Domain Knowledge and Representation in Genetic Algorithms for Real World Scheduling Problems

Ioannis T. Christou
Delta Technology
1001 International Blvd., Dept. 709
Atlanta, GA 30354

Armand Zakarian
Delta Technology
1001 International Blvd., Dept. 709
Atlanta, GA 30354

Abstract

This paper discusses the issues that arise in the design and implementation of an industrial-strength evolutionary-based system for the optimization of the monthly work schedules for the pilots of Delta Air Lines. We detail the system's multiple and often conflicting goals and rules, providing the background for understanding the problem. Then, we describe the algorithm that we use to solve it. One important difference of our approach from other commonly used GA implementations is our use of the GA as a feasibility procedure: the first phase of our approach is responsible for building a very high quality partial solution based on the domain knowledge of the problem. The GA is responsible for completing this solution, finding a feasible solution to the remaining problem. We illustrate the impact that the representation can have on overall performance by comparing two implementations of the same algorithm based on two "orthogonal" encodings of the problem at hand. The computational results show how with the help of hill climbing techniques in the evaluation function (either as repair procedures or as part of a decoder function) we were able to successfully put the system into production. We also make computational comparisons with exact methods which show a clear, though unexpected, superiority of the GA against them.

1 Introduction

Genetic Algorithm research has matured a lot during the past decade, both in theory as well as in practice. From the theoretical standpoint, many aspects of the statistical mechanisms responsible for the power of GAs have been well investigated, providing bounds on the effectiveness of the algorithms under various assumptions [13]. From the practical point of view however, researchers had mixed experiences with different implementations and different domains, which may be partly explained by the "No Free Lunch" theorems [14] stating that when averaged over all problem domains the effectiveness of GAs is as good as that of exhaustive search. In the following pages, we provide a detailed description of the issues arising in the design and implementation of an industrial-strength system for solving the BidLine Generation Problem (BLP), the problem of generating monthly work schedules for the pilots of an airline. We show how by exploiting the domain knowledge of the problem, and with the help of appropriate representations and local search heuristics, we were successful in putting the system in production at Delta Air Lines. This system provides the planners with very high quality solutions together with response times that are much more efficient than the previous semi-automatic modes used at Delta.

1.1 The BidLine Generation Problem

Each month, an airline must construct a set of *trips* (also known as *pairings*) to be flown by the airline crew members. These trips are sequences of flights (also known in the industry as *legs*) beginning and ending in a base city.

This problem is the *Crew Scheduling* problem, which in the airline industry is often called *Crew Pairing* (CP) problem, and because of its substantial economic impact, it has received wide consideration; it can be formulated as a set covering problem, and approaches for solving it include exact optimization procedures [2] as well as heuristic methods based on genetic algorithms [10] and other techniques.

Once CP is solved, the BLP is defined as the problem of putting together these trips to form month-long work schedules for the pilots of the airline. These schedules are called *bidlines*, or lines of time, or simply lines. Once the lines of time are finalized, the pilots bid for them, and are awarded their preferences according to their seniority.

1.2 The Rules and Objectives of BidLine Generation

As is the case with the CP problem, several rules determine the validity of the placement of a trip in a line. These rules are defined in terms of certain properties of the trips to be placed on a line. The defining properties are the number of days a trip lasts, the number of flying hours in each day of the trip, and the *credit* of a trip, which roughly corresponds to the total flying time of the trip. The most important of these rules are the *Credit Rule* stating that the total credit of a line must be within a certain window $[C_l, C_u]$, the *Flying Time Window Rule* stating that for any sliding window of W calendar days there must be no more than f hours of flying, and the *Coverage Rule* stating that the lines must cover all the trips except for a small number of trips whose total credit will be less than C_l.

