

A Study on the Feasibility of Using a Genetic Algorithm to Generate Realistic Air Traffic Scenarios Based on Recorded Field Data

*Robert D. Oaks**

*FAA William J. Hughes Technical Center
Atlantic City International Airport, NJ, 08405*

Abstract

Air traffic scenarios based on recorded live data are used for the development and analysis of decision support tools used by air traffic controllers. Frequently these scenarios need to be modified in order to create aircraft-to-aircraft encounters that are not present in the live data. This paper shows that a genetic algorithm can be used to time shift the flights in a scenario in order to create encounters with specific constrained characteristics. For this study these constraints were the distributions of the closest point of approach and the encounter angle of the encounters. The paper first describes how a genetic algorithm was implemented and then presents the results of two series of tests. The first series of tests were designed to determine if the implementation of a genetic algorithm could solve the problem. The second series of tests were designed to assess the time it took the implementation to solve the problem. The results of the study showed that a genetic algorithm could solve the problem in a reasonable time.

Introduction

Traffic flow management decision support tools such as the User Request Evaluation Tool (URET), developed by the MITRE Center for Advanced Aviation Systems Development, and the Center-TRACON Automation System (CTAS), developed by the National Aeronautics and Space Administration/Ames Research Center, use simulation as a tool for development, technical assessment, and field evaluation.

The air traffic scenarios, used by these simulations, are data files describing the flow of aircraft traffic through an airspace over a period of time. For traffic flow management decision support tools, these airspaces are generally those defined for Terminal Radar Approach Control facilities (TRACONs), such as those that manage arrivals and departures around New York and Dallas/Fort Worth, and Air Route Traffic Control Centers (ARTCCs), that manage air traffic as it crosses the country. The scenario data files contain planning/advisory information and track data. The time-stamped planning/advisory data describe the aircraft's planned flight; it includes its flight plan and flight plan amendments, interim altitude clearances, and hold information. The track data represents the aircraft's actual flight path. It consists of a series of 4-dimensional components: two defining the aircraft's position (using either

* Senior Systems Analyst, Signal Corporation, ACB-330 Simulation and Modeling Group, FAA William J. Hughes Technical Center, Atlantic City International Airport, NJ, 08405; E-mail: rdoaks@acm.org; Member, AIAA.

XY-coordinate or latitude/longitude pair); another component defining its altitude; and the fourth component being an associated time.

One form of these scenario files is the playback scenario, which is based on the format of recorded intercomputer messages. Playback scenarios composed of test data specifically designed to test its internal algorithms are used to develop the decision support tools. But in order to ensure their operational capability, the tools are tested with scenarios based on recorded field data containing the situations they are designed to protect against.

Playback scenarios are also used to evaluate decision support tools because they provide the most realistic simulation environments. For example, the FAA's Engineering and Integration Services Branch (ACT-250) at the William J. Hughes Technical Center in Atlantic City, New Jersey, has used recorded field data in two recent studies. Their URET Conflict Prediction Accuracy Study¹ used simulation scenarios based on field data recorded at the Indianapolis ARTCC and their URET/CTAS Trajectory Prediction Accuracy Study² used playback scenarios based on field data recorded at the Indianapolis ARTCC for URET and Fort Worth ARTCC for CTAS. In both of these studies the playback scenarios were generated directly from the recorded field data.

However, frequently it is necessary to modify the scenarios. For example, The FAA's Free Flight Phase One Program Office tasked ACT-250 to develop the scenarios used to verify the accuracy requirements of the URET Core Capability Limited Deployment (CCLD), the operational implementation of the URET. For this effort, the recorded field data was modified to induce aircraft-to-aircraft and aircraft-to-airspace encounters that did not

exist in the recorded data. ACT-250 did this by changing the start times of the aircraft flights through time-shifting.³ This created the necessary number of encounters between the aircraft while retaining the actual routes and profiles the aircraft originally flew.

