Crew Pairing Optimization with Genetic Algorithms

Harry Kornilakis and Panagiotis Stamatopoulos

Department of Informatics and Telecommunications
University of Athens
Panepistimiopolis, 157 84 Athens, Greece
{harryk,takis}@di.uoa.gr

Abstract. We present an algorithm for the crew pairing problem, an optimization problem that is part of the airline crew scheduling procedure. A pairing is a round trip starting and ending at the home base, which is susceptible to constraints that arise due to laws and regulations. The purpose of the crew pairing problem is to generate a set of pairings with minimal cost, covering all flight legs that the company has to carry out during a predefined time period. The proposed solution is a two-phase procedure. For the first phase, the pairing generation, a depth first search approach is employed. The second phase deals with the selection of a subset of the generated pairings with near optimal cost. This problem, which is modelled by a set covering formulation, is solved with a genetic algorithm. The presented method was tested on actual flight data of Olympic Airways.

1 Introduction

Given a timetable containing all the flight legs that an airline company must carry out, the airline *crew scheduling problem* [8] consists of assigning individual crew members to flight legs, so that the assignment is legal and crew costs are minimized. Crew scheduling is separated into two independent subproblems. The *crew pairing problem* (CPP), which will be the focus of this paper, is the process of finding a set of round trips (*pairings*) starting and ending at the home base, covering all flight legs that the company has to carry out with a minimal cost. The *crew assignment problem* is the assignment of individual crew members to pairings and it will not be considered here. What we propose in this paper is to employ a genetic algorithm based approach to deal with the CPP.

The main difficulty of the CPP is the huge size of the search space, which grows exponentially with the number of flight legs. This means that even for average sized problems containing a few hundred flight legs, an intractably large number of legal solutions may exist. Finding the optimal solution among them may not be possible to do efficiently and often methods that produce only a good approximation of the optimal solution have to be used. Another consideration is the nature of the constraints, which are usually based on aviation laws and regulations and, in general, are non-linear. Such constraints tend to be rather difficult to model in an algorithm that automatically solves the CPP.

Crew scheduling is an important problem for airline companies. In fact, crew costs is the second greatest operational expense for airlines, exceeded only by the cost of fuel consumption [2]. As an example, in [1], it is reported that American Airlines spent 1.3 billion dollars on crew costs in 1991. Therefore, a low cost solution to the CPP can save airlines millions of dollars per year. Almost every major airline is using a system that automates crew scheduling. However, computational tests, conducted with actual data, led to the conclusion that many of the solutions provided by these systems needed significant improvement [6].

Much work has been done in the past for tackling the CPP. Traditional approaches are based on Operations Research techniques [19], some of them exploiting parallelism as well [10]. Various methods based on genetic algorithms [7,15,16] or neural networks [14] have been also proposed. Finally, Constraint Programming has been used as a platform for solving the CPP [17], in some cases combined with parallel processing [11].

The paper is organized as follows. In section 2, we deal with the CPP in greater depth and in section 3, we present our proposed method for the CPP, focusing on the optimization procedure and the genetic algorithm that has been implemented for it. Section 4 presents the experimental results on actual flight data of Olympic Airways and finally, our conclusions appear in section 5.

2 The Crew Pairing Problem

The objective of the airline crew pairing problem is to minimize costs associated with assigning crews to flight legs [6]. Every crew has a home base and a pairing is a round trip, consisting of a sequence of flight legs, which starts and ends at the crew's home base. Each pairing is composed of a number of legal workdays, called *duties*, which are separated by rest periods. Airline crew pairing seeks to find a subset of legal pairings such that each flight leg is covered, preferably by only one crew, and so that the total cost of these pairings is minimal.

In order to be legal, the construction of a pairing must take into account a number of constraints. Some of these constraints, like the temporal and spatial constraints, follow directly from the definition of the problem, while others are results of laws and regulations. Specifically, the following types of constraints can be identified:

- Temporal constraints: The departure of a flight leg should obviously take place after the arrival of the previous leg of the pairing. Additionally, a certain amount of time, called transition time, must pass between the arrival of a leg and the departure of the next.
- Spatial constraints: For every two consecutive flight legs in a pairing, the second must depart from the airport that the first arrives at. Also, the first leg of a legal pairing must depart from the home base of the crew and the last leg of each pairing must end at the home base.

