }

            }

            minDecimal = ((hVuelo%100) * 100) / 60;
            if (minDecimal < 10) {
                  resulMinDec = "0" + minDecimal;
            } else {
                  resulMinDec = "" + minDecimal;
            }

            sol += "\n" + fechaL + "\n" + horasVuelo + (hVuelo/100) + " horas y
            " + (hVuelo%100) + " minutos" + " (" + (hVuelo/100) + "." +
            resulMinDec + ")" + "\n\n" + nLin + "\n" + tramos + "\n" + horas +
            "\n\n";
```

```
		System.out.println(sol);

		return sol;
	}


	/**
	 * Método que graba en un archivo de texto las soluciones encontradas,
	 * añadiéndolas al archivo cada vez
	 *
	 * @param solution
	 * @param archivo
	 */
	public static void saveSolution(String solution, String archivo) {
		try {
			FileWriter flS = new FileWriter(archivo, true);
			BufferedWriter fS = new BufferedWriter(flS);

			fS.write(solution);
			fS.close();
		}catch(IOException e) {
			System.out.println("Error E/S en fichero escritura");
		}
	}

	/**
	 * Método que calcula las horas de vuelo realizadas en un leg, pasándole
	 * las horas a la que comienza y finaliza dicho leg, devolviendo el
	 * número de horas realizadas como un entero
	 *
	 * @param horaInicio
	 * @param horaFin
	 * @return Devuelve las horas de vuelo realizadas como un entero
	 */
	public static int calcFlightHours(int horaInicio, int horaFin) {
		int minInicio = 0;
		int minFin = 0;
		int horasInicio = 0;
		int horasFin = 0;
		int minTotal = 0;
		int horasTotal = 0;
		int tiempoTotal = 0;
		int minTotalesInicio = 0;
		int minTotalesFin = 0;
		int tiempoTotalMin = 0;

		minInicio = horaInicio % 100;
		minFin = horaFin % 100;
		horasInicio = horaInicio / 100;
		horasFin = horaFin / 100;

		minTotalesInicio = (horasInicio * 60) + minInicio;
		minTotalesFin = (horasFin * 60) + minFin;

		// Si horaFin < horaInicio es que pasamos de día, por lo que
		// debemos sumar 24 horas (1440 minutos)
		if (horaFin < horaInicio)
			minTotalesFin = minTotalesFin + 1440;
```

```
            tiempoTotalMin = minTotalesFin - minTotalesInicio;
            horasTotal = tiempoTotalMin / 60;
            minTotal = tiempoTotalMin % 60;

            tiempoTotal = (horasTotal * 100) + minTotal;

            return tiempoTotal;
      }

      /**
       * Método que calcula la diferencia de tiempo en minutos que existe entre
       * dos horas pasadas como parámetros
       * @param horaInicial
       * @param horaFinal
       * @return Tiempo existente en minutos entre dos horas dadas
       */
      public static int calcDifMin(int horaInicial, int horaFinal) {
            int minInicio = 0;
            int minFin = 0;
            int horasInicio = 0;
            int horasFin = 0;
            int minTotalesInicio = 0;
            int minTotalesFin = 0;
            int difMinTiempo = 0;

            minInicio = horaInicial % 100;
            minFin = horaFinal % 100;
            horasInicio = horaInicial / 100;
            horasFin = horaFinal / 100;

            minTotalesInicio = (horasInicio * 60) + minInicio;
            minTotalesFin = (horasFin * 60) + minFin;

            difMinTiempo = minTotalesFin - minTotalesInicio;

            return difMinTiempo;
      }

      /**
       * Método que va acumulando las horas de vuelo realizadas y
       * las devuelve como un entero
       *
       * @param totalHorasVuelo
       * @param numHorasVuelo
       * @return Devuelve las horas de vuelo acumuladas como un entero
       */
      public static int addFlightHours(int totalHorasVuelo, int numHorasVuelo) {
            int minTotal = 0;
            int horasTotal = 0;
            int minHV = 0;
            int horasHV = 0;
            int sumaMin = 0;
            int sumaHoras = 0;
            int totalMin = 0;
            int totalH = 0;
            int totalM = 0;
            int totalTiempo = 0;

            minTotal = totalHorasVuelo % 100;
            horasTotal = totalHorasVuelo / 100;
            minHV = numHorasVuelo % 100;
            horasHV = numHorasVuelo / 100;

            sumaMin = minTotal + minHV;
            sumaHoras = (horasTotal * 60) + (horasHV * 60);

            totalMin = sumaMin + sumaHoras;

            totalH = totalMin / 60;
            totalM = totalMin % 60;

            totalTiempo = (totalH * 100) + totalM;

            return totalTiempo;
      }
}

package CrewPairingP;

/*******************************************************************************
 *                                                                             *
 * Clase encargada de contener listado de Legs leído del archivo de datos      *
 *                                                                             *
 *                                            CREW PAIRING PROBLEM             *
 *                                     Proyecto Fin de Master - U O C        *
 *                                                                             *
 * @author Jesús Manuel Puentes Gutiérrez                                      *
 *                                                                             *
 * @director Angel A. Juan                                                     *
 *                                                                             *
 *******************************************************************************/


public class Inputs {
      private Leg[] legList;

      /* Constructor */
      public Inputs(int nLegs) {
            legList = new Leg[nLegs];
      }

      /* getters */
      public Leg[] getLegList() {
            return legList;
      }

      /* Muestra lista de Legs */
      public String showLegList() {
            String temp="";
            for (Leg l: legList)
                  temp += l.toString();
            return temp;
      }

