

# **Design and Performance of an Improved Genetic Algorithm Implementation for Time-Shifted Air Traffic Scenario Generation**

James A. Ritchie III; FAA ANG-C55

Andrew J. Fabian; FAA ANG-C55

Christina M. Young, PhD; FAA ANG-C55

Mike Paglione; FAA ANG-C55

January 2016

DOT/FAA/TC-TN16/3

Document is available to the public through the National Technical Information Service, Springfield, Virginia 22161



U.S. Department of Transportation  
**Federal Aviation Administration**

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

[THIS PAGE IS INTENTIONALLY LEFT BLANK]

## **NOTICE**

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof. The United States Government does not endorse products or manufacturers. Trade or manufacturer's names appear herein solely because they are considered essential to the objective of this report. This document does not constitute FAA certification policy. Consult your local FAA aircraft certification office as to its use.

This report is available at the Federal Aviation Administration William J. Hughes Technical Center's Full-Text Technical Reports page: [actlibrary.tc.faa.gov](http://actlibrary.tc.faa.gov) in Adobe Acrobat portable document format (PDF).

[THIS PAGE IS INTENTIONALLY LEFT BLANK]

<b>1. Report No.</b> DOT/FAA/TC-TN16/3		<b>2. Government Accession No.</b>		<b>3. Recipient's Catalog No.</b>	
<b>4. Title and Subtitle</b> Design and Performance of an Improved Genetic Algorithm Implementation for Time-Shifted Air Traffic Scenario Generation				<b>5. Report Date</b> January 2016	
				<b>6. Performing Organization Code</b> ANG-C55	
<b>7. Author(s)</b> James A. Ritchie III, Christina M. Young, Andrew J. Fabian, Mike Paglione				<b>8. Performing Organization Report No.</b> DOT/FAA/TC-TN16/3	
<b>9. Performing Organization Name and Address</b> U. S. Department of Transportation Federal Aviation Administration, William J. Hughes Technical Center Atlantic City International Airport, NJ 08405				<b>10. Work Unit No. (TRAIS)</b>	
				<b>11. Contract or Grant No.</b>	
<b>12. Sponsoring Agency Name and Address</b> U. S. Department of Transportation, Federal Aviation Administration NextGen Technology Development & Prototyping Division Washington, DC 20590				<b>13. Type of Report and Period Covered</b> Technical Note	
				<b>14. Sponsoring Agency Code</b> DOT	
<b>15. Supplementary Notes</b> The authors identified above represent the following organizations: James A. Ritchie III, Christina M. Young, Andrew J. Fabian, and Mike Paglione with FAA ANG-C55.					
<b>16. Abstract</b> This technical note documents the implementation of a Genetic Algorithm (GA) designed to generate time-shifted flight scenarios for the purpose of Conflict Probe (CP) evaluation. The history of generating time-shifted flight scenarios is discussed. An improved implementation called JavaCat is explained in detail. Two experiments to test the performance of the program are described. The goal of the experiments is to determine a set of parameters that will minimize the run time of the program as well as the number of generations required to create an acceptable conflict-filled scenario. The selected air traffic scenarios include flights from one busy, low delay day in two Air Route Traffic Control Centers (ARTCCs): Chicago (ZAU) and Denver (ZDV). Statistical software is used as part of the experimental design process as well as to process the resulting data. Three iterations of 20 runs are conducted in order to test different input combinations to the program, including population size, number of computers or "islands", and the migration rate of the data from the "islands". High variance is found in the results of the first experiment, so a second experiment is conducted in order to determine the effects of one specific parameter, the number of islands. Nine replications of three runs are conducted, and the results analyzed with statistical software, as in the first experiment. The improved implementation combined with the knowledge gained from these experiments shall provide the FAA a valuable tool in generating future air traffic scenarios for testing air traffic control automation.					
<b>17. Key Words</b> genetic algorithm, evolutionary algorithm, time-shifting, air traffic scenario, conflict probe			<b>18. Distribution Statement</b> This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161. This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at <a href="http://actlibrary.tc.faa.gov">actlibrary.tc.faa.gov</a> .		
<b>19. Security Classif. (of this report)</b> Unclassified		<b>20. Security Classif. (of this page)</b> Unclassified		<b>21. No. of Pages</b> 45	<b>22. Price</b>
Form DOT F 1700.7 (8-72)		Reproduction of completed page authorized			

[THIS PAGE IS INTENTIONALLY LEFT BLANK]

## **Acknowledgements**

The authors would like to thank Robert Oaks and Brian Schnitzer of General Dynamics Information Technology (GDIT) for providing support, general guidance, and valuable feedback while creating this document.

