Crew Pairing Optimization Based on CLP

Christina Pavlopoulou* Aristides P. Gionis†
Panagiotis Stamatopoulos Constantin Halatsis

University of Athens, Department of Informatics
Panepistimiopolis, 157 71 Athens, Greece
E-mail: {pavlo,gionis,takis,halatsis}@di.uoa.gr

Abstract

The crew pairing optimization problem is faced by airline companies as an intensive part of the crew scheduling process. Crew scheduling is the assignment of cockpit and cabin crews to the flight legs that a company has to carry out during a predefined period of time. Due to the significant contribution of the crew cost to the overall operating cost of an airline company, the automation of the crew scheduling procedure is highly desirable. However, the crew pairing optimization subproblem of crew scheduling is extremely difficult and combinatorial in nature due to the large number and complexity of the involved constraints. The requirement for optimality makes it even more difficult. Many attempts have been made in the past 40 years to tackle the crew pairing optimization problem using methods from Operations Research. In this paper, an approach based on pure Constraint Logic Programming is presented, which leads to an elegant and flexible modeling of the problem. The whole process is divided into three distinct phases, the job construction, the pairing construction and a set partitioning problem. All three phases are viewed as constraint satisfaction problems. A prototype system that has been built using the Constraint Logic Programming language ECL^{i}PS^e is also presented.

1 Introduction

The cost for flying crews is very high for all airline companies and is second only to fuel cost. However, while for the fuel cost there is no much potential for reductions, the crew cost can be controlled through better crew utilization. This is the reason why most carriers have invested a lot of money, even since the early 60s, to this direction [Spi61, AFST69, Rub73, MMK79, MS81, Ger89, HP93].

*Currently at Purdue University, Department of Computer Sciences.
†Currently at University of Edinburgh, Department of Artificial Intelligence.

The problem that needs to be solved is the automation of the *crew scheduling* procedure. The input to this problem is the published flight schedule of the company comprising the departure and arrival airports, the departure and arrival times as well as the aircraft types for all flights (or *flight legs*, or just *legs*) which have to be carried out during a predefined period of time. This period is usually one month. A flight leg is an elementary non-stop transportation between two specific sites. What has to be computed is a feasible allocation of the required cockpit and cabin crew personnel to every flight leg. This computation is extremely difficult since there is a huge search space to be explored and a lot of constraints to be respected (e.g. governmental and international regulations, contractual obligations, company restrictions etc.). It is even more difficult when there is a requirement for the optimum solution, with respect to a given well-defined cost function. The optimum solution is the one which minimizes the cost function, so as to achieve the lowest possible crew operating cost or, equivalently, the highest possible crew utilization.

Although the whole crew scheduling process might be formulated as a single, but quite complex, constraint satisfaction problem, it is normally tackled in two phases for reasons of management of its combinatorial nature. The first phase is usually referred to as the *anonymous crew scheduling* and the second as the *crew rostering*. The output of the anonymous crew scheduling is a set of *pairings*, i.e. sequences of flight legs that cover exactly the flight schedule of the company. Each pairing has the property that can be flown by a single crew, however the specific crew members which are to be assigned to the pairing will be computed during the next phase, the crew rostering [BBM+92, Rya92, KG94]. The set of pairings computed by the anonymous crew scheduling phase has to be the optimum one, therefore this phase is also referred to as *crew pairing optimization*.

In this paper, the crew pairing optimization problem is considered and a specific methodology is described for tackling it in a Constraint Logic Programming (CLP) [vH89, FHK+92] environment. An appropriate representation of the problem is proposed together with the required mathematical formulation of the involved constraints. The whole approach has been realized using the ECL^{i}PS^e language [ECL95], an instance of the CLP class of languages which provides, among other facilities, constraint satisfaction techniques over finite domains.

Various approaches taken from the Operations Research (OR) area have been followed so far for the solution of the crew pairing optimization problem [LMO88, AGPT91]. More precisely, the final stage of this problem is an instance of the *set partitioning* (*covering*[1]) *problem* which has been extensively studied and tackled via OR methods in the past [Mar74, FK90, BJ92]. In addition, an application of OR methods in a CLP environment

[1]Set covering is used when *deadheading* (or *positioning*) is allowed, i.e. transportation of crew members as passengers to another place, in order to commence or continue work.

for tackling the same problem is reported in [GC95]. However, although these methods might be sometimes quite efficient for specific cases of input data, they lack a general and elegant way of describing what is needed to be computed and which constraints are involved. Moreover, OR methods are not flexible enough to cater for possible changes of the problem's input data, thus they cannot be easily adapted to various needs.

On the other side, what is achieved by a pure CLP approach is a declarative way of describing a constraint satisfaction problem which is combined, by taking into consideration the procedural point of view, with an a priori pruning of the search space through a constraint propagation mechanism. At each step, the constraints are exploited as much as possible, and then a choice is made which triggers more propagation. Choices are made non-deterministically. In this way, the resolution mechanism of a logic programming language is combined with constraint satisfaction techniques resulting in a very powerful and elegant scheme of computation.