- Fleet constraints: Cockpit crew is usually designated to operate only one type of aircraft. Therefore, all flights in a pairing generated for cockpit crew must be performed by the same type of aircraft. On the other hand, cabin crew may be placed in any type of aircraft. Consequently, pairings that are legal for cabin crew may be illegal for cockpit crew, making the cockpit crew problem easier to solve.
- Constraints due to laws and regulations: In order to be legal, pairings must follow government laws and collective agreements. These usually define the maximum duration of a duty, the maximum flight time allowed over a period of time, the minimum length of the rest period between two duties etc. These values may vary depending on the length or the destination of a flight leg and usually are quite different for cockpit and cabin crews.

Another consideration is that it is possible that, besides the crew that is on duty, other crew members are on a plane, travelling as passengers, in order to move to a specific airport and continue a round trip. This process is known as *deadheading* and is an additional cost to the airline, since deadheaded crews get normally paid, even though they are not working, and occupy seats that would otherwise be given to passengers. Obviously, good solutions should avoid deadheaded flights, however it is not always the case that solutions with no deadheading exist. It is also possible that a solution with a few deadheaded flights is better than one with no deadheading. In any case, allowing deadheading results to a larger space of feasible solutions and finding the optimal solution becomes harder.

The cost of a solution is another factor that is critical to the problem. Usually the cost of each pairing is equal to the wages of the crew. This, again, depends on the policy of the airline company, but, in general, it is a function of the flying time, the duty time or the number of duties of a pairing. The total cost of a solution is usually the sum of the costs of all pairings in it, but an additional penalty for deadheaded flights may be introduced.

Looking at the nature of the constraints and the cost function, it becomes obvious that modelling them is not a trivial task. In fact, in most cases, it is impossible to formulate the legal pairings or the cost function as linear equations and therefore use a linear programming framework to solve the entire problem directly. For example, a constraint from the scheduling of Olympic Airways that we used in our test problems was "If a duty starts at a non-domestic airport, it cannot include the home base as an intermediate stop".

3 Solving the Crew Pairing Problem

Due to the large size of the search space, the nature of the constraints and the cost function, which usually are highly non-linear, we follow the approach of decomposing the CPP in two separate phases [18].

1. The pairing generation, where a large number of legal pairings, composed of the flight legs in the timetable, is generated and the cost of each pairing is calculated.

2. The optimization of the solution, where a subset of the generated pairings is selected, so that every flight leg in the schedule is included in at least one pairing and the total cost is minimized.

Furthermore, the pairing generation is itself decomposed in the process of generating a large set of legal duties from the flight legs and in the process of generating the set of pairings by using the duties previously found (Fig. 1).

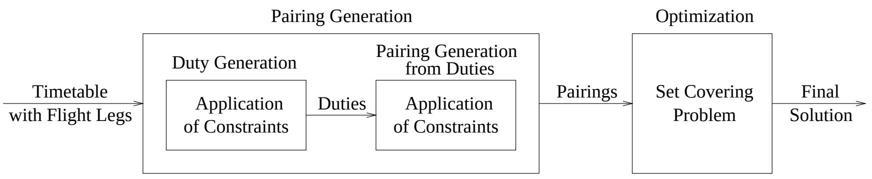
Pairing Generation
Optimization
Duty Generation
Pairing Generation from Duties
Timetable with Flight Legs
Application of Constraints
Duties
Application of Constraints
Pairings
Set Covering Problem
Final Solution

Fig. 1. The parts of the method for solving the CPP

One advantage of this approach is that the constraints and the cost function are taken into account only during the first phase and therefore the optimization is independent of them. In fact, once a large set of legal pairings has been generated, the second phase can be modelled as a *set covering problem*, a widely known combinatorial optimization problem.

3.1 The Pairing Generation

The input to the pairing generation is the set L of all the flight legs that appear in the timetable. The constraints are defined as a function $C : 2^L \rightarrow \{0, 1\}$, where 2^L is the powerset of L, $C(p) = 1$ if p is a legal pairing and $C(p) = 0$, if not. We want to generate the set of pairings $P = \{p \in 2^L \mid C(p) = 1\}$ which contains all pairings that satisfy the constraints. Modelling the constraints as a function like this makes the checking for the satisfaction of the constraints independent of the rest of the search.