Ron Wilkinson, GDIT, provided technical support throughout the experiments.

The development of JavaCat would not have been possible without the support of the rest of the ANG-C55 software development team: Andrew Tasso, FAA; Byron Hoy and Ian Wilson, CSSI; and Michael Bevilacqua, (formerly) CSSI.

## Executive Summary

The Modeling and Simulation Branch of the Federal Aviation Administration (FAA) conducts research to assess the operational and technical feasibility of proposed system changes to National Airspace System (NAS) operations. The branch supports several initiatives under the Next Generation Air Transportation System (NextGen) with the goal of improving safety and efficiency in the NAS. A key area of research is evaluation of ground automation supporting air traffic controllers, including conflict probes (CP). Effective testing of a CP requires conflicts between flights in a given air traffic scenario, which are generally not found in recorded traffic where flights have been maneuvered to maintain separation standards. However, the branch has been successful in manipulating recorded air traffic scenarios to induce conflicts.

To generate scenarios for CP testing, the FAA has developed a method of time-shifting. In this approach, flights in previously recorded data are shifted in time, by random amounts, in order to induce conflicts. The method of finding these time-shifts is implemented as a genetic algorithm, which is a search heuristic that mimics the processes of biological evolution. An early implementation, named Cat, uses theories of survival of the fittest, to find the time-shift values that would create a scenario with the desired event characteristics.

A new implementation, JavaCat, is developed using object-oriented design techniques. These techniques make the program more extensible and maintainable, allowing for additional genetic algorithm strategies, like the island model, to be implemented very easily. The island model implementation benefitted from this choice by allowing more island model migration strategies to be introduced more easily in the future. The program also makes use of well-established internal libraries for evaluating the scenarios, making the program more accurate in its examination of potential solutions.

Two experiments are performed to determine how the different parameters for the island model and GA affect the performance of JavaCat. The first experiment tests population size, number of islands, and migration period, using recorded air traffic scenarios from two different Air Route Traffic Control Centers (ARTCC). Results indicate that population size has a significant effect on the number of generations and duration of the GA to find an acceptable solution; however, high run-to-run variance is observed in the data. A second experiment is developed in order to evaluate one specific factor, the number of islands, with more replications and tighter control of computer resources. This experiment demonstrates that both six and nine islands typically perform better on average than three islands; however, there is no significant difference in performance between six and nine islands. Both experiments provide valuable information on the effect of different factors on the performance of JavaCat. JavaCat's improved design combined with the knowledge gained from these experiments will provide the FAA with a valuable tool in generating future air traffic scenarios for testing air traffic control automation.

# Table of Contents

<b>1. INTRODUCTION</b> .....	<b>1</b>
<b>2. PRIOR WORK IN AIR TRAFFIC SCENARIO GENERATION</b> .....	<b>2</b>
2.1 GENETIC ALGORITHM IMPLEMENTATION – CAT .....	2
2.1.1 <i>Basic GA Structure</i> .....	2
2.1.2 <i>Reproduction Process</i> .....	3
2.1.3 <i>Evaluation</i> .....	9
2.2 ISLAND MODEL AND JAGUAR .....	11
<b>3. NEW IMPLEMENTATION: JAVACAT</b> .....	<b>12</b>
3.1 CHANGES TO BASIC GA STRUCTURE .....	12
3.2 CHANGES TO REPRODUCTION .....	12
3.2.1 <i>Selection</i> .....	12
3.2.2 <i>Mating</i> .....	13
3.2.3 <i>Crossover and Mutation</i> .....	13
3.2.4 <i>Elitism</i> .....	13
3.3 CHANGES TO EVALUATION .....	14
3.3.1 <i>Evaluation Tools</i> .....	14
3.3.2 <i>Calculating Constraint Fitness</i> .....	14
3.4 POPULATION UPDATES OUTSIDE OF REPRODUCTION .....	17
<b>4. ORIGINAL EXPERIMENT</b> .....	<b>19</b>
4.1 APPROACH .....	19
4.1.1 <i>Experimental Design</i> .....	19
4.1.2 <i>Scenarios</i> .....	20
4.2 RESULTS .....	22
4.2.1 <i>Model</i> .....	22
4.2.2 <i>Response Variables</i> .....	22
4.2.3 <i>Analysis</i> .....	24
<b>5. SUPPLEMENTAL EXPERIMENT</b> .....	<b>27</b>
5.1 APPROACH .....	27
5.2 RESULTS .....	27
5.2.1 <i>Model</i> .....	27
5.2.2 <i>Response Variables</i> .....	28
5.2.3 <i>Analysis</i> .....	28
<b>6. CONCLUSIONS</b> .....	<b>31</b>
<b>7. RECOMMENDATIONS AND FUTURE WORK</b> .....	<b>32</b>
<b>8. REFERENCES</b> .....	<b>33</b>
<b>APPENDIX</b> .....	<b>35</b>