The purpose of this paper is to exhibit the appropriateness of pure CLP for mastering the combinatorial nature of the crew pairing optimization problem. Firstly, this problem is introduced and CLP as well as ECL^{i}PS^e are briefly described. Then, the approach employed to work on the crew pairing optimization is presented and the way this is handled in ECL^{i}PS^e is highlighted. Next, the resulting system is outlined and, finally, the adopted approach is discussed.

2 Crew Pairing Optimization

The usual approach to the crew pairing optimization is to divide it into two subproblems. First, all legal pairings are built (pairing construction) and then a subset of pairings that covers all legs exactly once is selected (set partitioning problem).

The input to the pairing construction phase is the set of legs the airline company has to carry out. All legs are combined in sequences that respect the imposed constraints and, thus, they can be performed by a single crew. A pairing can last several days provided that there is the necessary rest time for the crew that carries it out. A basic constraint is that a pairing should start and end at the same airport. This airport is called *base airport* and it is at the place where the crew members employed by the airline company live. Other regulations and constraints that must be satisfied by each crew pairing are the following [CM92]:

- **Temporal constraints**

  There must be enough time between consecutive legs, so that every leg starts after the previous one has finished.

- **Local constraints**

  The arrival airport of a leg must be the same with the departure airport of the succeeding leg.

- **Flight Duty Time (FDT)**

  FDT is the time during which a crew member operates on an aircraft as a member of its crew. This parameter must not exceed a certain number of hours.

- **Number of legs per working day and number of legs per pairing**

  Upper bounds exist for both these factors.

The phase responsible for the set partitioning problem accepts the created pairings which have been also assigned a cost. The cost is usually related with the crew utilization. The system selects those pairings that cover all flight legs with the minimum cost.

It should be noted that each leg must belong to only one pairing and each pairing must contain every leg at most once. Pairings last several days, although legs may be performed once every day. If, for example, daily legs are considered, which is true in the context of this paper, then the question is how the set of produced pairings can be combined in such a way that each leg is carried out once every day during a long period of time, such as a month. The answer is simple: each pairing must start every day. This idea is better illustrated in the example given in Table 1. Each of the legs in this example is performed every day. Given a set of regulations and constraints, a possible solution is shown in Table 2. The results proposed there contain two crew pairings spanning over a period of five days. Since the legs occur every day, $7 \times 5 = 35$ working days per week are needed to cover the given set of legs.

3 CLP and ECL^{i}PS^e

CLP is a novel programming framework that enhances logic programming with constraint satisfaction techniques. While preserving the declarative nature of logic programming, it exhibits surprising efficiency for solving a certain class of combinatorial problems. While the straightforward solution of such problems might be done through a *generate-and-test* approach, CLP introduces a new method of computation, the *constrain-and-generate* one. In case of large scale combinatorial problems, the former method cannot lead to results within finite time.

CLP is often referred to as $CLP(X)$ where X stands for the set over which the problem constraints are stated. There exist languages where X is the set of real numbers $\Re$ [JMSY92], the set of rational numbers, a set of boolean values etc. A very interesting case

Flight Number	*Departure Airport*	*Arrival Airport*	*Departure Time*	*Leg Duration*	*Aircraft Id*
242	ath	mic	08:15	60	1
245	mic	ath	10:15	60	1
214	ath	her	12:15	45	1
217	her	ath	13:45	45	1
160	ath	kav	15:35	90	1
169	kav	ath	17:55	85	1
058	ath	rod	07:25	50	2
066	rod	ath	10:00	45	2
120	ath	tri	12:05	45	2
174	tri	ath	13:40	45	2

Table 1: Example of a flight plan for two specific aircrafts. The three letter codes stand for the departure and arrival airports of each leg. The flight number, the departure time, the leg duration and the aircraft id also appear.

Pairing	*Day*	*Flights*		
1	1		160	169
		ath	15:35-17:05 kav	17:55-19:20 ath
	2		214	217
		ath	12:15-13:00 her	13:45-14:30 ath
	3		120	174
		ath	12:05-12:50 tri	13:40-14:25 ath
	4		242	245
		ath	08:15-09:15 mic	10:15-11:15 ath
2	1		058	066
		ath	07:25-08:15 rod	10:00-10:45 ath

Table 2: A possible solution for crew pairings. The flight number appears above the respective leg. The solution consists of a set of two pairings. The first pairing endures four days and the second one.

is the one where a problem is described via constraints which involve variables that range over finite domains. Then, the corresponding instance of $CLP(X)$ is called $CLP(FD)$. The first representative of the $CLP(FD)$ scheme of computation is the CHIP language [DvHS+88]. CHIP was developed at the European Computer-Industry Research Centre (ECRC). The constraint facilities of CHIP have now been integrated with ECRC's Sepia and Megalog systems, resulting into the ECL^{i}PS^e language.