Unfortunately, the size of the problems that normally arise in airlines is prohibitive for the complete generation of every valid pairing. For example, in an average sized problem of 1000 legs, we would have to check 2^{1000} pairings for validity, so an exhaustive search of the search space is obviously intractable. Furthermore, the more pairings we generate, the harder the optimization phase, which as we shall see is an NP-complete problem, becomes. Therefore, we wish to keep as few pairings as possible for the optimization, also making certain that these pairing will lead to a solution of high quality after the optimization.

The pairing generation is divided into two parts. During the first part a depth-first search is used, which systematically forms possible duties (sets of flight legs), checks if they satisfy the constraints and stores them if they do. The best of the generated duties are kept and provided as input to the second part, where a similar process is used to initially generate a large number of legal pairings and afterwards keep the best of them. During the entire run of the algorithm, special attention is given so that the number of generated pairings that contain each flight leg is balanced, i.e. each flight leg is part of almost as many pairings as any other. It is also important that each flight is part of a minimum number of pairings, so that not only is the existence of a solution guaranteed, but also there are multiple choices for the pairing that will be chosen to cover each flight leg. The presence of such multiple choices leads to greater diversity in the search space during the optimization phase, making it easier to find a solution of high quality.

Generation of Duties from Legs. We implement the duty generation as a depth-first search in the space of all possible subsets of the set L of all flight legs. Essential to this search is the function we use to model the constraints, which we shall call $validDuty : 2^L \rightarrow \{0, 1\} \times \{0, 1\}$. This function takes as input a set of flight legs and has two return values. The first return value is equal to 1 if the input legs form a valid duty that satisfies every constraint in the problem, otherwise 0 is returned. The second return value is equal to 1 if it is possible to create a valid duty by adding more legs to the set of input legs. In this case, the search should continue deeper, otherwise the search should stop and the second return value of *validDuty* is 0. A function named `searchForDuties` that can be used to search for valid duties is given next.

```
function searchForDuties(currentDuty, duties)
  [keep, continue] <- validDuty(currentDuty)
  if keep==1 then insert currentDuty into duties.
  if continue==1
    For each leg that departs after the arrival of the last leg of
    currentDuty and from the same airport do
      Insert the leg into currentDuty.
      searchForDuties(currentDuty, duties).
      remove the leg from currentDuty.
```

In the above fragment of pseudocode, `currentDuty` is a set of legs that is used to represent the duty that is checked for validity. The array `duties` holds all the valid duties that have been found. Calling the function `searchForDuties` with `currentDuty` containing only one leg will yield all valid duties that begin with that leg. Therefore, calling `searchForDuties` for each leg in L we obtain every valid duty.

After the completion of the search algorithm, we have generated a very large number of duties. Using so many duties as input to the next phases of the pairing generation and the optimization will cause serious performance problems. Therefore, a subset containing the duties of the highest quality possible should be chosen and kept. To this end, an algorithm that performs the selection of the best duties has been implemented. The criterion for keeping or not a duty is related to how useful this duty will be in the search for an optimal solution to the CPP. This is decided by a heuristic function, which returns the ratio of the flight time of the duty over the total duration of the duty.

This completes the duty generation. The set of duties that is kept will be used as input to the next part of the algorithm, the generation of the pairings.

Generation of Pairings from Duties. The generation of pairings from the duties is performed, using a depth-first search, in practically the same way as the generation of duties. The set of legs that were used as input is replaced by the set of duties and the algorithm instead of generating a set of valid duties generates a set of valid pairings. Since we gave a detailed description of that depth-first algorithm in the previous paragraph, we shall not repeat it here.

3.2 The Optimization

Once we have generated a large set of pairings, we continue to the optimization phase, which is modelled as a set covering problem, defined as follows:

Given a set M with m elements and n subsets $M_j \subseteq M$ with associated costs c_j, $j = 1, 2 \dots n$, find a subset S of $\{1, 2 \dots n\}$ such that $\bigcup_{j \in S} M_j = M$ and $\sum_{j \in S} c_j$ is minimized.