## List of Figures

Figure 1. Pseudocode for a generic genetic algorithm .....	3
Figure 2. Flow chart of Cat GA process .....	4
Figure 3. Pseudocode for Stochastic Universal Sampling (SUS).....	5
Figure 4. Illustration of fitness proportionate selection.....	6
Figure 5. Queen Bee mating example .....	7
Figure 6. Illustration of single-point crossover .....	8
Figure 7. Illustration of two-point crossover .....	8
Figure 8. Graphical representation of crossover, followed by mutation in a single gene .....	9
Figure 9. The plot of the function $g(x)$ , where the $x$ -axis is the distance ( $x$ ) from the target range, and the $y$ -axis is the fitness value .....	16
Figure 10. Plot of $f(x)$ , the fitness function when outside of the target range, where the $x$ -axis is the distance from the target range, and the $y$ -axis is the fitness value.....	17
Figure 11. Total number of flights throughout the day on May 1 in ZAU .....	21
Figure 12. Relationship between duration and generation response variables in the original experiment.....	23
Figure 13. Predictor profiler for ZAU .....	25
Figure 14. Predictor profiler for ZDV .....	26
Figure 15. Relationship between duration and generation response variables in the supplemental experiment .....	28
Figure 16. Box plot and confidence interval diamonds for number of generations and duration .	29
Figure 17. Predictor profiler for supplemental experiment .....	29

## List of Tables

Table 1. Factors and levels used in the experiment.....	20
Table 2. Parameter estimates for generation response .....	24
Table 3. Parameter estimates for duration response .....	24

# 1. Introduction

The Modeling and Simulation Branch (also referred to by organization code ANG-C55) of the Federal Aviation Administration (FAA) conducts research to assess the operational and technical feasibility of proposed system changes to National Airspace System (NAS) operations. The branch supports several initiatives under the Next Generation Air Transportation System (NextGen) with the goal of improving safety and efficiency in the NAS. A key area of research is evaluation of ground automation supporting air traffic controllers, including conflict probes (CP). A CP monitors flights in the air and looks along their predicted paths for potential violation of separation standards.

To support the development and testing of a CP, the FAA must simulate the CP in a realistic context that is indicative of the operational environment. Thus, recordings of actual air traffic are desirable since they contain the real-world events like misinterpreted messages and noise in the tracking data. To evaluate the CP, trajectories need to contain events that either violate separation standards or at the very least come close to violating said standards. These events are referred to as conflicts and encounters, respectively. Air traffic controllers manage aircraft to maintain separation in the live NAS, so recorded data are not useful as-is for testing the CP. However, the data can be used as a basis for generating test scenarios by shifting individual flight tracks forward or backward in time in order to induce conflicts and encounters. The Modeling and Simulation Branch has extensive experience in the methodology of generating time-shifted test scenarios. While they are very useful for testing CP performance, time-shifted air traffic scenarios are also valuable in other research that requires realistic conflicts and encounters.

The remainder of this document provides a background on pre-existing air traffic scenario generation methods, description of the new implementation, two experiments performed using the new implementation, conclusions, and future work to be done. Section 2 provides information on the original implementation of creating time-shifted scenarios, details the original genetic algorithm implementation, and describes the genetic algorithm process and how it is implemented to suit the team's needs. Section 3 describes the new Java implementation of the genetic algorithm, and compares it to the original implementation. Section 4 discusses the setup and results of the first experiment for the new implementation. Section 5 explains the approach and results of a second experiment. Section 6 summarizes the conclusions of the work, and Section 7 describes the future work to be done on the new implementation.