The objectives are two-fold: on one hand we want to minimize the total number of lines built so as to minimize the staffing needs of the airline. On the other hand, since the airline is bound by contract to build high quality lines we want to build lines with as high quality as possible. Quality is a multi-term concept that attempts to capture the needs and desires of the pilots for quality of life issues. A contract between the Delta Air Lines Pilots Association (ALPA) and Delta Air Lines defines the notion of "purity" of a line, and this notion can be further divided in "day-purity" and "trip-purity". A line is "day-pure" when every trip in it departs on the same day of the week; similarly, a line is "trip-pure" when it consists of pairings that are essentially the same. The quality of an overall solution is further complicated by other desiderata.

Overall, the quality of the solution is a rather complicated function of properties of the lines built but with no given closed form expression. Nevertheless, in [8] the authors propose a cost function to compute the cost of a (legal) line that uses adjustable parameters to influence the quality of a line. Then, they use column generation techniques to solve the BLP. One main disadvantage of this approach is that it is not a fully automatic procedure. Furthermore, they do not solve the problem to completion since in all their reported runs, they leave an unacceptably large number of trips unassigned.

1.3 Formulation

BLP is a true set partitioning problem with a side constraint for the open time, and from this respect, it is a harder problem than the CP problem, because in CP more than one crew can cover the same leg for an extra cost. In any case, as is shown in [4], the rules that apply to the BLP, make it NP-hard.

In order to give an exact mathematical definition of the BLP, let

$$\mathbf{L} = [\mathbf{l_1 l_2} \dots \mathbf{l_n}]$$

be the matrix whose columns represent all the valid lines of time $\mathbf{l_i}$ that can be built using any valid combination of trips in the current category. We represent these lines as p-dimensional vectors in $\mathbf{B}^p$ where $\mathbf{B} = \{0, 1\}$ and p is the number of trips in our category. If the i-th bit of the vector is set, then the line represented by that vector must include the i-th trip in it. Assuming we have a cost function $\mathbf{q}(\mathbf{l})$ that assigns to every legal bidline a cost representing the quality of the line, we can formulate the BLP as follows:

$$\max \sum_{i=1}^{n} \mathbf{q}(\mathbf{l_i}) x_i \tag{1}$$

$$s.t. \begin{cases} \begin{bmatrix} \mathbf{L} | I \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} & = e \\ \begin{bmatrix} \mathbf{0} | c_1 c_2 \dots c_p \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} & \leq C_l \\ x & \in \mathbf{B}^n \\ s & \in \mathbf{B}^p \end{cases}$$

where

$$\begin{cases} c_i & = \text{credit of the i-th trip.} \\ e & = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \end{cases}$$

Nevertheless, finding a cost function that will accurately measure the quality of a line is a highly non-trivial task. But ALPA provides a description of a series of steps that should be taken in order to obtain high quality lines. The existence of these guidelines led us to the decision to use a two-phase system. In the first phase, a greedy algorithm based on these guidelines (to be described in 2.1) constructs as many high quality lines of time as possible. In the final second phase (to be described in 2.2), a Genetic Algorithm solves the remaining problem of putting together into legal bidlines all the remaining open trips. Our approach is thus almost the opposite of the approach taken at Federal Express (described in [1]) where the authors formulate a cost function that approximates the objectives of the BLP process, and use Simulated Annealing (SA) to compute as many high quality lines as possible, and then a greedy algorithm to act as a hill climber for finishing up. The authors of the FedEx paper did not have to provide a complete solution since their system was to be used as a cost evaluation tool.