ECL^{i}PS^e is a Prolog system whose aim is to serve as a platform for integrating various logic programming extensions. Many interesting extensions of ECL^{i}PS^e are based on a special data type, which is called *metaterm*. Various constraint related libraries are supported, which have been built on the metaterm facility, namely the finite domains library, the generalized propagation library etc. The first is the one which brings to ECL^{i}PS^e the constraints functionality of CHIP for finite domains.

The finite domains library of ECL^{i}PS^e implements constraints that involve integer as well as atomic data. The concept of *domain variables* is used which range over finite domains. These are defined through the built-in predicate `::/2`. For example, the sequence of goals `X :: 1..5, Y :: [red, green, blue], Z :: [3, 7..9, 12]` define the domain variables `X`, `Y` and `Z` ranging over the domains {`1`, `2`, `3`, `4`, `5`}, {`red`, `green`, `blue`} and {`3`, `7`, `8`, `9`, `12`} respectively. Constraints are either *arithmetic* or *symbolic*. An arithmetic constraint is an equality (`#=/2`) or inequality (`##/2`, `#</2`, `#<=/2`, `#>/2`, `#>=/2`) relation between two *linear terms*, where a linear term is composed of domain variables and integers in a way respecting its linearity. For example, `2*X-Z ## X+3` is an arithmetic constraint stating that the involved linear terms should be different, for some values of the variables `X` and `Z`. An example of a symbolic constraint is `element(I, [red, green, blue], W)` which states that `W` (a domain variable) should be the `I`-th (also a domain variable) element of the list `[red, green, blue]`. Enumeration (`indomain/1`) as well as optimization (`min_max/2`, `min_max/5`) predicates are also provided by ECL^{i}PS^e.

The procedure for solving a constraint satisfaction problem using the finite domains library of ECL^{i}PS^e is to define a set of domain variables, together with their domains, that model the problem concepts. Then, the constraints that define the relations among these variables have to be stated. Actually, these constraints define the subset of the search space that contains the solutions of the problem. The last step is to trigger an enumeration procedure, which cooperates with the internal constraint propagation mechanism and with the backtracking mechanism of the Prolog engine, leading finally to the required solutions, if they exist. In case of an optimality problem, ECL^{i}PS^e provides a branch-and-bound method that computes the optimum solution of the enumeration procedure, with respect to a given cost function. An example that demonstrates most of the finite domains functionality of ECL^{i}PS^e is the following:

```
?- X :: 1..5, Y :: 3..8, X+Y #> 9, 5*X-Y #<= 12,
   min_max((indomain(X), indomain(Y)), Y-X).
Found a solution with cost 6
Found a solution with cost 4

Y = 7
X = 3
yes.
```

In this example, `X` and `Y` are defined as domain variables with values from `1` to `5` and from `3` to `8` respectively, their sum is stated to be greater than `9`, then the linear term `5*X-Y` is stated to be less than or equal to `12` and, finally, the branch-and-bound mechanism of ECL^{i}PS^e is activated, through the `min_max/2` goal, which is asked to compute the appropriate values of `X` and `Y`, i.e. ones that satisfy `indomain(X)` and `indomain(Y)`, which minimize the linear term `Y-X`.

4 Crew Pairing Optimization under ECL^{i}PS^e

Besides the concept of pairings, it is useful to define *jobs* as well. A job (or *duty period*) practically corresponds to the flights performed in a day. More precisely, the continuous period of time during which a crew is on duty is a job. Thus, a crew pairing is a sequence of jobs. Consecutive legs within a job must satisfy a number of regulations and constraints. Usually a job doesn't last more than 24 hours, although this is not forbidden. Moreover, the rest time between two consecutive legs within the same job doesn't exceed a certain amount of hours.

Before the construction of pairings, all the legal, possible and "good" jobs are found. Good are those jobs that would lead to the construction of pairings with low cost. The input of this process is the set of all flight legs. The jobs created in this phase are used to form legal pairings. The pairing construction phase is akin to the job construction phase as both entities are subject to similar constraints. Because the number of all legal pairings is large, heuristic techniques that produce the good pairings are implemented, that is, the pairings that would lead to a solution with small cost. The last phase of the crew pairing optimization is a case of the set partitioning problem. From the set of all pairings produced in the previous phase, the subset that covers all flight legs at minimum cost has to be chosen.

In the context of this paper, daily legs are considered, that is each leg is performed once in a day, although a pairing lasts several days. The airline company has only one base airport. Therefore, all pairings start and end at this base airport.

4.1 Job Construction

A leg is represented by the following ECL^{i}PS^e fact:

$$leg(Key, Departure\_Airport, Destination\_Airport,$$
$$Departure\_Time, Duration, Aircraft\_Id).$$

where Key is a number that identifies uniquely a leg and has the property: if one leg is identified by Key_1 and another leg by Key_2 and $Key_1 < Key_2$, then $Departure\_Time_1 \leq Departure\_Time_2$. $Aircraft\_Id$ is the identifying number of the aircraft which carries out the flight.