## 2. Prior Work in Air Traffic Scenario Generation

A study was performed to evaluate the effect of time shifting flights for scenario generation, and how it would affect the performance of a conflict probe. It was determined that flights can be time shifted up to one hour without loss of trajectory accuracy (Paglione et al., 1999). With these results, the first implementation for creating time-shifted air traffic scenarios was implemented by the Engineering and Integration Services Branch (ACT-250), an earlier iteration of ANG-C55, for Lockheed Martin Air Traffic Management Division (LMATM). These scenarios were used in accuracy testing of the User Request Evaluation Tool Core Capability Limited Deployment (URET CCLD) system (Oaks and Paglione, 2001). This system generated a random time value from a normal or uniform distribution and added it to the base time of a flight, repeating this random process for every flight in a traffic scenario. After applying the time-shifts, a new scenario was generated. The scenario was then evaluated and regenerated if it was not found acceptable.

Although not necessarily an efficient method, time-shifting as a way of creating air traffic scenarios with induced conflicts was proven successful (Oaks and Paglione, 2001). This version found the appropriate time-shifts by randomly guessing the time-shift values as described above. While the initial implementation can create a scenario that satisfies the given constraints, other algorithms could be used to decrease the time taken to solve these problems.

### 2.1 Genetic Algorithm Implementation – Cat

A few years later, the FAA’s Simulation and Modeling Group (ACB-330) developed a program named Cat<sup>1</sup> which used a genetic algorithm (GA) to choose time-shift values in a much more efficient manner compared to the initial time-shifting methods described above (Oaks, 2002). GAs are based on the evolutionary process of natural selection, in which a population of solutions will evolve into new, and possibly better, solutions based on the characteristics or constraints of the desired solution. They belong to a larger class of evolutionary algorithms (EA), and are one of the more popular algorithms in optimization and search problems (Mitchell, 1998). The remainder of this section describes the GA and how it is implemented in Cat.

#### 2.1.1 Basic GA Structure

Genes form the most basic unit of the GA, and they represent the time-shift for the aircraft flights in the input air traffic scenario. Genes can be represented by  $\langle \Delta t_1, \Delta t_2, \dots, \Delta t_n \rangle$ , where each  $\Delta t_n$  is associated to a unique Aircraft ID (ACID) (Oaks, 2002). Cat relies on the assumption that the list of time-shift values matches a sorted list of ACIDs.

The original implementation of Cat restricted the amount of time each flight could be shifted to up to one hour. The time-shift values are represented in tens of seconds, and were constrained to fall between 0 and 359 (Oaks, 2002). This constraint was applied since it was shown in a previous study that time shifting flights by more than an hour causes a statistically significant effect on the horizontal trajectory prediction accuracy of the conflict probe (Paglione et al., 1999). An updated implementation of Cat allowed the flights to be time-shifted either forward or backward in time, by at most up to one hour in either direction (Oaks and Paglione, 2002).

---

<sup>1</sup> Cat is 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 spaceship *Red Dwarf* after all but one of its crew was killed by a radiation leak.