While it would have been possible to generate scenarios meeting the desired constraints through trial and error, it was desirable to calculate these changes algorithmically. While researching various random search techniques, it was determined that a genetic algorithm (GA) might be applicable. The following quote on the use of GAs in general was found to be appropriate:

. . . if the space to be searched is large, is known not to be perfectly smooth and unimodal (i.e. consists of a single smooth 'hill'), or is not well understood, or if the fitness function is noisy, and if the task does not require a global optimum – i. e., if quickly finding a sufficiently good solution is enough – a GA will have a chance of being competitive with or surpassing other 'weak' methods (methods that do not use domain-specific knowledge in their search procedure)⁴

Genetic Algorithm Implementation

Genetic algorithms (GAs) were invented by John Holland at the University of Michigan in the 1960s and 1970s. They are considered the most prominent example of evolutionary programming. Comprehensive information regarding the history, study, and application of GAs can be found in the literature.^{4,5,6}

GAs are a class of algorithms that derive their behavior from a metaphor of the processes of evolution. As such, there is no specific GA; instead GAs are more of an approach to solving a problem. All GA approaches have the following traits in common: a population

of chromosomes, a fitness function, selection according to fitness, crossover to create new offspring, and mutation.

For this study, a GA was implemented in a program named Cat.[†] This program generates a set of delta times that can be used to time shift the flights in a scenario, inducing encounters meeting specific user-specified constraints.

To find aircraft-to-aircraft encounters, Cat compares each flight's track points with the track points from each of the other flights. An encounter is defined to exist between two flights when the closest point of approach (CPA) between the flights is less than 20 nautical miles in the horizontal plane and less than either 1000 or 2000 feet in the vertical axis, depending on the aircraft's altitude. In addition to the distance at CPA, an encounter has an encounter angle (EA) defined at the CPA. This angle is defined as 180° when the aircraft are flying directly toward each other and 0° when they are in-trail. Cat tallies these encounter occurrences in the ten constraint bins defined in Tables 1 and 2.

Cat uses a genetic algorithm to evolve a set of possible solutions so that the number of aircraft-to-aircraft encounters in each of these bins falls between input bounds.

[†] Cat was developed using gcc, the GNU C/C++ Version 2.7.2.3 compiler, and libg+, the GNU C/C++ Version 2.7.2 libraries. Cat was implemented on a Sun Ultra ES-4500 400 megahertz workstation using the Solaris Version 2.6 operating system. Cat was named for the character Cat on the British television series Red Dwarf. Cat is a humanized feline; the result of 3,000,000 years of evolution on the space ship Red Dwarf after all but one of its crew were killed by a radiation leak.

Table 1: CPA Constraint Bins

Bin #	Constraint
1	0 nm ≤ CPA < 5 nm
2	5 nm ≤ CPA < 10 nm
3	10 nm ≤ CPA < 15 nm
4	15 nm ≤ CPA < 20 nm

Table 2: EA Constraint Bins

Bin #	Constraint
5	0 deg ≤ EA < 30 deg
6	30 deg ≤ EA < 60 deg
7	60 deg ≤ EA < 90 deg
8	90 deg ≤ EA < 120 deg
9	120 deg ≤ EA < 150 deg
10	150 deg ≤ EA < 180 deg

Definition of the Chromosome Population

In a GA, a chromosome is defined as an array of bits or characters that represent a potential solution to a problem. These bits or characters are defined as the chromosome's genes. The values these genes can assume are defined as alleles. A population of these chromosomes is a subset of all solutions to the problem.

In the program Cat, a chromosome is defined to be a sequence of delta times (genes) represented by the tuple $\langle \Delta t_1, \Delta t_2, \dots, \Delta t_n \rangle$, where there is a delta time associated with each flight. Therefore the number of genes in a chromosome is equal to the number of flights in the scenario. These delta times represent the time (in tens of seconds) that the start time of a flight is to be modified to start earlier than its original start time. For example, the chromosome $\langle 0, 75, 9, \dots \rangle$ means to start the first flight at its original time, to start the second flight 750 seconds earlier than its original start time, to start the third flight 90 seconds earlier, etc. Each gene is restricted to assume 360 values (alleles) (i.e., 0, 1, 2, ..., 359) thereby restricting the amount of time a flight could be shifted earlier in time to one hour.