4.1.1 Job Representation Using Domain Variables

A job is represented by a list of lists with 0/1 domain variables, each of which corresponds to a flight leg. A variable has the value 1 when the corresponding leg is part of the job and it has the value 0 otherwise.

The maximum duration of a job is considered to be 24 hours. If a list corresponds to a diary day, then a job can be represented by two lists, since a 24-hour job may span in two diary days at most. If the following group of legs is available

```
leg(1, ath, lgw, deptime(7, 45), 60, 1).
leg(2, ath, lhr, deptime(7, 45), 90, 1).
leg(3, lgw, ath, deptime(12, 45), 60, 1).
leg(4, lhr, ath, deptime(18, 0), 90, 1).
```

then a job would be represented by the following list:

$$[[x_{11}, x_{12}, x_{13}, x_{14}], [x_{21}, x_{22}, x_{23}, x_{24}]]$$

Generally, the following list represents a 24-hour job:

$$[[x_{11}, x_{12}, \ldots, x_{1n}], [x_{21}, x_{22}, \ldots, x_{2n}]]$$

where the first list (x_{1i} variables) stands for the first day and the second list (x_{2i} variables) stands for the second day. The index i refers to the leg with key i.

The above representation apart from being suitable for expressing all constraints has one more advantage: it is general. Practically no change would be required to the program code if it was decided that a job might last more than 24 hours, but certainly less than some other limit, e.g. 72 hours, and, thus, more sublists would be needed.

4.1.2 Job Constraints

It is quite difficult to describe with linear equations and inequations a real life constraint satisfaction problem. The success of the description depends highly on the representation of the problem as well as on the way the constraints are set. Sometimes, instead of thinking what should be true, it is easier trying to express what is forbidden.

For two 0/1 variables y and z, the operation $y\ NAND\ z$ may be defined, which is false only when both y and z are true. This operation expresses the fact that two events cannot occur simultaneously. The boolean operation $NAND$ corresponds to the arithmetic inequality $y + z \leq 1$.

The above remark is quite useful in creating the constraint expressions. Suppose that two legs with corresponding domain variables y and z cannot occur in the same job. The arithmetic expression for that constraint is then $y + z \leq 1$.

In job construction, three kinds of constraints can be distinguished: the constraints that depend on the nature of the problem and are obligatory, the constraints that are dictated by the representation used and are useful for the efficient and quick solution of the linear system of constraints and, finally, the heuristic constraints. These constraints are not part of the airline company's regulations; they are invented and are used to reduce the number of jobs produced.

Suppose now that $A = [[x_{11}, x_{12}, \ldots, x_{1n}], [x_{21}, x_{22}, \ldots, x_{2n}]]$. The way constraint expressions are built is explained in detail next.

I. **Constraints imposed by the problem**

- **Maximum number of legs per job**

  Each job must contain a limited number of legs. This constraint expression can be constructed easily if all the domain variables of a list are added and then their sum is bound by the maximum number of legs, called Max.

$$\sum_{i=1}^{n}(x_{1i} + x_{2i}) \leq Max$$

  where n is the number of variables contained in each day sublist.

- **Maximum flight duty time per job**

  This constraint expression is similar to the above. Each sublist member is multiplied by the duration of the corresponding leg, called T_i. The sum is then bound by $MaxFDT$, the maximum flight duty time allowed:

$$\sum_{i=1}^{n}(x_{1i} + x_{2i}) \cdot T_i \leq MaxFDT$$

- **Local constraints**

  The arrival airport of each leg in a job must be the same with the departure airport of the following leg in the same job. This is a complicated case of the $NAND$ constraints mentioned above.

  Suppose that two domain variables y and z are adjacent in list A and represent legs for which the arrival airport of leg y is different from the departure airport of z. Then, these variables should not have simultaneously the value 1. The constraint $y + z \leq 1$ should be set. It has to be noted here that in this paper deadheading is not allowed. If now y and z are not adjacent in A and w is the only element between them, then y and z cannot have the value 1 at the same time if $w = 0$ and, thus, $y + z \leq 1 + w$ should be true. In general, if y and z are separated by k elements (for example $A = [[\ldots, y, w_1, \ldots], [\ldots, w_k, z, \ldots]]$), then the constraint expression needed to be set is:

$$y + z \leq 1 + \sum_{i=1}^{k} w_i$$

  for all possible pairs of y and z in list A, such that the arrival airport of y is different from the departure airport of z.

- **Time constraints**

  - There should be sufficient time between legs for briefing and debriefing. So, if two legs do not satisfy this condition, they should not belong to the same job. If y and z are the domain variables that correspond to such legs, then the following should hold:

$$y + z \leq 1$$

  - Furthermore, adjacent legs in a job should be relatively close one another: the time period between the arrival time of the first and the departure time of the second should be less than 10 hours. This constraint is expressed in the same way as the local constraints. That is, for two domain variables y and z that cannot have simultaneously the value 1 and are separated by k domain variables the following should be true:

$$y + z \leq 1 + \sum_{i=1}^{k} w_i$$

    This constraint has to be stated for all possible pairs of y and z in list A, such that the distance between the arrival time of y and the departure time of z is greater than Max, where Max is the maximum rest time between

two legs in the job.