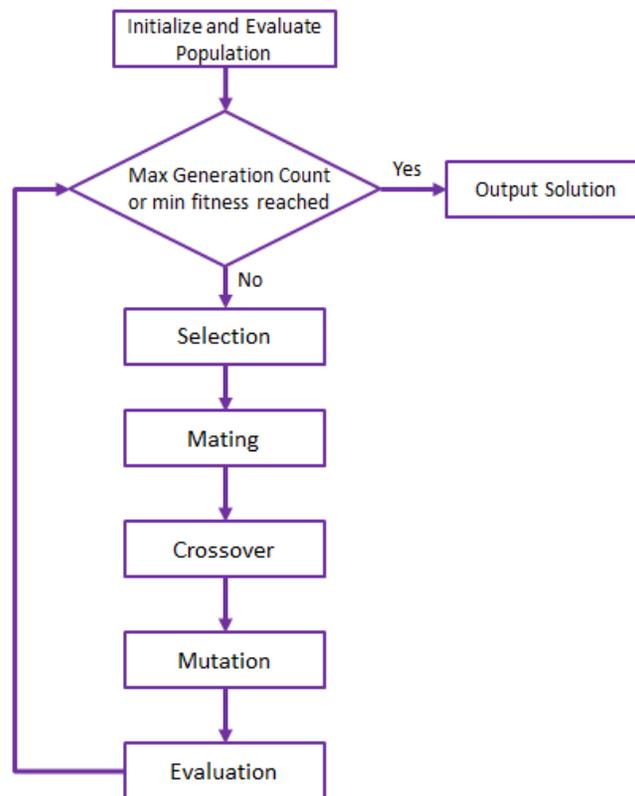
A collection of genes is called a chromosome. Chromosomes in the GA are the potential solutions to the problem the GA is trying to solve, which is obtaining time shift values that create an air scenario with useful conflict properties. Every chromosome contains an entire set of genes, which are the associated time-shift values for each aircraft in an air traffic scenario (Oaks, 2002). In Cat, a chromosome is an array of genes. All chromosomes in a GA comprise the “population.” A population is a subset of the entire solution space, which the GA is currently examining. Cat stores the population as an array of chromosomes.

### 2.1.2 Reproduction Process

To search for better solutions, the algorithm mimics the process of reproduction. Cat initializes the population by generating a random set of chromosomes. These chromosomes are evaluated to determine the initial fitness. Then, a *while*-loop begins the evolutionary process. Selection, mating, crossover, and mutation, described below in detail, are completed to create an offspring population which is then evaluated to calculate the new fitness of each chromosome (Oaks and Paglione, 2002). The generation is incremented, and the process begins again if the termination criteria have not been met. Figure 1 displays the pseudocode for a generic genetic algorithm adapted from (Mitchell, 1998). In the process defined in Cat, create an offspring population by crossover is broken into two components: mating chromosome pairs together and crossover reproduction. This is shown in Figure 2, which illustrates the GA process in Cat.

```
Initialize generation to zero
Initialize the population
Evaluate the fitness of each chromosome in the population
While (test for termination criteria fails) {
    Manage the elite
    Select a parent population
    Create an offspring population by crossover
    Randomly mutate the chromosomes of the offspring population
    Save the offspring as the current population
    Evaluate the fitness of each chromosome in the population
    Increment generation
}
```

**Figure 1. Pseudocode for a generic genetic algorithm**



**Figure 2. Flow chart of Cat GA process**

The details of the selection, mating, crossover, mutation, and elitism subprocesses will be discussed in the following subsections.

### 2.1.2.1 Selection

In each generation, a proportion of the existing population is selected for breeding the new population. This process is called selection. Chromosomes are selected using an algorithm, and the selected chromosomes move on to the mating process. It is the first subprocess of reproduction. Individuals in the population (chromosomes) are selected based on how well they match the constraints. Selection must not be too strong to select suboptimal highly-fit chromosomes (chromosomes that are high in fitness, but actually make it harder to find the most fit chromosomes), reducing diversity needed for more progress through the solution space; selection must also be not too weak to cause a slow evolution (Mitchell, 1998). In Cat, five different selection approaches are implemented for the user to choose from.

An example of selection that is implemented in Cat is tournament selection. In this process, two chromosomes from the population are chosen at random via a uniform distribution. The two chromosomes are compared, and the one with the higher fitness is chosen. This process continues until a population of the selected chromosomes is created.

Another way of performing selection is with Stochastic Universal Sampling (SUS). Cat implements four selection techniques that use SUS: Sigma scaling, Boltzmann scaling, fitness proportionate selection, and rank selection.