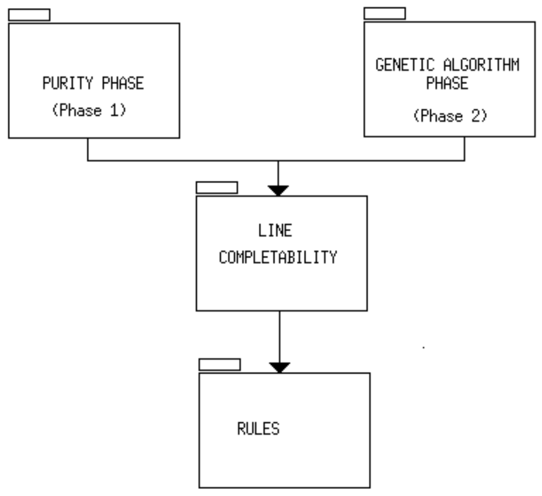

Figure 1: System Design

2 System Architecture

Our approach is guided by the domain knowledge available to us in an almost algorithmic form for what constitutes a high-quality line. The system, in the Object-Oriented (OO) fashion is divided among four major components, or modules, that provide implementations of well-defined interfaces for certain responsibilities. The four modules (and their interactions) are shown in figure 1. The rule subsystem employs the strategy pattern [6] to implement all rules efficiently in a uniform manner. The line completability component, implements a mechanism for answering the following question: Given a line with some trips assigned in it, is there a set of open trips that can complete this line, i.e. render it a legal line? The mechanism implemented is a tree construction and traversal that alternates between depth-first and breadth-first search.

2.1 The First Phase: Domain Knowledge Incorporation

In the first phase of the system, called the *purity phase*, we build high quality lines following the steps below:

1 Family Creation: Choose among the open trips a set of trips that are "essentially" the same. This set forms a family of pairings to be used in line construction.

2 Perfect Lines: Build as many perfect (both day-pure *and* trip-pure) lines from this family as possible.

3 Trip-pure Lines: Build as many lines as possible using trips only from the family.

4 Day-pure Lines: Build as many lines as possible maintaining day-purity, by using one or more *filler* trips that meet best certain criteria.

5 Line Completion: If there are still left open trips from the family, build as many lines as possible that can be built with less than a specified number of different fillers.

6 Estimation: Estimate the number of remaining lines to be built from the current remaining open (unassigned) trips.

7 Stacking Test: Perform a *stacking test* that ensures that there exists no interval of time during which there are more open trips than the estimated number of lines. If the stacking test fails, undo as many of the lines built as necessary to pass the stacking test.

8 Loop: While there remain sufficient open trips that can form a family repeat the above steps.

The algorithm captures many of the techniques for line building that the planners used to build high-quality lines of time, and it follows the contract between ALPA and the company.

The stacking test in the seventh step implements a semi-assignment algorithm [9], that ensures that no interval of time in the period is "neglected".

2.2 The Second Phase: Solving Set Partitioning with a Genetic Algorithm

Once the first phase of the algorithm ends, we are left with a number of high quality complete lines of time which constitute a *partial solution* to the overall BLP. To complete the solution, we must now arrange the remaining open trips (which add up to a total credit that is usually enough to build another 20% to 30% more lines) into valid lines of time.

To achieve this goal, we employ the GA paradigm as the search mechanism for exploring the space of valid combinations of trips into lines. In this light, the GA acts as a feasibility procedure that attempts to arrange the remaining open trips into valid lines of time leaving open only a few trips whose credit sums up to less than C_l.

We designed a decoder strategy combined with repair procedures and hill-climbing heuristics. All our representations (to be discussed in detail in the next section) treat the genotype of an individual as a "blueprint" or guideline for the phenotype of the individual. The phenotype of an individual is an arrangement of some of the open trips into valid lines of time. The objective value of the individual is the total open time remaining after the objective function has performed this arrangement.

As soon as a feasible solution is found, the GA terminates returning this feasible solution. In this light, the approach we take is similar to approaches described in [11, 12] where the authors use decoder techniques to solve highly constrained optimization problems allowing for infeasible individuals to drive the genetic search.

In order to improve the initial arrangement of open trips into lines of time done by the "blueprint" of the individual, if the open time remaining is high (we are still infeasible), we execute a hill-climbing heuristic that attempts to swap trips between lines that were just constructed and trips in the open time; we perform the swap only if it decreases the new overall open time. This swap heuristic has a complexity of $O(t^2)$ where t is the number of trips left open after the purity phase. When the open time remaining after the hill-climber returns is close to the feasible region, another hill-climber attempts to modify lines built in the purity phase in order to increase their total credit. This procedure can often find a feasible solution in early generations, because the lines built in the purity phase contain lines that can increase their total credit by swapping low-credit trips with high-credit trips from the open time.