The size of the population of chromosomes in Cat is an input parameter. The initial population consists of one chromosome that

represents the start times of the original field data; i.e., the chromosome $\langle 0, 0, 0, \dots \rangle$. The remaining chromosomes in the initial population have delta times selected randomly from a uniform distribution in the interval 0 to 359.

Definition of the Fitness Function

The fitness function in a GA produces a score for each chromosome, which is a measure of how well the chromosome solves the particular problem. The fitness of a population may be defined either as the average of all of the fitnesses of the population's chromosomes or as the best of the fitnesses of the chromosomes in the population. The goal of the GA is to evolve its population until its fitness reaches some desired value.

The goal of the program Cat is to find a chromosome that results in tallied encounter counts that fall between desired bounds in each of the ten constraint bins defined in Tables 1 and 2. These bounds are input to Cat in the form of ten pairs of constraint bounds – a low bound and a high bound for each of the ten constraint bins.

The fitness function implemented in Cat rewards both individual and multiple instances where tallied counts fall within these constraint bounds; while at the same time penalizing instances in which the measured value is below the low bound or above the high bound.

The fitness function defined for Cat is:

$$F = \frac{2^X}{2^{10}} \quad (1)$$

where

$$X = \sum_{i=1}^{10} x_i \quad (2)$$

The calculation for x_i depends on three conditions.

1. If the tallied count for a bin lies below the lower bound (i.e., if $count_i < lobound_i$) then

$$x_i = \left(\frac{count_i}{lobound_i} \right)^2 \quad (3)$$

2. If the tallied count for a bin falls within the lower and upper bounds (i.e., if $lobound_i \leq count_i < hibound_i$) then

$$x_i = 1 \quad (4)$$

3. If the tallied count falls above the upper bound (i.e., if $hibound_i < count_i$) then

$$x_i = \left(\frac{lobound_i + hibound_i - count_i}{lobound_i} \right)^2 \quad (5)$$

For each of these equations, $lobound_i$ refers to constraint bin i 's low bound, $hibound_i$ refers to its high bound, and $count_i$ refers to the number of encounters tallied to be in the constraint bin.

The value of x_i provides a value of 1.0 if the tallied count is between the low and high bounds and rapidly decreases to 0.0 as the count gets further away from either of the bounds. This is seen in Figure 1, which is a plot showing the individual contribution (x_i) as a function of the measured count ($count_i$). The two vertical dashed lines represent the bin's low bound and upper bound.

Once the individual values of x_i are computed for each of the ten constraint bins, X is computed as their floating-point sum as defined in Equation (2). This results in X being a weighted count of the number of bins in which the constraints have been satisfied.

The fitness (F) is then calculated as defined in Equation (1). Figure 2 is a plot of how fitness (F) varies as a function of the sum of the individual contributions (X). This plot shows how the fitness exponentially approaches the value of 1.0 as the number of satisfied constraint bins approaches 10.0.