The calculation of SUS depends on the “expected value” of the chromosome’s fitness to minimize the spread of values in the candidates. As defined by Mitchell (1998), the “expected value” is the expected number of times an individual will be selected to reproduce. Sigma scaling, Boltzmann selection, fitness proportionate, and rank selection are different ways of calculating the expected value (*ExpVal*) for the chromosomes. SUS is implemented in Cat using the following pseudocode as shown in Figure 3 (Mitchell, 1998).

```
ptr = Rand(); Returns random number uniformly distributed in [0,1]
for(sum = i = 0; i < N; i++)
    for(sum += ExpVal(i,t); sum > ptr; ptr++)
        Select(i);
```

**Figure 3. Pseudocode for Stochastic Universal Sampling (SUS)**

To start the SUS implementation, a random number is selected from a uniform distribution. In the first *for*-loop, *sum* and *i* are initialized to 0. This *for*-loop continues until the variable *i* reaches the total population size, *N*. In the second *for*-loop, the expected value, based on the current chromosome (*i*) in the current iteration (*t*), is added to the current sum value. If the *for*-loop doesn’t exit, the current chromosome is selected. This allows for a chromosome to be selected more than once, adding a “copy” of it to the selected list.

### **Sigma Scaling**

Sigma scaling is based on an individual’s contribution to the total fitness of the population. The expected value is calculated based on the fitness of the individual chromosome, the average fitness of the population, and the standard deviation of the fitness of the population. The equation Sigma scaling implements is shown in equation (1) as restated from Mitchell (1998).

$$ExpVal(i, t) = \begin{cases} 1 + \frac{f(i) - \bar{f}(t)}{2\sigma(t)} & \sigma(t) \neq 0 \\ 1 & \sigma(t) = 0 \end{cases} \quad (1)$$

where  $f(i)$  is the fitness of the current chromosome,  $\bar{f}(t)$  is the average fitness of the population, and  $\sigma(t)$  is the standard deviation of the fitness of the population. These expected values are then used in SUS to complete the selection process.

### **Boltzmann Scaling**

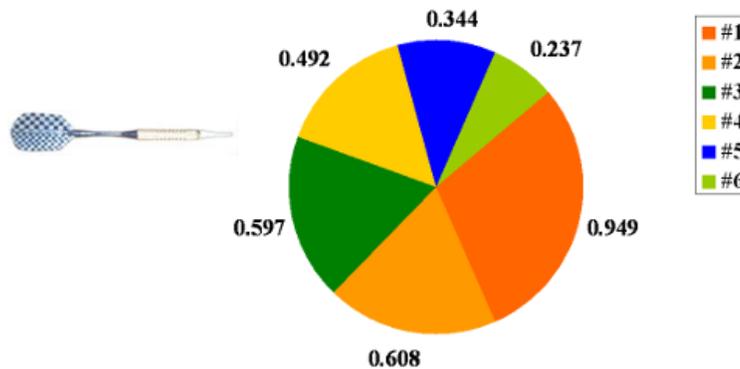
Boltzmann is the probability of selection that varies as the process evolves. This is similar to “simulated annealing” in which the temperature, a measure of controlling the selection process, is varied based on a preset schedule (Mitchell, 1998). In the beginning, selection pressure is low, so every individual has an equal probability of being selected for mating. As time goes on, the temperature is adjusted, increasing the selection pressure of the GA to narrow in to the best part of the search space while maintaining an appropriate level of diversity (Mitchell, 1998). Cat implements Boltzmann selection using equation (2) as restated from Mitchell (1998).

$$ExpVal(i, t) = \frac{e^{f(i)/T}}{\langle e^{f(i)/T} \rangle_t} \quad (2)$$

where  $T$  is the temperature and  $\langle e^{f^{(i)}/T} \rangle_t$  is the expected fitness over the population at time  $t$ . Temperature is gradually decreased according to a predefined schedule, which allows for the selection pressure to increase gradually.

### Fitness Proportionate

Fitness proportionate selection is based on a chromosome's contribution to the total fitness of the population (Oaks and Paglione, 2002). The expected value in fitness proportionate selection is defined as the fitness of the chromosome divided by the average fitness of the population (Mitchell, 1998). A chromosome is randomly selected with unequal probabilities. If the chromosome has a higher fitness, then it has a higher probability of being selected. This is represented by the dart and chart shown in Figure 4.



**Figure 4. Illustration of fitness proportionate selection**

In Figure 3, the highest fit chromosome (fitness of 0.949) has the largest chance of being picked, but it does not have a probability of over 0.5 since the second and third highest fit chromosomes still contribute to the fitness of the population.

### Rank Selection

In rank selection, each chromosome is ranked according to their fitness value, and the expected value is based on the individual's rank instead of its fitness value (Mitchell, 1998). This keeps the selection pressure high when the variance in fitness is low.