The amount of credit that remains open after the final arrangement of trips into lines forms the return value of the evaluation function that must be minimized. We convert objective function value to fitness value according to standard linear mapping to a fixed interval $[0, MaxFit]$, so that the less the objective function value is, the bigger the fitness of the individual becomes.

The workings of the GA we used are explained in detail in [3, 4]. The evolution follows the steady-state model with a high degree of reproduction (defaulted to 70%). Selective pressure is done using the roulette wheel sampling strategy for selection of parents, and the standard operators of one-point crossover and mutation are applied to produce the new generation. Schematically, figure 2 shows the workings of the GA.

DGA (our GA), is a distributed asynchronous GA based on the island model of evolution, with a simple policy for migrations of individuals among the islands. Whenever, through chance, an island becomes almost deserted, the best fit individuals from the other islands migrate to this empty island with a very high probability. The empty island becomes fertile ground for highly fit individuals with no common ancestors to reproduce. The distributed nature of the GA and the aging mechanism it employs are designed to help maintain the diversity of the population without the aggressive behavior the mutation operator exhibits at high rates [3]. The aging mechanism removes from each population all individuals that have reached their lifespan (measured in number of generations they lived). This lifespan is computed as a random variable with a mean value

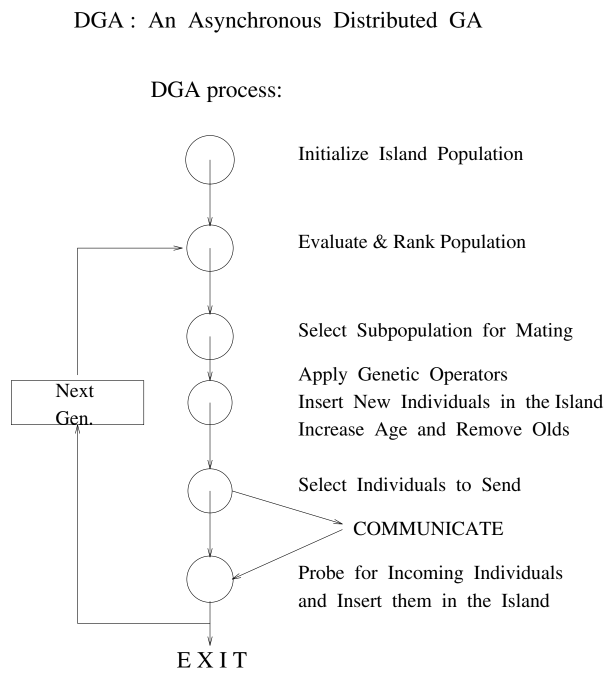
DGA : An Asynchronous Distributed GA
DGA process:
Initialize Island Population
Evaluate & Rank Population
Select Subpopulation for Mating
Apply Genetic Operators
Insert New Individuals in the Island
Increase Age and Remove Olds
Select Individuals to Send
COMMUNICATE
Probe for Incoming Individuals
and Insert them in the Island
Next Gen.
E X I T

Figure 2: The workings of a DGA process

proportional to its fitness.

Finally, if after a fixed number of generations (defaulted to 30) we do not have a feasible solution, we break up a number of lines (usually 10) built in the purity phase and we start the GA again with more open trips -and thus with a less constrained problem to solve. As mentioned before, the stopping criterion is the appearance of a feasible solution. The GA would also stop if it ever ran for 30 generations without finding a feasible solution and there were no lines from the purity phase left unbroken (a situation that has never materialized.)

3 Representation Issues

3.1 Trips to Lines Representation

Both representations we use are based on decoder techniques: the first one is a simple and straightforward representation. Each position in the individual's string (allele) represents a trip to be placed in the line represented by the value of the string at this position (the gene). A decoder scans the string from left to right attempting to assign each trip to the lines indicated by the gene. If this assignment is not possible, we assign the trip to another line as close as possible to the one indicated. In such a case, we change the chromosome's gene at the current allele to indicate the line where the trip was actually assigned. The representation is shown schematically in figure 3.