In other words, we seek a collection of subsets such that every element of M is contained in a least one selected subset and the total cost of all selected subsets is minimum.

The correspondence to the optimization phase of the CPP is direct, by setting M equal to the set of all flight legs and M_j equal to the generated pairings.

Notice that by modelling the problem as a set covering problem, instead of a set partitioning, we allow solutions which may contain crews travelling as passengers on some flights. As we have mentioned, this is acceptable and in some cases it may even yield better solutions than forcing exactly one crew per flight. Furthermore, it is possible that no feasible solutions exist under the constraint that each leg is covered exactly once. Set covering has been proven to be NP-complete [9] and, therefore, no algorithm is known that can find the optimal solution efficiently, for input data of significant size. In fact, the best reported methods for exact optimization are based on branch-and-bound and deal with problems of sizes around 400 rows (flight legs) and 4000 columns (pairings) [12]. However, the size of problems that commonly arise in airlines is far larger, reaching thousands of rows and millions of columns. Therefore, in order to tackle problems of this size, approximate methods are used, which produce solutions that only approximate the optimal, but can handle large problems efficiently. In [18] and [4], two such methods are presented and success is reported with problems of up to 10000 rows and 1000000 columns.

The method we propose is based on [3], where a genetic algorithm is given that can be used for efficiently solving the set covering problem. We modified this algorithm to better fit the specific characteristics of the set covering problems that arise as part of a CPP.

Overview of Genetic Algorithms. *Genetic algorithms* (GA) are methods for random search based on the evolutionary process of species found in nature. They were introduced in 1975 [13] and since then they have been used successively on a wide range of combinatorial optimization problems.

According to the theory of evolution [5], populations in nature evolve by following the process of natural selection and survival of the fittest. The members of the population that are able to adjust to changes in the environment are more likely to survive and reproduce. On the other hand, individuals who are less fit become extinct. Through this process the genes of the fittest individuals are passed on to future generations and eventually become predominant as generations pass. This mix of characteristics from fit ancestors produces even fitter individuals and in this way species become stronger and better adjusted to the environment.

Genetic algorithms are based on the simulation of the above process, by taking an initial population of solutions and applying a number of genetic operators on it. Each individual of the population is a possible solution to the problem, encoded as a string (usually binary) called *chromosome*. Each character of the chromosome is called a *gene*. The fitness of each individual is represented by an objective function, which is a real valued function defined over the set of all chromosomes. The value of the objective function is representative of the quality of the corresponding solution. The procedure of combining two members of the population to produce offsprings is called *crossover*. Through crossover, individuals containing genes from both parents are produced. By combining high quality solutions we hope that children will inherit the best genes of both parents. In order to avoid getting stuck in local minima during the search, *mutation* is used, i.e. the random change of a few genes of the offspring, after the crossover. Finally, the new individuals replace the weaker members of the existing population and this procedure continues until a satisfactory solution is found.

Genetic Algorithm for Crew Pairing Optimization. The method we use for the set covering problem follows the general framework of GA we presented previously. Certain modifications and additions have been made to accommodate with the specifics of the problem, including a method for correcting solutions that violate constraints, i.e. solutions where there exist flights that are not covered by any selected pairing, turning them into feasible ones.

A binary string coding is used for chromosomes with each chromosome having length equal to the number of pairings. Each gene corresponds to one pairing and when it is 0 it means that the corresponding pairing is not part of the solution. If the gene is equal to 1, then the corresponding pairing is included in the solution. The objective function we use to represent the fitness of each individual is

$$f = \sum_i c_i g_i + deadheadingPenalty \cdot deadheadedFlights$$

where c_i is the cost of the i-th pairing, g_i is the value of the i-th gene of the solution, *deadheadingPenalty* is a constant that is used to penalize flights which are covered more than once in the solution and *deadheadedFlights* is the number of such flights.

By using this cost function, we are able to reach solutions that contain low cost pairings, but also have a low percentage of deadheading, which, as we have mentioned, poses an additional cost to airlines. Furthermore, we have experimentally noticed that a careful assignment of the constant *deadheadingPenalty* may help the genetic algorithm to converge to a high quality solution faster.

