Real Time Genetic Scheduling of Aircraft Landing Times

Vic Ciesielski, Paul Scerri

Abstract—
Evolutionary approaches are not usually considered for real time scheduling problems due to long computation times and uncertainty about the length of the computation time. We argue that for some kinds of problems, such as optimizing aircraft landing times, genetic algorithms have advantages over other methods as a best solution is always available when needed, and, since the computation is inherently parallel, more processors can be added to get higher quality solutions if necessary. Furthermore, the computation time can be decreased and the quality of the generated schedules increased by seeding the genetic algorithm from a previous population. We have performed a series of experiments on landing data for Sydney airport on the busiest day of the year. Our results show that high quality solutions can be computed in the time window between aircraft landings.

Keywords— **Genetic Algorithms, Scheduling, Evolutionary Algorithms, Optimization, Anytime Algorithm.**

I. Introduction

REAL time problems are characterized by the need to have certain computations completed and answers ready within a limited time otherwise it will be too late for the answers to be of any use. One approach to real time problems is the use of 'anytime algorithms' [1], procedures which can be interrupted at any time and will always have a result available but will produce a better result given more time.

Genetic algorithms[2], [3] are a problem solving strategy, based loosely on Darwinian evolution, that has been successfully used for a large number of scheduling and optimization problems[4], [5]. Genetic algorithms are generally associated with long computation times and great uncertainty about how long a computation will take. Consequently they are not normally considered for real-time problems, such as the optimal scheduling of aircraft landing times. Despite the perceived computational disadvantages, we believe that genetic algorithms might in fact, perform well in real time situations, since:

1. A population of potential solutions (i.e. schedules) is always available. The longer one runs the computation the better these solutions should become. However, at any time, there is always a best solution available. Thus genetic algorithms can be considered as a form of anytime algorithm.
2. Genetic algorithms are inherently parallel. If more computation is needed to get an acceptable result in the time available, one can simply add more processors.

Both authors are with the Department of Computer Science, RMIT University, Melbourne, Australia. E-mail: vc@cs.rmit.edu.au

(Appears in D. Fogel (Editor) *The IEEE International Conference on Evolutionary Computation (ICEC98)*, Pages 360-364, Anchorage, May, 1998)

The scheduling of aircraft arrival times at a busy airport is a difficult problem for many reasons. There is an inherent tradeoff between leaving wide safety margins between aircraft and maximizing the number of aircraft that take off and land each hour, particularly during peak hours. It has been estimated that the problem of air traffic congestion will cost $US10 billion in Europe alone by the year 2000.

A. Goals

Our primary goal is to investigate the applicability of genetic algorithms to the problem of real time scheduling of aircraft arrival times at airports. We are particularly interested in:

1. Whether the constraints and the constantly changing situations such as aircraft landings and new arrivals in the scheduling horizon can be adequately represented with the a genetic algorithm approach.
2. Whether the landing sequences computed in the time available will be good enough.
3. Whether specialized crossover and mutation operators which favour changes to newly arrived planes are better than standard binary crossover and mutation.
4. A deployed scheduling system is required to deliver a new schedule when a plane lands or new planes enter the scheduling horizon. Thus a sequence of schedules need to be generated. We expect that new schedules constructed by seeding the new optimization run with chromosomes from the previous population will be better than those constructed from scratch.

II. Scheduling of Aircraft Arrival Times

The focus of this work on is what happens after planes enter the 'scheduling horizon' of an airport, usually about 40 minutes before landing. The location and landing order of planes is controlled by the local air traffic controllers. When a plane enters the scheduling horizon, it has an optimal landing time. This is the time it would land if there were no other aircraft at the airport and it could travel at its optimal speed. At busy times, planes arrive at a faster rate than can be landed. Slow planes come into the scheduling zone before faster ones. If planes were allocated landing times on a first come first served basis, runways would remain idle as the queue of planes wanting to land increased. The key resource in this domain is time on a runway.

At Sydney airport, for which we have data, landing slots are allocated at 3 minute intervals. This leaves a conservative margin for safety and permits several takeoffs between landings, however it is a major contributor to congestion.