- **Job duration**

  The duration of a job must be at most 24 hours. If the difference between the departure time of a leg and the arrival time of the following leg is more than 24 hours, then the two legs should not belong to the same job. Thus, if y is the domain variable for the first leg and z is the domain variable for the following leg, then:

  $$y + z \leq 1$$

  This constraint has to be stated for all possible pairs of y and z in list A, such that the departure time of y and the arrival time of z have a distance greater than 24 hours.

II. **Constraints imposed by the representation**

- **Beginning of a job**

  Each job must start at the first day, because otherwise the representation with two lists would be meaningless. Therefore, the following constraint must be satisfied:

  $$\sum_{i=1}^{n} x_{1i} \geq 1$$

  where x_{1i} are the domain variables of the first sublist and n is their total number.

III. **Heuristic constraints**

- **Minimum number of legs per job**

  It is useless to construct jobs with very few legs, for example 0 or 1. Therefore, the number of legs each job comprises must be greater than Min. The expression for this constraint follows:

  $$\sum_{i=1}^{n} (x_{1i} + x_{2i}) \geq Min$$

  where n is the number of variables contained in the sublists of list A.

- **Departure airport of a job**

  A job should start from the base of the airline company. This is not obligatory, but it reduces the number of jobs, since it keeps those which lead to the construction of good pairings (pairings with small cost). If the departure airport of a leg is other than the base airport, then a job should not start with that

leg. Let y be the domain variable for the specific leg and suppose k variables w_i precede y in list A ($A = [[w_1, \ldots, w_k, y, \ldots], [\ldots]]$). Then it should be true that:

$$y < 1 + \sum_{i=1}^{k} w_i$$

The above constraint should be set for every y in the first sublist of A whose departure airport is different from the base airport.

- **Arrival airport of a job**

  For the same reasons a job should end at the base airport. If the number of times that the base airport appears in the job as the departure airport is equal to the number of times it appears in the job as the arrival airport, then this requirement is satisfied. That is:

$$\sum_{dep_i = base} (x_{1i} + x_{2i}) = \sum_{arr_i = base} (x_{1i} + x_{2i})$$

  where dep_i and arr_i are the departure and arrival airports of the leg i respectively.

- **Flight legs in the second diary day**

  Only a certain number of flight legs (strictly less than n) can be realized in the second diary day. Therefore, after a specific leg, no other leg is comprised in a job, that is

$$\sum_{i=k+1}^{n} x_{2i} = 0$$

  where k is the key of the last leg permitted in a job, x_{2i} the variables in the second sublist of list A and n their total number.

4.2 Pairing Construction

The jobs created in the previous stage are used for the construction of the pairings. It should be mentioned that the representation of the jobs is similar to the representation of the legs described in the previous paragraph.

A job is represented by the following ECL^{i}PS^e fact:

$$job(Key, Departure\_Airport, Destination\_Airport, Departure\_Time,$$
$$Flight\_Duration, Job\_Duration, Flights, Number\_Of\_Flights).$$

where Key is a number that identifies uniquely a job and it has the following property: if one job is identified by Key_1 and another job is identified by Key_2, then $Departure\_Time_1 \leq Departure\_Time_2$. $Departure\_Airport$ and $Destination\_Airport$ are the departure airport

of the first leg and the destination airport of the last leg respectively and $Departure\_Time$ is the departure time of the first leg contained in the job. $Flight\_Duration$ is the sum of the duration of flights, while $Job\_Duration$ is the difference between the arrival time of the last leg contained in the job and the departure time of the first leg. Furthermore, $Flights$ is a list containing the leg keys that correspond to the legs of the job and $Number\_Of\_Flights$ is the length of this list.

4.2.1 Pairing Representation Using Domain Variables

A pairing is represented by a list of lists (sublists), each of which represents a day. Each sublist contains 0/1 domain variables, each of which corresponds to a job. A variable has the value 1, when the corresponding job takes place the specific day of the specific pairing and it has the value 0 otherwise. The purpose is to build such lists of domain variables that will be subject to the constraints. Suppose that the following set of jobs is created from the previous stage:

```
job(1, ath, ath, deptime(7, 45), 120, 360, [1, 3], 2).
job(2, ath, ath, deptime(7, 45), 180, 705, [2, 4], 2).
```

If a pairing is allowed to last at most three days, then the following list that contains three sublists represents a pairing:

$$[[x_{11}, x_{12}], [x_{21}, x_{22}], [x_{31}, x_{32}]]$$

The variables x_{1i} correspond to the first day, the variables x_{2i} correspond to the second day and so on. The index i corresponds to the job with key i. Generally, the following list represents a pairing that lasts k days (k-day pairing)

$$[[x_{11}, x_{12}, \ldots, x_{1n}], [x_{21}, x_{22}, \ldots, x_{2n}], \ldots, [x_{k1}, x_{k2}, \ldots, x_{kn}]]$$

where n is the number of all jobs.