The selection of the members of the population that will become parents is based on their order in the population, sorted in descending order based on their fitness function. That means that the fittest individual is the most likely to be selected, the second fittest individual is a little less likely to be chosen, and so on. Note that the probability of selection depends exclusively on the order in the population and not on the absolute value of the fitness function for the individual. In exactly the same way, we choose the member of the population that will be replaced at every generation. The only difference is that in this case the individual that is the least fit has the highest probability of being chosen.

Once two parents have been chosen from the population, the crossover operation is performed on them in order to produce a new solution that inherits characteristics from both parents. We use the *uniform crossover operator.* This means that if a gene has the same value in the chromosomes of both parents, this value is assigned to the same gene of the offspring chromosome. If a gene is equal to 0 for one parent and equal to 1 for the other, the corresponding gene of their offspring may become either 0 or 1, with equal probability. The reason for choosing this variation of the crossover operator was that it provided a better shuffling of the parents' genes in forming the offspring.

After crossover has been used to produce a new individual, the mutation is applied on it. The purpose of mutation is to prevent the search from getting trapped in the local minima, by randomly altering a few genes of the chromosome and therefore directing the search towards new areas in the search space. We propose a method of mutation that depends on the density (the percentage of genes equal to 1) of the fittest individual of the population. Initially, a fixed number of genes are randomly selected from the chromosome. These are the genes that will be mutated. Each of the genes is mutated to 0 with probability equal to the percentage of 0s in the chromosome of the fittest individual, otherwise it is mutated to 1. For example, if in the chromosome of the fittest individual 900 genes are 0 and 100 genes are 1, then each gene is mutated to 0 with probability 90% or to 1 with probability 10%. By mutating in this way, we are able to keep information about the density of fit individuals in the offspring's chromosome. Such information is preserved during crossover, but if mutation to 0 and 1 was made with equal probability, such information, which is quite critical for getting good results, would be lost.

By looking at the way that crossover and mutation are performed, it is obvious that after these operations, the new solution will not necessarily be feasible, i.e. not every flight leg will be present in at least one pairing. In order to avoid having infeasible solutions in the population, after the mutation, we apply a correction algorithm on the new chromosome. This algorithm works by changing the value of some genes to 1 and as a result by adding the corresponding pairings to the solution, until every flight leg is covered and the solution becomes feasible. For each uncovered leg in the solution, we add a pairing that covers that leg. A simple heuristic is used to help us select one pairing among all the pairings which contain that particular leg. That heuristic favours pairings of low cost that when selected cover as many uncovered legs as possible and as few legs that have already been covered as possible. For a more detailed and formal description of the correction method see [3].

One last important thing that should be mentioned about the genetic algorithm is the way that the initial population is created. It is critical to the performance of the search that the initial population is as diverse as possible, so that a large part of the search space will be explored early. Therefore, the best method to initialize the population is one that generates individuals as randomly as possible. Specifically, we randomly insert in the solution new pairings that don't have common legs with the other pairings already selected. Then, once we've reached the point where we can't find such a pairing, we run the correction algorithm on the solution to make it a feasible one. Through this process, we obtain a population whose members are feasible solutions of a reasonably high quality. Furthermore, since a great deal of randomness is present in the generation of individuals, the population created displays the wished diversity.

Having presented the individual parts of the algorithm, we can now give its complete description:

```
Randomly generate an initial population of chromosomes.
Repeat until a satisfactory solution is reached
  Select two of the fittest members of the population for parents.
  Apply crossover on the parents to produce a child chromosome.
  Apply mutation on the child chromosome.
  Correct the child chromosome so that it becomes a feasible solution.
  Select one of the weakest members of the population to be replaced.
  Replace the selected individual with the child.
```

4 Experimental Results

In order to test the efficiency of the proposed algorithm, we ran it using real input data from the flight schedule of Olympic Airways. We used the flight schedule for the 737–200 fleet, for April 1998. The schedule consisted of 2100 flight legs, passing through 29 different airports. The pairings were generated based on the regulations of Olympic Airlines for cockpit crews. Examples of the constraints that were used are the following:

- The minimum transition time between legs in a duty is 45 minutes.
- The flight time of a duty has a maximum value, which is 15 hours for long-range flights, 9 hours for non-domestic flights and 8 hours for domestic flights.
- At most one duty intersects with any calendar day.