In Cat, a population is created based on one of these selection strategies, and the selected population proceeds to the mating process.

#### 2.1.2.2 Mating

Mating, or parentage, is when chromosomes are matched with each other. In nature, there are a variety of different mating rituals between different animal species. This process is normally part of the selection process, but Cat implemented it as a separate feature (Petzinger et al., 2011). Cat implements four mating algorithms which can be chosen by the user.

### Shuffled/Sorted

Two ways a selected population can be matched are shuffled mating and sorted mating. Shuffled mating is when the chromosomes are mated randomly with each other. Each chromosome has an equal chance of being mated with any other chromosome, and there is no favoritism towards higher or lower fit chromosomes. This places them next to each other in a list in preparation for the next step of the GA, crossover.

Sorted mating is when the chromosomes with the highest fitness are mated with the chromosomes with the lowest fitness. This was the original implementation of mating used by Cat, which was then replaced by the “Queen Bee” method.

### Queen Bee Random and Sorted

The “queen bee” mating method is based on how queen bees mate in real life (Jung, 2003). There are two queen bee algorithms that are implemented in Cat: queen bee random and queen bee top. Both queen bee mating algorithms set the chromosome with the highest fitness set as the “queen bee”. Queen bee top’s algorithm mates the queen bee with the top half of the selected population. Queen bee random’s mates the queen bee chromosome randomly with the other chromosomes in the selected population. The queen bee chromosome becomes the mate to multiple other chromosomes. Figure 5 shows a small example of a mating process in which the queen bee is mated with two other chromosomes.

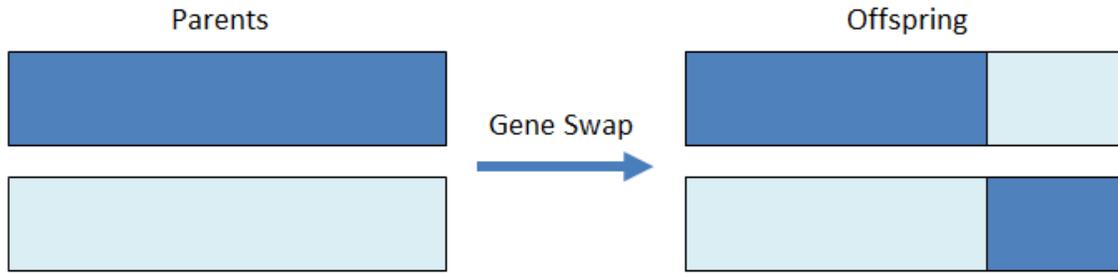
Queen Bee:	7	2	5	1	3	4
Mate:	1	3	6	5	7	3
Queen Bee:	7	2	5	1	3	4
Mate:	4	0	3	9	2	8

Figure 5. Queen Bee mating example

Each time it is mated, the queen bee chromosome is copied into a list of “mated” chromosomes. This means that half of the list will be a copy of the queen bee chromosome and the other half of the list will consist of the mates to the queen bee. The determination of mates depends on which algorithm is used, as described above. Once either process is complete, the algorithm moves on to crossover and mutation.

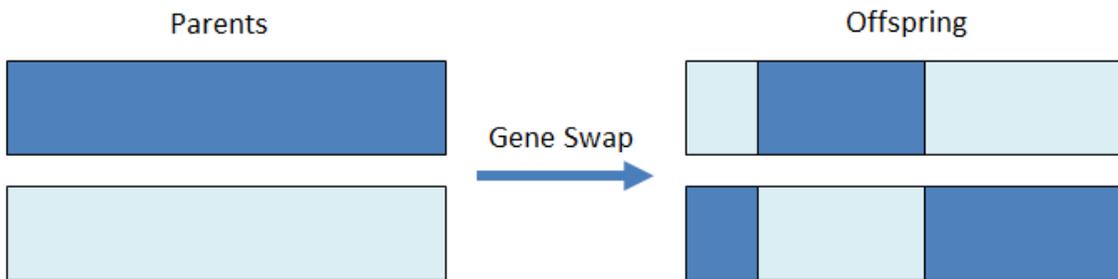
### 2.1.2.3 Crossover and Mutation

Once the parents are matched, genes are exchanged between the mated chromosomes in a process referred to as crossover. One strategy, single-point crossover, chooses a random location on the chromosome to be a boundary. On one side of the boundary, the genes are swapped between the parent chromosomes, while on the other side no swapping occurs. This approach is shown in Figure 6.