4.2.2 Pairing Constraints

The pairing constraints are similar to the job constraints and their construction is made in the same way. Suppose that

$$B = [[x_{11}, x_{12}, \ldots, x_{1n}], [x_{21}, x_{22}, \ldots, x_{2n}], \ldots, [x_{k1}, x_{k2}, \ldots, x_{kn}]]$$

is the representation of a k-day pairing when n jobs are available. x_{ij} stands for the domain variable that corresponds to the job with key j on the i-th day.

I. **Constraints imposed by the problem**

- **Maximum number of legs per pairing**

  If the j-th job comprises N_j legs, then

  $$\sum_{i=1}^{k}\sum_{j=1}^{n} N_j \cdot x_{ij} \leq Max$$

  where Max is the maximum number of legs per pairing.

- **Maximum number of legs per pairing during three, four and five days**

  This constraint ensures that the flights are distributed uniformly to the days of the pairing. The inequation is constructed in the same way as in the above constraint, for every three, four and five consecutive days in the pairing, setting the appropriate upper limits for the corresponding sums.

- **Rest time**

  There should be enough time between two jobs in a pairing so that the crew members can rest. In order to ensure this requirement, the following is done: for every two domain variables y and z that represent jobs in the pairing list, it is checked whether these jobs can belong to the same pairing, that is if there is sufficient time between the arrival time of the first job and the departure time of the second job. If not, then

  $$y + z \leq 1$$

  Since a job lasts 24 hours maximum, not all possible couples of jobs need to be checked. It is necessary to check only these couples of domain variables that belong to adjacent days.

- **Local constraints**

  These constraints are similar to the local constraints of the jobs. They ensure that the departure airport and the arrival airport of a pairing are the same with the base airport of the airline company. It is reminded that in this paper, it is assumed that an airline company has only one base airport. Because every job starts and ends at the base airport (heuristic constraint), always the arrival airport of a job is the same with the departure airport of the following job. If an airline company has more than one base airport, a specific constraint which ensures the identity of the arrival and departure airports of adjacent jobs must be defined.

II. **Constraints imposed by the representation**

- **Beginning of a pairing**

  A pairing should start at the first day, because if not, then the first sublist would be empty. So:

$$\sum_{j=1}^{n} x_{1j} \geq 1$$

- **Unique legs in a pairing**

  A leg may belong to more than one job. Therefore a pairing may contain the same leg more than once, which is forbidden. In order to solve this problem, the pairing list is searched and all the jobs that contain the same leg are found. Suppose that these jobs correspond to the domain variables y, z and w. Their sum must be less than or equal to 1, that is: $y + z + w \leq 1$. The above procedure is repeated for every leg.

4.3 Set Partitioning Problem

In the previous section, the way pairings are created was described. From these pairings a set must be selected that covers all legs and at the same time minimizes a cost function. This problem is called set partitioning. Suppose that m legs must be carried out and l pairings have been constructed. If the problem is formulated mathematically, the function that must be minimized is:

$$\sum_{j=1}^{l} c_j \cdot x_j$$

when:

$$A \cdot x = e, x_j = 0 \text{ or } 1, \forall j : 1 \leq j \leq l$$

where c_j is the cost assigned to x_j, A is an $m \times l$ matrix containing 0 and 1, $e = (1, 1, \ldots, 1)$, i.e. e is an m-dimensional vector with all its elements equal to 1.

4.3.1 Cost Function

The cost function is very important in solving the set partitioning problem. As it is not specific, its selection depends on the airline company's regulations and ideas about cost effective crew schedules. The cost function used in this paper is the total pairing duration, that is the sum of the job durations. So, trying to minimize the cost function means choosing pairings that have the smallest duration. If there are n jobs and $Job\_Duration_i$ is the duration of the job with key i then the cost C of a pairing is equal to:

$$C = \sum_{i=1}^{n} Job\_Duration_i$$

4.3.2 Set Partitioning Constraints

Suppose that $A = [P_1, P_2, \ldots, P_l]$ is a list containing all the pairings created during the pairing construction stage, where P_i is the i-th pairing. Moreover, consider a list $X = [X_1, X_2, \ldots, X_l]$ where X_j is a 0/1 domain variable, which corresponds to pairing P_j. If the domain variable X_j has the value 1, then the pairing P_j is contained in the solution, otherwise it isn't.

- **Basic constraint**

  The basic constraint imposed by the set partitioning problem is that each leg must be contained exactly once in the final solution. If there are m legs to be covered, then for each leg with key i the following constraint is set:

  $$\sum_{1 \leq j \leq l, i \in P_j} X_j = 1$$

  That is, the sum of the domain variables, which correspond to pairings that contain the leg with key i must be equal to 1.

- **Maximum number of pairings**

  The final solution must contain a maximum number of pairings. This constraint, which is actually a heuristic one, is expressed by the following equation:

  $$\sum_{i=1}^{l} X_i \leq Max$$

  where Max is the maximum number of pairings permitted in the final solution.