As the cost of each pairing we used the value of the function

$$floor(1000 \cdot duration\_of\_pairing/flight\_time\_of\_pairing)$$

where $duration\_of\_pairing$ is the total time that the pairing started from the beginning of its first duty until the end of its last duty, and $flight\_time\_of\_pairing$ is the sum of the flight times of all the flight legs in that pairing.

During the pairing generation phase, initially 70358 legal duties were generated, out of which the 22090 best duties were stored for the next phase. The average number of flight legs in each of these duties was 3.88 legs per duty. Afterwards, using these duties as input we generated 66425 legal pairings and kept the 11981 best of them. The average cost of each pairing was 6148.24 and it was composed of 5.3268 flight legs in average. Most of the pairings that were finally kept were composed of either one or two duties. It is easy to see that this is a natural consequence of the cost function we used, which significantly favours smaller pairings.

In the optimization phase, we ran the genetic algorithm to find the best subset of the 11981 pairings, which covered the 2100 flight legs. After some experimentation, it was decided to fix the population of each generation to 10 individuals. Increasing the population size led sometimes to better results, but with a significant decrease in efficiency. In addition, the *deadheadingPenalty* parameter of the fitness function was decided, experimentally, to be fixed to 1000, giving a total penalty of around 1% for deadheaded flights to the fitness of an individual. This contribution, although relatively small, was sufficient to guide the algorithm to solutions with very few deadheaded flights. The cost of the best solution found, plotted against the number of generations that have passed, appears in Fig. 2. The progress of the optimization in greater detail, for various generations, is shown in Table 1. The table gives the total cost, the number of pairings and the number of legs covered by more that one pairing (deadheaded flights) for the best solution of the generation. We can see that as the genetic algorithm runs, the solution constantly gets better. Even though initially many flights are deadheaded and the cost is high, eventually after 20000 generations, we find a solution of zero deadheading and of cost 976004. This requires about 30 minutes of elapsed time on a PC with a 350 MHz Pentium III processor.

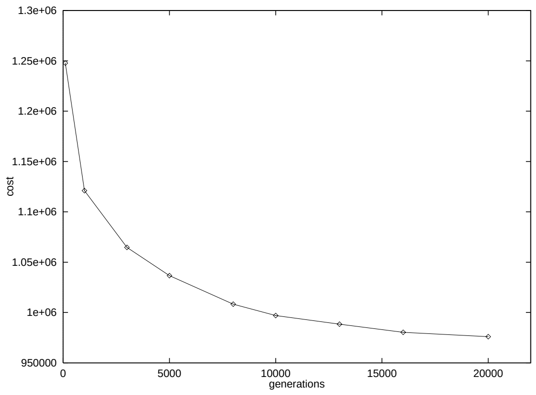

Fig. 2. The progress of the optimization

Table 1. The best solution of various generations

Generation	100	1000	3000	5000	8000	10000	13000	16000	20000
Cost	1247776	1121057	1064647	1036596	1008276	996994	988418	980344	976004
Pairings	635	592	567	551	537	531	529	526	525
Deadheading	71	10	6	6	4	4	2	0	0

Finally, we'll compare the result we found with the result found by the project Parachute [11], a project that combines constraint programming with parallel processing, in order to automatically solve the crew scheduling problem. We ran Parachute on the same data set (April 1998) using the same constraints and the same cost function. The comparative results, showing the total cost of the solution, the number of pairings in the solution and the number of deadheaded flight legs, appear in Table 2. Our proposed method gives a solution of cost lower by 194148 (16.5%) and has no deadheaded flight legs. The running times of both methods were of the same order of magnitude.

Table 2. Comparison of our method and Parachute

	Total Cost	No. of Pairings	Deadheaded Legs
Our Method	976004	525	0
Parachute	1170152	611	25 (1.1%)

Another comparison we carried out was to test our method with the datasets that were used by Beasley and Chu in [3]. It has to be noted, though, that these datasets represented various set covering problems, not necessarily similar in nature to ones coming from crew pairing problems like the one we faced in our work. For example, in these datasets, there exists 30–70% of overcovering in the solutions, while in CPPs the corresponding percentage is 1–5%. In addition, the density of the set covering problems in [3] is 2–20%, while, normally, in CPPs, the density is around 1%. However, the results we obtained by running our method, which is definitely tuned for CPPs, on these set covering problems were of similar, though not better, quality to those achieved by Beasley and Chu.