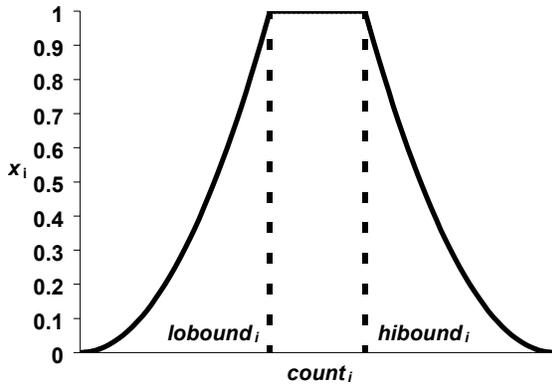


Figure 1: Constraint Bin Contribution to Fitness

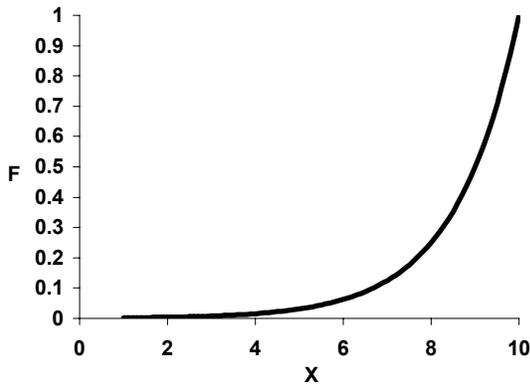


Figure 2: Graph of the Fitness Function

Definition of the Selection Process

The first step in evolving a new generation of chromosomes is to select parent chromosomes from the current population. In most GAs the parent chromosomes are selected from the population with the probability of selection being directly proportional to a chromosome's fitness.

In the program Cat, the number of parent chromosomes selected in Cat is the same as the number of chromosomes in the population. Cat uses "stochastic universal sampling" as its selection technique. This technique selects chromosomes based on each chromosome's expected representation in the selected population. The expected representation is defined as the ratio of the individual's chromosome's fitness and the sum of the fitnesses in the population. Unlike purely random selection techniques, this technique ensures that fit chromosomes are not statistically lost in the selection process.^{4,6}

Definition of the Crossover Technique

The next step in evolving a new generation of chromosomes is to create an offspring population from the selected parent chromosomes based on crossover. Generally crossover occurs according to a probability called the probability of crossover. When it occurs the genes of paired parent chromosomes are combined according to some rule to create new chromosomes. In most implementations, this is done by swapping the parent's genes at selected locus positions. If crossover does not occur the selected parent's chromosomes become the offspring chromosomes.

In Cat, the occurrence of crossover depends on an input parameter P_c , which is the probability of crossover. When crossover occurs, Cat swaps the genes between two parent chromosomes using two randomly selected points. This reduces positional bias⁴.

Definition of the Mutation Technique

The final step in evolving a new generation of chromosomes is mutation. Each of the genes in the chromosomes of the offspring population is considered for mutation according to a probability of mutation. When mutation occurs the gene is randomly changed to another valid value (allele).

For mutation, Cat replaces a gene with a delta time selected randomly from a uniform distribution in the interval 0 to 359. As a result, each flight can be shifted to start randomly up to one hour prior earlier in time. The occurrence of mutation in Cat depends on an input parameter P_m , which is the probability of mutation.

Results

Two series of test runs were made to determine if Cat can effectively calculate a set of delta times that can be used to modify a scenario so it will have user-specified distribution of encounters. The first, called the solvability tests, were to determine if the program could solve the problem. The second, called the time-to-solve tests, were to analyze how long it took Cat to repetitively solve a single problem.

All of these test runs were made using one hour of interpolated track data derived from field data recorded at the Indianapolis ARTCC. This recorded data consists of 493 flights with 344 aircraft-to-aircraft encounters. The distributions of these baseline encounters for closest point of approach (CPA) and for encounter angle (EA) are shown in Table 3.[‡]

[‡] It important to note that the 20 encounters in Bin #1 of Table 3, which had a CPA less than 5 nm are not ATC conflicts, since not all of the rules defining an ATC conflict were taken into account.

Table 3: Original Encounter Counts

Bin #	Constraint	Original Count
1	$0 \leq \text{CPA} < 5$	20
2	$5 \leq \text{CPA} < 10$	68
3	$10 \leq \text{CPA} < 15$	119
4	$15 \leq \text{CPA} < 20$	137
5	$0 \leq \text{EA} < 30$	83
6	$30 \leq \text{EA} < 60$	44
7	$60 \leq \text{EA} < 90$	51
8	$90 \leq \text{EA} < 120$	51
9	$120 \leq \text{EA} < 150$	58
10	$150 \leq \text{EA} < 180$	57

Solvability Tests

The solvability tests consisted of three runs. The first run attempted to evenly distribute the 344 original encounters evenly across both the CPA and the EA constraint bins. To accomplish this Cat's input parameters specified 85 to 87 encounters in each of the CPA bins (Bins #1-4) and 56 to 58 encounters in each of the EA bins (Bins #5-10). The input also specified a high value of 0.99 for the desired fitness of the solution. This meant the program would terminate with a solution that comes close to meeting the constraints. This was done to ensure the program would terminate in a reasonable amount of time.