5 A Prototype System

A prototype implementation of the system has been carried out in ECL^{i}PS^e and it is based on the constraint equations and inequations described in the previous paragraphs. The user can communicate with the system through a graphical user interface, which provides various facilities. This interface is based on the PCE extension of ECL^{i}PS^e. The main window consists of some buttons responsible for selecting the system operations and of two windows; the message window and the graphics window. In the first window, various messages appear that inform the user about operations selected and performed. In the second window, the graphical representation of the solution appears after the execution of the program.

The input of the system is a file containing information about the flights that have to be scheduled. Each line of this file represents a leg, and contains all information needed for

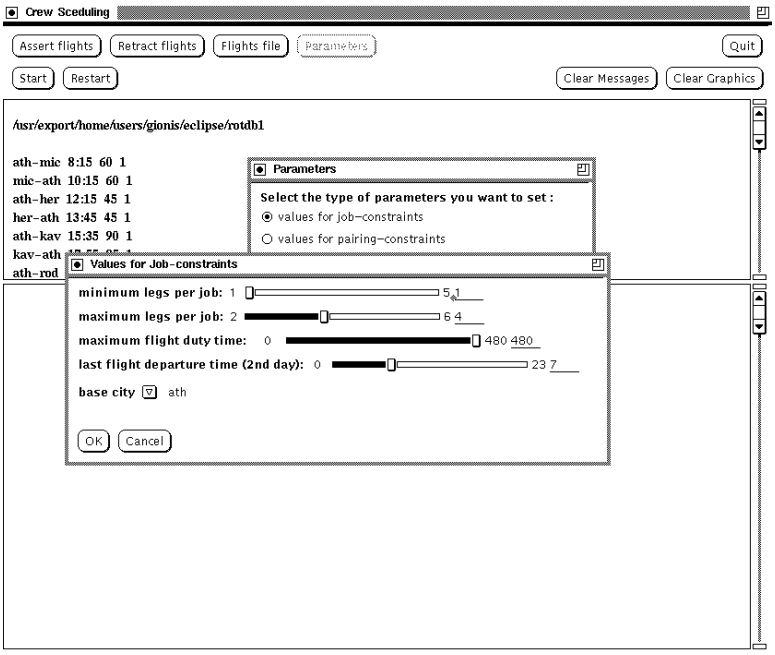

Figure 1: Sample of program execution: A flights file has been asserted. Dialog boxes for the customization of constraints also appear.

this leg in a quite obvious format. Once the user has specified the flights file, the system transforms the leg information into ECL^{i}PS^e facts (e.g. the ones in paragraph 4.1.1). If the user inserts another flights file, besides the initial one, the new legs are appended. Therefore, the user has the opportunity to combine as many flights files as he/she wishes in one program execution. These files can be created and modified by any system editor. Furthermore, the system itself provides a friendly way for creating such flights files.

Apart from the leg information, the user can assign values to the constraint parameters involved in most constraint equations. A dialog box appears for each type of constraints (job constraints, pairing constraints and set partitioning constraints) as it is shown in Figure 1. The parameters have a default value with which the constraints will be set, if no change is made. The user can customize the constraints and his/her choice may depend on the constraints imposed by the airline company or on the desirable execution time, since the values of the constraint parameters affect the number of jobs or pairings produced. For example, one can define the maximum number of legs per day, the flight duty time or the base airport.

When the execution of the program module has finished, the solution appears in the message window. A graphical representation of the solution is also shown in the graphics window (Figure 2). Other facilities available to the user are the retraction of legs, the restarting of the program and the windows clearing.

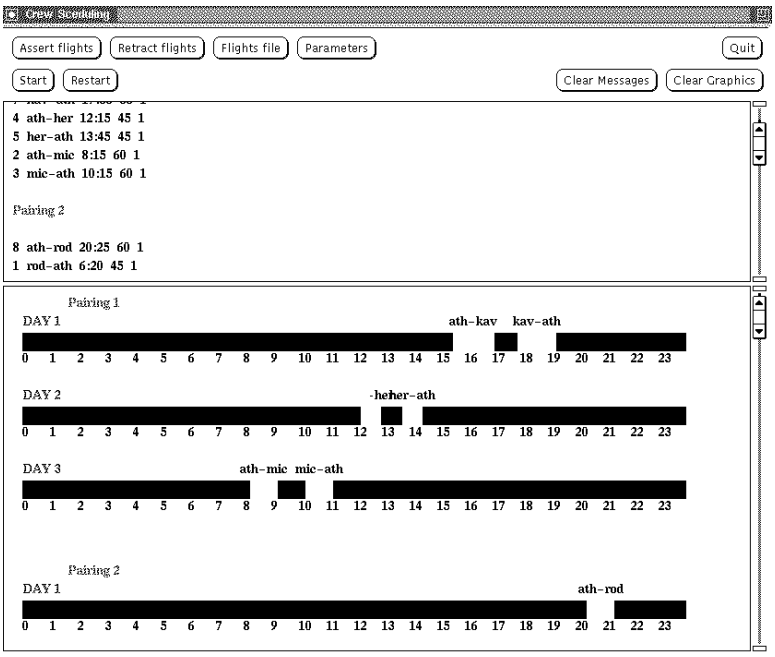

Figure 2: Sample of program execution: An optimal solution has been found and appears in both the message and the graphics windows.