The work reported in this paper is concerned with variable times between landings. Other factors that must be taken into account in generating schedules are:

- Some planes have higher priority than others; a 747 will generally have higher priority than a Cessna.
- While arriving planes have a preferred runway, some planes are constrained to use specific runways.
- There are constraints in the amount of time that must pass before, say, a small plane can land after a big plane.

It is undesirable to change landing times too much after they have been allocated since this causes ripple effects and additional communication in notifying pilots of changes in landing arrangements. Scheduling of take-offs is not considered since these are interspersed with landings. A plane can be asked to land up to 3 minutes earlier than its optimal time to improve a schedule. It is not possible for a plane to land more than three minutes early, that is, more than three minutes before its optimal arrival time..

A closely related problem that has received much attention in the literature is the job shop scheduling problem[6], [7].

III. The Test Data

The data we are using for the problem was generated from a simulation of air traffic for Easter Thursday at the Sydney airport, which is the busiest day of the year. Two data sets were generated, one containing 28 planes arriving in a 37 minute period, the other 29 planes arriving in a 38 minute period.

IV. The Genetic Algorithm

Note that we actually need to solve a sequence of optimization problems. Whenever planes land and new planes enter the scheduling horizon we have a new optimization problem.

We have compared two genetic algorithms:

1. The standard binary genetic algorithm as described in [2] and implemented in the GAUCSD package[8]. In this case each problem in the sequence requires a new chromosome encoding which deletes planes that have landed and adds planes that have recently arrived. The new schedules are built from 'scratch', that is from a random initial population as shown below:

Standard Algorithm
For each 3 min interval do
  Randomly initialize population
  While (time < 3 mins or not(convergence)) do
    Select 2 parents
    Apply crossover operation
    Apply mutation operator
    Insert new children into the population
    Use elitist strategy where the best 10%from the previous generation are copied to next generation

2. A 'seeding' modification. Rather than starting from scratch, we build, or seed, the new initial population from the one left at the end of the last problem by deleting planes that have landed and inserting newly arrived planes as shown below:

Seeding Approach
For each 3 min interval do
  If first 3 min interval
    then Randomly initialize population
    else Seed from final population of previous 3 min interval
  While (time < 3 mins or not(convergence)) do
    Select 2 parents
    Apply crossover operation
    Apply mutation operator
    Insert new children into the population
    Use elitist strategy where the best 10%from the previous generation are copied to next generation

A. Encoding

Each gene on the chromosome consists of 8 bits and represents a landing time (7 bits) and a runway (1 bit) as shown in figure 1. A plane with a zero landing time will be the next one landing ($Time = now$). A non zero landing time is the number of 30 second intervals from *now*. In our data there are two runways so 1 bit is enough to encode the runway.

Time	R	Time	R	Time	R	Time	R	.	.

Fig. 1. Encoding of Chromosomes

Associated with with the chromosomes is a global table of planes as shown in figure 2.

1		2		3		4			
0	0	6	1	9	0			.	.

index	Id	Size	Opt0	Opt1
1	TXZ	Big	12:01	12:04
2	FDS	Small	12:03	12:00
3	JKL	Big	12:01	12:05

Fig. 2. Encoding of Chromosomes: Example

Static information about planes is kept in a global table as shown in figure 2. The 'opt0' column gives the optimal landing time on runway 0 and the 'opt1' column the optimal landing time on runway 1. Assuming that $now = 12:00$, the encoding shown in figure 2 corresponds to a schedule where plane 1 is to land at 12:00:00 on runway 0, plane 2 at 12:03:00 on runway 1 and plane 3 at 12:04:30 on runway 0.

A landing schedule as presented to the user is a table consisting of a time, a runway and the Id of the plane scheduled to land. The schedule must be shown in order of increasing landing times. Producing a landing schedule

from any chromosome simply requires building an empty table of landing slots at 30 second intervals for each runway, walking down the chromosome and using the time and runway to index and fill in this table. This is similar to the 'schedule builder' approach used in job shop scheduling [9]. Note that a considerable number of these 30 second landing slots will be empty.