5 Conclusions

In this paper, we have given a method that can be used to solve the crew pairing problem. We separated the problem into two phases, the pairing generation and the optimization, and we used a depth-first search for the first phase and a genetic algorithm, extending work by Beasley and Chu, for the second. In order to test the performance of the algorithm, we ran it with real data from the schedule of Olympic Airways. The experimental results were very satisfactory, giving a low cost solution and improving on previous results on the same data set. Therefore, through this experiment, we have shown that the method we propose can be a viable solution for the crew pairing process of airlines.

References

1. Anbil R., Gelman E., Patty B., Tanga R. Recent Advances in Crew-Pairing Optimization at American Airlines. *Interfaces*, 21(1):62–74, 1991.
2. Andersson E., Housos E., Kohl N., Wedelin D. Crew Pairing Optimization. in Yu G. (ed.) *Operations Research in the Airline Industry.* Kluwer Academic Publishing, 1997.
3. Beasley J. E., Chu P. C. A Genetic Algorithm for the Set Covering Problem. *European Journal of Operational Research*, 94:392–404, 1996.
4. Carpara A., Fischetti M., Toth P. A Heuristic Algorithm for the Set Covering Problem. *Operations Research*, 47:730–743, 1999.
5. Darwin C. *The Origin of Species.* John Murray, 1859.
6. Desaulniers J., Desrosiers J., Dumas Y., Marc S., Rioux B., Solomon M. M., Soumis F. Crew Pairing at Air France. *European Journal of Operational Research*, 97:245–259, 1997.
7. Eremeev A. A Genetic Algorithm with a Non-Binary Representation for the Set Covering Problem. In *Proceedings of Operations Research '98*, pages 175–181, 1999.
8. Etschmaier M. M., Mathaisel D. F. Airline Scheduling: An Overview. *Transportation Science*, 19:127–138, 1985.
9. Garey M. R, Johnson D. S. *Computers and Intractabitity: A Guide to the Theory of NP-Completeness.* W. H Freeman, 1979.
10. Goumopoulos C., Alefragis P., Housos E. Parallel Algorithms for Airline Crew Planning on Networks of Workstations. In *Proceedings of the International Conference on Parallel Processing*, 1998.
11. Halatsis C., Stamatopoulos P., Karali I., Bitsikas T., Fessakis G., Schizas A., Sfakianakis S., Fouskakis C., Koukoumpetsos T., Papageorgiou D. Crew Scheduling Based on Constraint Programming: The PARACHUTE Experience. In *Proceedings of the 3rd Hellenic-European Conference on Mathematics and Informatics HERMIS '96*, pages 424–431, 1996.
12. Hoffman K. L., Padberg M. Solving Airline Crew Scheduling Problems by Branch and Cut. *Management Science*, 39:657–682, 1993.
13. Holland J. H. *Adaption in Natural and Artificial Systems.* MIT Press, 1975.
14. Lagerholm M., Peterson C., Söderberg B. Airline Crew Scheduling Using Potts Mean Field Techniques. *European Journal of Operational Research*, 120:81–96, 2000.
15. Marchiori E., Steenbeek A. An Evolutionary Algorithm for Large Scale Set Covering Problems with Application to Airline Crew Scheduling. In *Real-World Applications of Evolutionary Computing*, LNCS 1803, pages 367–381, 2000.
16. Ozdemir H. T., Mohan C. Flight Graph Based Genetic Algorithm for Crew Scheduling in Airlines. *Information Sciences*, 133:165–173, 2001.
17. Pavlopoulou C., Gionis A., Stamatopoulos P., Halatsis C. Crew Pairing Optimization Based on CLP. In *Proceedings of the 2nd International Conference on the Practical Applications of Constraint Technology PACT '96*, pages 191–210, 1996.
18. Wedelin D. The Design of a 0-1 Integer Optimizer and its Application in the Carmen System. *European Journal of Operational Research*, 87:722–730, 1995.
19. Yan S., Tung T.-T., Tu Y.-P. Optimal Construction of Airline Individual Crew Pairings. *Computers and Operations Research*, 29:341–363, 2002.