The results of this first run are shown in Table 4, where it is seen Cat generated a set of delta times for which the requested encounter counts fell within the constraint bounds of all of the bins – i.e., this solution had a fitness value of 1.0. Cat generated this solution in 173 generations using 2955.73 seconds of CPU time. This solution contains 344 encounters, but these are not the same 344 encounters that occurred in the baseline data because the time modification results in different encounters.

Table 4: Constraint Bins for the First Run

Bin #	Constraint	low bound	high bound	Count
1	$0 \leq CPA < 5$	85	87	85
2	$5 \leq CPA < 10$	85	87	85
3	$10 \leq CPA < 15$	85	87	87
4	$15 \leq CPA < 20$	85	87	87
5	$0 \leq EA < 30$	56	58	58
6	$30 \leq EA < 60$	56	58	58
7	$60 \leq EA < 90$	56	58	56
8	$90 \leq EA < 120$	56	58	57
9	$120 \leq EA < 150$	56	58	58
10	$150 \leq EA < 180$	56	58	57

Figure 3 is a plot showing how the fitness of the population increased with the evolving generations of the chromosome population. The thicker line in the plot represents the best fitness encountered up to that generation. The narrower line in the plot represents the average fitness of the chromosomes in that generation.

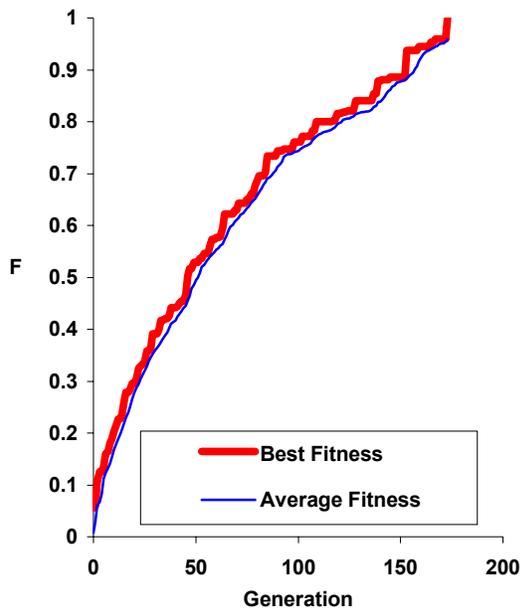


Figure 3: Fitness vs. Generation

In the second solvability test Cat was asked to increase the number of encounters. In addition, the desired fitness was lowered to 0.975. Cat generated the results shown in Table 5 in 172 generations with a best fitness of 0.978738. This solution contains 388

encounters with the count in one bin (Bin #9) not falling within the desired bounds.

Table 5: Constraint Bins in Second Run

Bin #	Constraint	low bound	high bound	Count
1	$0 \leq CPA < 5$	22	24	24
2	$5 \leq CPA < 10$	75	82	77
3	$10 \leq CPA < 15$	131	143	133
4	$15 \leq CPA < 20$	151	164	154
5	$0 \leq EA < 30$	91	100	91
6	$30 \leq EA < 60$	48	53	51
7	$60 \leq EA < 90$	56	61	60
8	$90 \leq EA < 120$	56	61	57
9	$120 \leq EA < 150$	64	70	63
10	$150 \leq EA < 180$	63	68	66

In the third solvability test Cat was required to increase the number of encounters and to distribute the encounters evenly across both the CPA and EA constraint bins. Table 6 presents the results of this run. The fitness for this solution was 0.985589. It contains 389 encounters with the count in one bin (Bin #1) not falling within the desired bounds. This solution was generated in 242 generations using 4235.17 seconds of CPU time.