The advantages of this encoding are that very fine time slices (30 seconds) can be represented and that fast binary crossover and mutation operators can be used. A big disadvantage is that invalid schedules are generated by crossover and mutation, for example two planes can be scheduled to land at the same time on the same runway.

Note that while it may appear that the problem requires a variable length chromosome to deal with an arbitrary number of planes waiting to land, this is in fact not the case since each 30 seconds any planes that have landed in that time are removed from the static table and the chromosomes, and new planes reaching the scheduling area added. This results in a sequence fixed length optimization problems.

The encoding described above was chosen after considerable analysis and experimentation with alternatives. Originally we treated the problem as a rearrangement/permutation problem and used order and position based operators[10]. A pair of genes on the chromosome corresponded to a 3 minute landing slot on each of two runways. Each slot contained the Id of the plane scheduled to land at that time. This encoding, while it is a relatively natural encoding of the problem had a number of disadvantages:

1. It resulted in illegal solutions since schedules were generated in which planes were listed to land earlier than was physically possible. In some runs there was not a single valid schedule in the population.
2. It required long chromosomes, which would need to be 6 times longer to deal with 30 second landing slots.
3. The position based crossover and mutation operators executed very slowly.

B. Fitness Function

We have chosen to deal with invalid solutions by use of the fitness function. Thus the fitness function must punish sub-optimal solutions and severely punish invalid ones. We have used the following penalties:

Fitness = INVALID_PENALTY (If Applicable)
+ CLASH_PENALTY (If Applicable)
*+ No_planes_too_close * TOO_CLOSE_PENALTY*
*+ No_planes_adj_too_close * ADJ_PENALTY*
*+ total_early * EARLY_PENALTY*
*+ total_delay * DELAY_PENALTY*

Fig. 3. Fitness Function

Penalty	Description	Val
INVALID	The schedule is invalid for any reason.	100
CLASH	Two planes scheduled to land at the same time.	10
TOO_CLOSE	Planes are scheduled to land on the same runway without an adequate time gap.	10
ADJ	Planes are scheduled to land on adjacent or crossed runways without an adequate time gap	10
EARLY	A plane scheduled to land too early.	3
DELAY	A plane is scheduled to land too long after its optimal landing time.	1

The fitness function used is shown in figure 3. The value of *total_early* is found by finding all planes which are scheduled to arrive too early, for each such plane finding the amount of time by which it is too early and computing the total for all early planes. *Total_delay* is the result of a similar calculation for planes which are scheduled to land later than their optimal time.

The relative sizes of the penalties can be roughly determined from domain knowledge, for example an invalid schedule must be heavily punished, while a plane landing after a small delay should only receive a small penalty. Determining the actual sizes of the penalties was mostly an empirical exercise.

C. Genetic Operators

We have compared standard binary crossover and mutation operators and modified binary crossover and mutation operators. The modifications are based on domain knowledge: Since we desire that the early parts of schedules do not change very much (a domain requirement) we decrease the probability of a mutation or crossover at the beginning of the chromosomes and increase it the end of the chromosome. Note that, due to the encoding, recently arrived planes are represented toward the end of the chromosome. Thus the operators tend to leave alone planes that have been in the system for some time and focus on newer ar-

rivals.

For our chosen encoding there are no straightforward ways of repairing invalid solutions or modifying the crossover and mutation operators to ensure that illegal solutions cannot arise. We use the fitness function to heavily penalize invalid solutions thus lowering the probability of their being selected for reproduction.

V. Results

The runs were carried out using a modified version of GAUCSD[8] using the elitist (10%) strategy. All runs up to 50,000 trials completed in less than 30 seconds on a SPARC workstation.

In a deployed scheduling system it is absolutely imperative for the system to deliver a valid schedule on demand. The schedule should be optimal or very close to optimal. Thus in our experiments we have focused on the number of valid schedules in the population, the fitness of the schedules, and how they vary with genetic algorithm parameters. Ideally we would like to find the settings which give the best possible results on both measures.