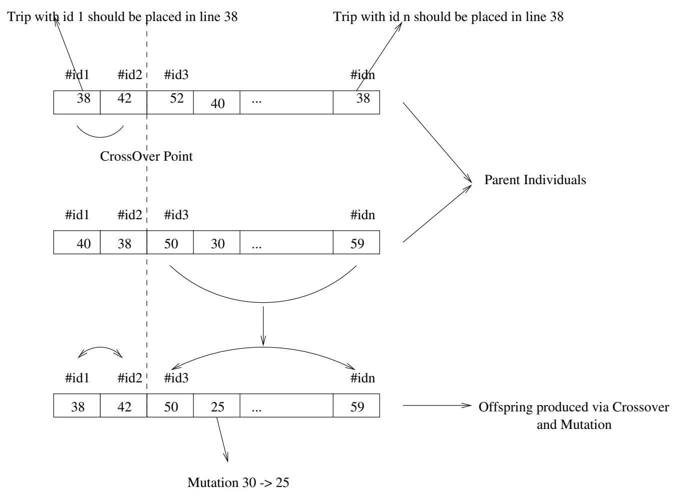

Figure 3: Trips to Lines Representation

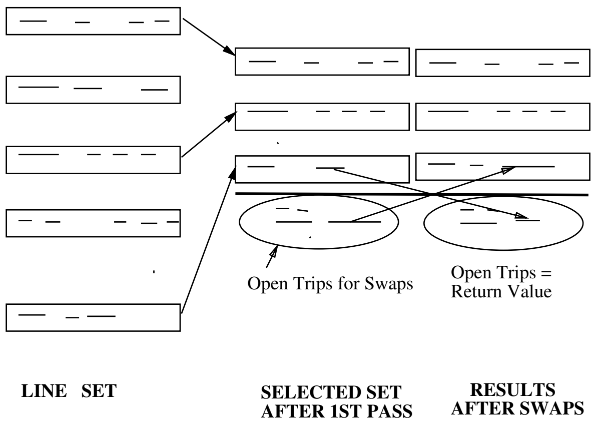

Figure 4: In the lines to trips representation, a set of (intersecting) lines is maintained, and an individual is a string of indices to this set. Swaps are performed to improve upon the solution provided by the initial decoding of the string.

3.2 Groups Representation

The second representation is based on the ideas of multi coordination techniques in GAs for grouping problems [5]. These ideas attempt to exploit the observation that in grouping problems such as set partitioning, it is the groups that should be the building blocks of the solutions and not the objects, and thus any representation and genetic operators should facilitate the transmission of whole groups from parents to children in order for the genetic search to be successful. Accordingly, the representation requires maintaining a set L of legal lines of time. For this reason, in this representation, before we run the GA, we generate a number of lines from the open trips using the familiar tree building algorithms used in the purity phase of the system. We generate these lines to have as much credit as possible, and to be as day-pure as possible (trip purity is hardly possible at this point). Any two lines in this set may have trips in common, which of course implies that in the final solution such lines cannot be both present. Each position in an individual's string now is an index to a line from the set L (see figure 4). An individual's chromosome is a string of indices to lines in L and we use a decoder to create lines of time as follows: we scan the chromosome from left to right; we select the trips in the line indicated by the gene we are currently examining, and we create a (possibly partial) line that is restricted to those trips we selected that are still open.

Often, after the decoder ends, many of the last lines are left incomplete with only a few trips in them because the original lines in L contained trips that were present in other lines used before in the string. From that point on, the same hill-climbing heuristics will take over and attempt to complete the partial lines and then improve upon the solution.

After the hill climber is executed, we augment the set L by any new lines that we created and that are not present in L. Then, we modify the individual's chromosome to indicate the lines in the set L that it contains. This idea (often called Lamarckian evolution) has been used many times in scheduling problems and usually helps improve the GA process (find a feasible solution faster).