6 Conclusions

The crew pairing optimization is a crucial operation for an airline company. As the cost of the crews is very high, a good solution of the problem that maximizes the crew utilization or other factors is desirable.

In this paper, an approach for the crew pairing optimization problem, based on pure CLP, is presented, which seems to suit the particularities of crew pairing optimization quite well. CLP has helped in expressing clearly and comprehensively the constraints of the problem. A simple working system has been also built in ECL^{i}PS^e. This system is characterized by good prototyping, which is combined with other positive aspects, like flexibility and maintainability. The code is simple, can be read and understood easily and the modules responsible for the setting of various constraints are independent. Therefore, it can be expanded easily, as only the new constraints have to be programmed, and no other changes to the rest of the system are required. The problem is that while low cost results are produced for small fleets, the developed system becomes inefficient when tested with more than a certain number of daily flights. This generally depends on the characteristics of these flights and the specific constraint parameters. However, we aspire that the basic contribution of this work is to provide a mathematically precise and easy to implement formulation for a crucial problem, as the crew pairing optimization. Our experience from various projects that made use of constraint programming has shown that usually modeling problem constraints in a straightforward way through "built-in" language constraints might

be quite inefficient in real-world problems. The reason is that problem constraints are too specific for the general purpose propagation techniques that the constraint languages use. However, such problem constraints provide the necessary input for developing built-in specific constraints and, thus, much better results can be obtained without resorting to specialized methods such as the ones of OR.

References

[AFST69] J. P. Arabeyre, J. Fearnley, F. C. Steiger, and W. Teather. The airline crew scheduling problem: A survey. *Transportation Science*, 3:140–168, 1969.

[AGPT91] R. Anbil, E. Gelman, B. Patty, and R. Tanga. Recent advances in crew-pairing optimization at American Airlines. *Interfaces*, 21(1):62–74, 1991.

[BBM+92] L. Bianco, M. Bielli, A. Mingozzi, S. Ricciardelli, and M. Spadoni. A heuristic procedure for the crew rostering problem. *European Journal of Operational Research*, 58:272–283, 1992.

[BJ92] J. E. Beasley and K. Jørnsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58:293–300, 1992.

[CM92] I. F. Croall and J. P. Mason, editors. *Industrial Applications of Neural Networks — Project ANNIE Handbook*. Springer-Verlag, 1992.

[DvHS+88] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 693–702, 1988.

[ECL95] *ECL^{i}PS^e 3.5: User Manual*, December 1995.

[FHK+92] T. Frühwirth, A. Herold, V. Küchenhoff, T. Le Provost, P. Lim, E. Monfroy, and M. Wallace. Constraint logic programming — An informal introduction. In G. Comyn, N. E. Fuchs, and M. J. Ratcliffe, editors, *Logic Programming in Action*, pages 3–35. Springer-Verlag, 1992.

[FK90] M. L. Fisher and P. Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6):674–688, 1990.

[GC95] N. Guerinik and M. Van Caneghem. Solving crew scheduling problems by constraint programming. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 481–498, 1995.

[Ger89] I. Gershkoff. Optimizing flight crew schedules. *Interfaces*, 19(4):29–43, 1989.

[HP93] K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

[JMSY92] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP($\Re$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[KG94] M. Kress and B. Golany. Optimizing the assignment of aircrews to aircraft in an airlift operation. *European Journal of Operational Research*, 77:475–485, 1994.

[LMO88] S. Lavoie, M. Minoux, and E. Odier. A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35:45–58, 1988.

[Mar74] R. E. Marsten. An algorithm for large set partitioning problems. *Management Science*, 20(5):774–787, 1974.

[MMK79] R. E. Marsten, M. R. Muller, and C. L. Killion. Crew planning at Flying Tiger: A successful application of integer programming. *Management Science*, 25(12):1175–1183, 1979.

[MS81] R. E. Marsten and F. Shepardson. Exact solution of crew scheduling problems using the set partitioning model: Recent successful applications. *Networks*, 11:165–177, 1981.

[Rub73] J. Rubin. A technique for the solution of massive set covering problems, with application to airline crew scheduling. *Transportation Science*, 7:34–48, 1973.

[Rya92] D. M. Ryan. The solution of massive generalized set partitioning problems in aircrew rostering. *Journal of the Operational Research Society*, 43(5):459–467, 1992.

[Spi61] M. Spitzer. Solution to the crew scheduling problem. In *Proceedings of the First AGIFORS Symposium*, 1961.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* The MIT Press, 1989.