	Special		Standard	
Trials	Seed	No Seed	Seed	No Seed
500	53.17	47.67	56.94	47.20
1,000	53.96	46.55	55.65	49.48
2,000	65.75	48.97	58.95	45.47
5,000	48.10	50.20	55.94	50.12
10,000	62.81	51.61	65.63	55.49
50,000	58.00	44.76	56.79	49.07

TABLE I
Percentage of Valid Solutions. Population = 50, Crossover rate = 0.1, Mutation rate = 0.01

We expected that the number of valid schedules would stay relatively constant and the fitness would increase uniformly with the number of trials, but, as the tables below show, this turned out not to be the case.

Tables I and II show the results of a number of runs whose goal was to determine whether the seeding approach was better than starting from scratch. Table I shows the number of valid schedules in the population as the number of trials increases. The results for seeding are clearly superior. We expected that the runs where seeding was not used would start with a lower number of valid schedules but would eventually reach the performance as when seeding was used. This turned out not to be the case. The non seeding performance never 'catches up' to the seeding performance. The same result is evident in the fitness of the best solutions as shown table II.

Tables I and II also show comparisons of results with our special crossover and mutation operators and the standard binary ones. Comparison of columns 2 and 4 and 3 and 5 of table I, and 2 and 4 and 3 and 5 of table II reveals that there is no real difference.

Tables III and IV were generated only for the *seeding* approach. They show how the number of valid solutions changes with population size, mutation and crossover rates. Results for both the special and standard crossover and mutation operators are shown.

	Special		Standard	
Trials	Seed	No Seed	Seed	No Seed
500	289.46	348.37	228.06	370.74
1,000	268.58	368.12	233.59	312.12
2,000	159.57	327.60	183.34	335.16
5,000	297.42	327.71	243.49	313.44
10,000	166.74	261.28	167.84	206.54
50,000	278.19	373.07	238.42	321.81

TABLE II
Best Fitness. Population = 50, Crossover rate = 0.1, Mutation rate = 0.01

Special xover and mutation					
Run	Pop	Xover Rate	Mut Rate	% Valid 2,000 Trials	% Valid 20,000 Trials
1	500	0.1	0.1	66.54	67.24
2	500	0.6	0.001	64.82	63.11
3	50	0.6	0.01	52.68	45.88
4	50	0.1	0.1	57.38	55.85
5	50	0.1	0.01	65.75	50.51

Standard xover and mutation					
Run	Pop	Xover Rate	Mut Rate	% Valid 2,000 Trials	% Valid 20,000 Trials
1	500	0.1	0.1	55.59	58.38
2	500	0.6	0.001	67.72	68.77
3	50	0.6	0.01	58.29	60.01
4	50	0.1	0.1	58.04	57.26
5	50	0.1	0.01	58.95	62.47

TABLE III
Percent of valid solutions at for different GA parameter values

The best results for both number of valid solutions and fitness were obtained for a high population size (500), a high crossover rate (0.6) and a very low mutation rate (0.001) after 20,000 trials and using the standard binary crossover and mutation operators. In general better results are associated with a high population and a low mutation rate. There is no clear superiority of the special crossover and mutation operators over the standard ones or vice versa.

As notes earlier we expected the fitness of the schedules to increase as the number of trials increased. As can be seen from the results its behaviour was somewhat erratic. Further investigation is needed to determine the reason for this.

Special xover and mutation					
Run	Pop	Xover Rate	Mut Rate	% Valid 2,000 Trials	% Valid 20,000 Trials
1	500	0.1	0.1	162.33	157.85
2	500	0.6	0.001	150.31	137.84
3	50	0.6	0.01	284.19	310.44
4	50	0.1	0.1	271.24	250.81
5	50	0.1	0.01	159.57	309.33

Standard xover and mutation					
Run	Pop	Xover Rate	Mut Rate	% Valid 2,000 Trials	% Valid 20,000 Trials
1	500	0.1	0.1	212.27	177.97
2	500	0.6	0.001	147.49	132.97
3	50	0.6	0.01	269.96	267.65
4	50	0.1	0.1	222.00	223.09
5	50	0.1	0.01	183.34	280.12

TABLE IV
FITNESS OF BEST SOLUTION FOR DIFFERENT GA PARAMETER VALUES

VI. Conclusions

The major goal of this work was to determine whether a scheduling system based on genetic algorithms could be used for scheduling aircraft landing times. We have established that a genetic scheduler can generate good schedules in the time available between landings. Thorough inspection and validation of the generated schedules by domain experts has not yet been done, but preliminary indications are that the schedules are of high quality and could be used at Sydney airport.