The main advantage of this representation is that it represents groups of trips that should form a line rather than simple guidelines for assignments of individual trips to lines. As a result, any standard crossover operator such as one-point crossover will create offspring that inherit whole lines from each parent, instead of the less meaningful guidelines of assignments of trips to lines; indeed the representation of trips to lines will create offspring that more often than not lead to large number of lines that are not present in either parent thus hindering the evolutionary process. For a more thorough discussion of this effect see [5].

4 The Runs

In December of 1997, an industrial-strength version of the system described in this paper was put in production at Delta Air Lines and has been used ever since to build the monthly work schedules of the pilots. The representation we used was the simpler to implement trips-to-lines representation. We wrote the system, called LOTO, entirely in C++ with the sole exception of the Genetic Algorithm library which predates the system and is written in ANSI C. DGA uses PVM 3.3 for interprocessor communication when running on a parallel or distributed computing environment. The system uses the Sybase relational DBMS for all I/O purposes. The results we present in this section are obtained from a uniprocessor IBM 595 RS6000 RISC machine with 256MB of on-board memory.

PROBLEM	Groups Representation				
	TL	PP	TP	DP	Time (mins)
767 CPT ATL 12/99	340	42	66	113	212
767 CPT CVG 12/99	98	13	17	20	50
767 CPT DFW 07/99	75	10	12	29	1
767 FO DFW 07/99	79	9	13	38	1
727 CPT ATL 05/99	156	27	34	82	24
727 CPT ATL 02/99	165	0	4	107	5
M88 FO DFW 06/99	148	15	24	62	18
M88 CPT DFW 02/99	151	24	31	88	48
L10 CPT ATL 07/99	78	12	33	32	2

Table 1: Results using the Groups Representation

PROBLEM	Trips to Lines Representation				
	TL	PP	TP	DP	Time (mins)
767 CPT ATL 12/99	341	45	68	119	221
767 CPT CVG 12/99	99	16	21	33	13
767 CPT DFW 07/99	76	12	14	31	2
767 FO DFW 07/99	79	8	12	38	1
727 CPT ATL 05/99	156	27	34	75	57
727 CPT ATL 02/99	166	0	4	106	49
M88 FO DFW 06/99	149	10	19	59	10
M88 CPT DFW 02/99	151	22	29	79	18
L10 CPT ATL 07/99	79	11	36	33	2

Table 2: Results using the Trips to Lines Representation

Table 1 shows the results of a number of runs we did over a wide range of categories and monthly periods using the groups representation. The column ”TL” represents the total number of lines constructed for this category. The columns ”PP”, ”TP”, and ”DP” represent the number of perfectly pure lines, trip-pure lines and day-pure lines respectively. The last column shows the run times in minutes of user time. The columns in table 2 have the same meaning but they represent instead results from runs we did using the simpler trips-to-lines representation.

A comparison of these results shows that the groups representation may give better results in some cases (fewer lines of time needed for a category, combined with more day-pure lines and sometimes even more perfect lines). However the difference in the quality of these results is not very significant. Far more significant is the following observation: when the GA runs without the hill-climber heuristics, both representations fail to find a feasible solution in most categories. Apparently the genetic operators of crossover and mutation by themselves can only produce a partial solution that the swaps can then significantly improve.

We also attempted to solve the BLP by exact methods. We used the formulation of section 1.3. We assigned to each line a cost that is a convex combination of the inverse of the total number of different trips in the line and the inverse of the total number of different trip departure days. We restricted the matrix $\mathbf{L}$ so as not to contain all valid lines (which cannot be generated in any reasonable time interval), but just all the lines generated by our GA in the evolutionary process. Basically, after having obtained a solution to the BLP from the GA, we saved *all* the lines generated, and passed the resulting set partitioning problem to CPLEX [7], a widely used LP/IP solver for solving linear, integer and combinatorial optimization problems. The test cases we tried were two ”easy” problems, namely the June 99 L1011 Atlanta Captains' category, and the June 99 MD88 Dallas Fort-Worth Captains' category. The larger of the two test cases contained less than two thousand columns, and a feasible solution was guaranteed because the GA had found it. Surprisingly, even after running for more than 48 hours on a 4 processor HP 9000 V-class server, CPLEX had not found even a single integer solution to any of the problems. This outcome shows that the BLP problem cannot be solved by exact methods using classical branch & bound methods. The reason probably lies in the fact that while the LP relaxation of the resulting problem can be solved within a few seconds, the nature of the constraints admits very few solutions, and therefore implicit enumeration has very little chance of coming up with a solution. It is the same reason that forces the system to spend most of its time in the GA phase trying to complete the partial solution constructed in the first phase. This also helps explain why the GA phase cannot find a feasible solution without the help of swaps. With the help of local improvement techniques, implicitly many more lines are examined from each individual that is being evaluated than when local swaps are turned off. In a landscape that has few feasible points, the genetic operators alone simply lack the search power to locate a desired point.