Table 6: Constraint Bins in Third Run

Bin #	Constraint	low bound	high bound	Count
1	$0 \leq CPA < 5$	95	105	94
2	$5 \leq CPA < 10$	95	105	95
3	$10 \leq CPA < 15$	95	105	96
4	$15 \leq CPA < 20$	95	105	104
5	$0 \leq EA < 30$	60	65	65
6	$30 \leq EA < 60$	60	65	65
7	$60 \leq EA < 90$	60	65	64
8	$90 \leq EA < 120$	60	65	65
9	$120 \leq EA < 150$	60	65	65
10	$150 \leq EA < 180$	60	65	65

Time-to-Solve Tests

Ten runs were made to analyze the time required by Cat to generate an acceptable fitness given the same input parameters, but using a different seed value for the random number generator.

Cat was able to generate a solution in each of these ten runs and in one run generated two chromosomes, each having equally fit

solutions. Since the GA is a stochastic process each of the ten runs produced a different result and used a different amount of CPU processing time. Figure 4 is a plot of the best fitness value as a function of elapsed CPU processing time for each of the ten runs.

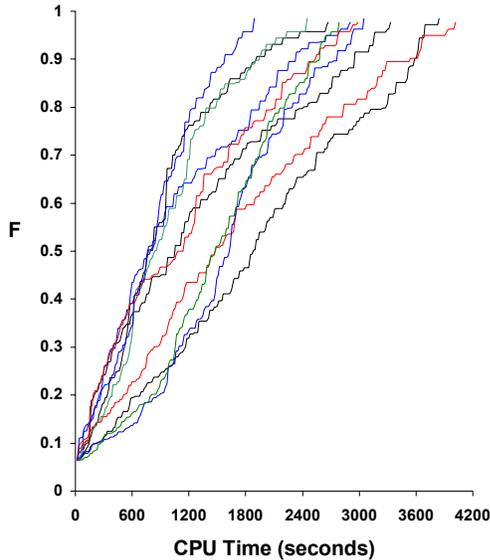


Figure 4: Best Fitness vs. CPU Processing Time

Table 7 shows the time required to generate a solution in each of the ten runs. These results show that a solution was generated in as little as 1890.69 seconds and as much as 4017.10 seconds. The average time-to-solve was 2993.74 seconds with a standard deviation of 628.17 seconds.

Table 7: Time to Solve in Test Runs

Run Number	Time-to-solve (CPU seconds)
0	3331.14
1	3844.34
2	2787.15
3	2673.55
4	2907.13
5	4017.10
6	3051.04
7	2449.61
8	2986.64
9	1890.69

Acknowledgments

The data used for this study was provided by the Conflict Probe Assessment Team, which is a part of the FAA's Engineering and Integration Branch (ACT-250) at the William J. Hughes Technical Center. The author thanks the members of that team for their support and help for this study.

Bibliography

1. Cale, Mary Lee, Paglione, Michael, Ryan, Dr. Hollis, Timoteo, Dominic, Oaks, Robert, User Request Evaluation Tool (URET) Conflict Prediction Accuracy Report, DOT/FAA/CT-TN98/8, WJHTC/ACT-250, April, 1999.
2. Paglione, M. M., Ryan, H. R., Oaks, R. D., Summerill, J. S., Cale, M. L., Trajectory Prediction Accuracy Report User Request Evaluation Tool (URET) / Center-TRACON Automation System (CTAS), DOT/FAA/CT-TN99/10, FAA ACT-250, May 1998.
3. Oaks, Robert D., Paglione, Mike, *Generation of Realistic Air Traffic Scenarios Based on Recorded Field Data*, Proceedings of the 46th Annual Meeting, Air Traffic Control Association, Washington, DC, November 4-8, 2001.
4. Mitchell, Melanie, *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, MA, 1998.
5. Goldberg, David E., *Genetic Algorithms in Searching, Optimization, and Machine Learning*, Addison-Wesley, 1989.
6. Michalewicz, Zbigniew, *Genetic Algorithms + Data Structures = Evolutionary Programs*, Third, Revised and Extended Edition, Springer, 1996.