}

package CrewPairingP;
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

/*******************************************************************************
 *                                                                             *
 * Clase encargada de leer los datos del fichero de datos para generar la      *
 * clase Legs con su correspondiente constructor. Utiliza Scanner para leer    *
 * los datos del fichero                                                       *
 *                                                                             *
 *                                                                             *
 *                                            CREW PAIRING PROBLEM             *
 *                                      Proyecto Fin de Master - U O C          *
*
 * @author Jesús Manuel Puentes Gutiérrez                                      *
 *                                                                             *
 * @director Angel A. Juan                                                     *
 *                                                                             *
 ******************************************************************************/


public class InputsManager {
      public static Inputs getInputs(String legsFilePath) {
            Inputs inputs = null;
        try
        {   // 1. COUNT THE # OF LEGS (# OF LINES IN legsFilePath)
           BufferedReader br = new BufferedReader(new FileReader(legsFilePath));
            String f = null;
            int nLegs = 0;
            while( (f = br.readLine()) != null )
            {   if( f.charAt(0) != '#' )
                    nLegs++;
            }

            // 2. CREATE THE INPUTS OBJECT WITH nLegs
            inputs = new Inputs(nLegs);

            FileReader reader = new FileReader(legsFilePath);
            Scanner in = new Scanner(reader);
            int k = 0;

            while( in.hasNextLine() )
            {   String s = in.next();
                if( s.charAt(0) == '#' )
                    in.nextLine();
                else
                {   int numLinea = Integer.parseInt(s);
                    String origen = in.next();
                    String destino = in.next();
                    int horaInicio = in.nextInt();
                    int horaFin = in.nextInt();
                    String fecha = in.next();
                   Leg leg = new Leg(numLinea, origen, destino, horaInicio,
                              horaFin, fecha);
                    inputs.getLegList()[k] = leg;
                    k++;
                }
            }
            in.close();

        }
        catch (IOException exception)
        {   System.out.println("Error processing inputs files: " + exception);
        }
        return inputs;
      }
}

package CrewPairingP;

/********************************************************************************
 *                                                                              *
 * Clase que contiene los datos de los Legs, con sus correspondientes          *
 * Getters y Setters, así como un formato básico de salida toString()          *
 * y un comparador para ordenar los Legs por la hora de inicio del Leg         *
 *                                                                              *
 *                                             CREW PAIRING PROBLEM             *
 *                                       Proyecto Fin de Master - U O C          *
*                                                                               *
 * @author Jesús Manuel Puentes Gutiérrez                                       *
 *                                                                              *
 * @director Angel A. Juan                                                      *
 *                                                                              *
 ********************************************************************************/
@SuppressWarnings("rawtypes")
public class Leg implements Comparable {

      /* Instance fields */
      private int numLinea;
      private String origen;
      private String destino;
      private int horaInicio;
      private int horaFin;
      private String fecha;

      /* Constructor */
      public Leg(int numLinea, String origen, String destino, int horaInicio,
                  int horaFin, String fecha) {
            super();
            this.numLinea = numLinea;
            this.origen = origen;
            this.destino = destino;
            this.horaInicio = horaInicio;
            this.horaFin = horaFin;
            this.fecha = fecha;
      }

      /* Getters and Setters */
      public int getNumLinea() {
            return numLinea;
      }
      public void setNumLinea(int numLinea) {
            this.numLinea = numLinea;
      }
      public String getOrigen() {
            return origen;
      }
	public void setOrigen(String origen) {
		this.origen = origen;
	}
	public String getDestino() {
		return destino;
	}
	public void setDestino(String destino) {
		this.destino = destino;
	}
	public int getHoraInicio() {
		return horaInicio;
	}
	public void setHoraInicio(int horaInicio) {
		this.horaInicio = horaInicio;
	}
	public int getHoraFin() {
		return horaFin;
	}
	public void setHoraFin(int horaFin) {
		this.horaFin = horaFin;
	}
	public String getFecha() {
		return fecha;
	}
	public void setFecha(String fecha) {
		this.fecha = fecha;
	}

	/**
	 * Método que devuelve una salida formateada de Legs
	 */
	@Override
	public String toString() {
		return "\nLeg [numLinea=" + numLinea + "\n\t\t\t " + origen
				+ " --- " + destino + "\n\t horaInicio=" + horaInicio
				+ ", horaFin=" + horaFin + ", fecha=" + fecha + "]\n\n";
	}

	/**
	 * Método para ordenar objetos según la hora de inicio del Leg
	 */
	public int compareTo(Object o) {
		Leg leg = (Leg)o;

		if(this.horaInicio == leg.getHoraInicio())
			return 0;
		else if (this.horaInicio < leg.getHoraInicio())
			return -1;
		else
			return 1;
	}
}

package CrewPairingP;

/*******************************************************************************
 *                                                                             *
 * Clase que gestiona un error de ejecución                                    *
 *                                                                             *
 *                                              CREW PAIRING PROBLEM           *
 *                                        Proyecto Fin de Master - U O C      *
 *                                                                             *
 * @author Jesús Manuel Puentes Gutiérrez                                      *
 *                                                                             *
 * @director Angel A. Juan                                                     *
 *                                                                             *
 ******************************************************************************/

@SuppressWarnings("serial")
public class LegException extends RuntimeException{
      LegException(String mensaje){
            super(mensaje);
      }
}
```

- Application's input data:

```
...
3028  MXP  BCN  1850  1950  03/09/2012
2050  BCN  MAD  2050  2140  03/09/2012
2240  MAD  BCN  2240  2310  03/09/2012
1010  BCN  VLC  1420  1440  03/09/2012
1009  VLC  BCN  1540  1610  03/09/2012
...
```

We show a partial capture of all input data. We can surf in the following Internet direction http://bscw.uoc.es to obtain all input data used. You should be previously authorized to access them by project's director and university.