5 Conclusions

We presented several issues that we faced during the design, implementation and evolution of a GA-based software system for solving the scheduling problem of bidline generation. We used the domain knowledge about the problem to separate the system in two phases (modules) that have clearly drawn responsibilities; the first phase is responsible for building high quality solutions whereas the second uses a GA to complete the partial solution that the first phase produced. We experimented with two different representations in order to understand which search mechanisms are better suited for this problem. The groups representation as expected, gives better results but it is only with the help of the hill climbing techniques after the decoder of the evaluation function that a feasible solution can be located.

The system is in production at Delta Air Lines since 1997 and has consistently produced very good quality solutions in a fraction of the time that the airline used to spend to build schedules before.

References

[1] K. W. Campbell, R. B. Durfee, and G. S. Hines. FedEx generates bid lines using simulated annealing. *Interfaces*, 27(2), 1997.

[2] Y. C. Cheng, D. J. Jr. Houck, J. M. Liu, M. S. Meketon, L. Slutsman, R. J. Vanderbei, and P. Wang. AT&T KORBX system. *AT&T Technical Journal*, 68(3):7–19, 1989.

[3] I. T. Christou and R. R. Meyer. Fast distributed genetic algorithms for paritioning uniform grids. In A. Ferreira, J. Rolim, Y. Saad, and T. Yang, editors, *Lecture Notes in Computer Science 1117*. Springer-Verlag, 1996. Proceedings of the 3rd International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular 96).

[4] I. T. Christou, A. Zakarian, J.M. Liu, and H. Carter. A two phase genetic algorithm for large scale bidline generation problems at Delta. *Interfaces*, 29(5), 1999.

[5] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, New York, NY, 1998.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.

[7] ILOG CPLEX Division, 889 Alder Ave. Incline Village, NV 89451. *Using the CPLEX 6.0 Callable Library*, 1998.

[8] A. I. Z. Jarrah and J. T. Diamond. The problem of generating crew bidlines. *Interfaces*, 27(4), 1997.

[9] J. Kennington and Z. Wang. A shortest augmenting path algorithm for the semi-assignment problem. *Operations Research*, 44(1), 1992.

[10] D. Levine. *A Parallel Genetic Algorithm for the Set Partitioning Problem*. PhD thesis, Illinois Institute of Technology - Dept. of Mathematics and Computer Science, 1994.

[11] Z. Michalewizc. The significance of the evaluation function in evolutionary algorithms. In L. D. Davis, K. De Jong, M. D. Vose, and L. D. Whtiley, editors, *Evolutionary Algorithms*, volume 111 of *The IMA Volumes in Mathematics and its Applications*, pages 151–166, New York, NY, 1999. Springer.

[12] B. Paechter, A. Cumming, H. Luchian, and M. Petriuc. Two solutions to the general timetable problem using evolutionary methods. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 300–305, Los Alamitos, CA, 1994. IEEE Press.

[13] M. D. Vose. What Are Genetic Algorithms? A Mathematical Perspective. In L. D. Davis, K. De Jong, M. D. Vose, and L. D. Whtiley, editors, *Evolutionary Algorithms*, volume 111 of *The IMA Volumes in Mathematics and its Applications*, pages 251–276, New York, NY, 1999. Springer.

[14] D. H. Wolpert and W. G. MacReady. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, 1995.