The application involves a sequence of scheduling problems since new schedules must be generated when planes land and new planes enter the scheduling horizon. Seeding new optimization runs from the chromosomes available at the end of the previous problem has turned out to be very satisfactory. Seeding leads to improved the fitness and a higher number of valid solutions in the population.

Domain knowledge suggested that specialized crossover and mutation operators which favoured newly arrived planes at the end of the chromosome might give improved performance. However they did not result in any real improvement over the standard operators.

We have established that genetic algorithms can produce high quality schedules in real time. The next stage in the development of a scheduling system based on genetic algorithms is to investigate whether the air traffic controllers' preferences, special requirements and exceptions can be adequately captured in the fitness function.

Acknowledgements

We thank Rick Evertsz from the Australian Artificial Intelligence Institute for providing the the data for the project and discussions related to the meaning of the data and general constraints and problems of aircraft arrival scheduling. We thank Dr. Linda Stern from the University of Melbourne for providing us with a position based crossover program for GAUCSD, and for key discussions on the encoding options. We also thank Glen Stevens for the early work done on the permutation based approach to the problem.

References

[1] S. J. Russell and S. Zilberstein, "Composing real-time systems," in *Proceedings of the Twelfth International Conference on Artificial Intelligence*, pp. 212–217, Morgan Kaufman, 1991.

[2] Goldberg, "Genetic and evolutionary algorithms come of age," *Communications of the CACM*, vol. 37, 1994.

[3] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*. Artificial Intelligence, New York: Springer-Verlag, 1992.

[4] P. S. Gabbert, D. E. Brown, C. L. Huntley, B. P. Markowicz, and D. E. Sappington, "A system for learning routes and schedules with genetic algorithms," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, (University of California, San Diego), pp. 430–436, 1991.

[5] G. Syswerda and J. Palmucci, "The application of genetic algorithms to resource scheduling," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, (University of California, San Diego), pp. 502–508, 1991.

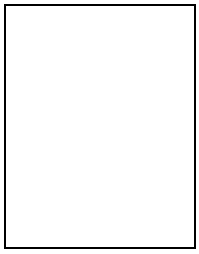
[6] H. L. Fang, P. Ross, and D. Corne, "A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, (University of California, San Diego), pp. 375–382, 1993.

[7] S. Uckun, S. Bagchi, K. Kawamura, and Y. Miyabe, "Managing genetic search in job shop scheduling," *IEEE Expert*, vol. 8, pp. 15–24, Oct. 1993.

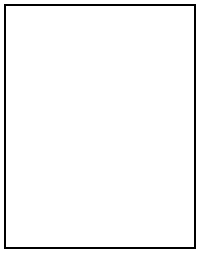
[8] J. J. Grefenstette and N. N. Schraudolph, *A User's Guide to GAUCSD 1.4*. Computer Science & Engineering Department, University of California, July 1992.

[9] G. Syswerda, "Schedule optimization using genetic algorithms," in *Handbook of Genetic Algorithms* (L. Davis, ed.), Van Nostrand Reingold, 1991.

[10] G. Stevens, *An Approach to Scheduling Aircraft Landing Times Using Genetic Algorithms*. Honours thesis, RMIT, Department of Computer Science, Nov. 1995.



Dr. Victor Ciesielski received his B. Sc. and M. Sc. degrees from the University of Melbourne (Australia) and his Ph. D. from Rutgers University. He is currently senior lecturer in the Department of computer science at the Royal Melbourne Institute of Technology. Dr. Ciesielski's research interests include applications of artificial intelligence and evolutionary computation, neural networks and and the integration of symbolic and neural computation. He has developed knowledge based and neural network systems in a number of diverse areas including insurance underwriting, geomechanics, medicine, law and environment conservation. He has published over 50 papers in these areas.



Paul Scerri Paul Scerri has been a student and research assistant in the Department of Computer Science at RMIT University. He is currently a post graduate student at Linkoping University in Sweden.