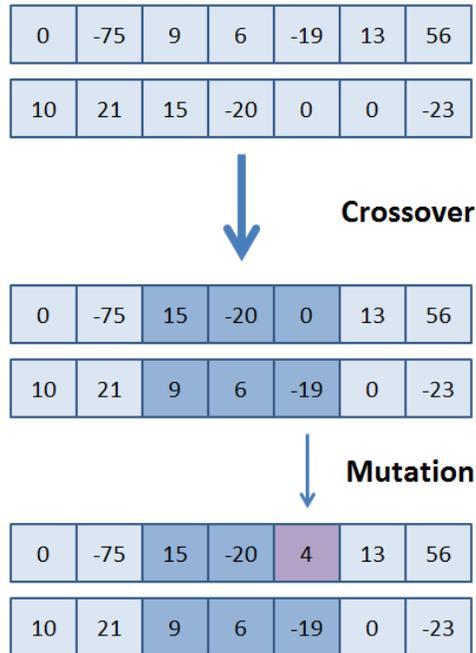
**Figure 6. Illustration of single-point crossover**

Another strategy is two-point crossover, in which two locations are chosen at random. The genes between the two points remain, and the genes outside of these two points are exchanged between the parents. Figure 7 illustrates the two-point crossover approach.



**Figure 7. Illustration of two-point crossover**

After crossover the genes may undergo mutation, a process which replaces the time-shift value for a mutated gene. The probability of mutation for each gene is set by the user. There is also a chance that crossover may not occur, again with the probability set by the user. Mutation can still occur if crossover does not happen. The time-shift value resulting from mutation is randomly generated from a distribution that is also specified by the user (uniform, Gaussian, and truncated Gaussian). Figure 8 shows an example of crossover with mutation.



**Figure 8. Graphical representation of crossover, followed by mutation in a single gene**

This representation shows that the middle portion of the two parent chromosomes is swapped via two-point crossover. Then one of the genes is mutated, giving it a new value. The probability of mutation of any one gene is set by the user to a fixed value. There is also a chance that crossover may not occur, and that probability is set by the user. Mutation can still occur if crossover does not happen.

#### 2.1.2.4 Elitism

Within the reproduction of the population, elitism is used to preserve the best fitness throughout the evolution cycle. Before reproduction, the program saves the best chromosome, or set of chromosomes, known as the elites. Once reproduction is complete, the program analyzes the chromosomes and compares them to the elites. The elites replace the weaker members of the population, driving the average fitness and the best fitness higher than it would have been without the elites. This helps improve the GA performance by ensuring that the best chromosomes are present in the succeeding generation (Oaks and Paglione, 2002).

#### 2.1.3 Evaluation

In order to evolve the population towards a desired goal, the program must know something about the quality of each solution. Upon completion of reproduction, every chromosome is tested to see whether the new scenario it would create would satisfy the given constraints of the desired solution. As stated earlier, the GA creates a time-shifted air traffic scenario with properties meeting a user-specified set of constraints. The constraints, in this case, describe events that violate separation standards (conflicts) or come close (encounters).

The constraints used for defining the desired solution in Cat began with the definitions of conflicts in (Paglione, Oaks, and Bilimoria, 2003). These definitions included conflicts with varied closest point of approach (CPA) of the two aircraft involved. Vertical flight phase is then added to the constraints, allowing the user to define whether the aircraft involved in the conflict

are level, descending, or ascending during the conflict event. Next in (Paglione, Oaks, and Summerill, 2003), encounter definitions were introduced to constrain the problem even further. The encounter constraints mirrored the conflict constraints, but with values that were more similar to those of encounter definitions (i.e. with larger separation values). Later to support the evaluation in (Crowell et al., 2011), measures of flight adherence (which characterizes how closely a flight is following the route known to air traffic control automation) and popup events were added as constraints. A popup conflict is one that occurs within a threshold number of minutes after either flight's start time or receipt of a clearance. Popup events are not realistic in an actual traffic scenario and the CP is likely not to perform well if a scenario has many of these events, so the inclusion of constraints to minimize the count of popup events is an important development.

To determine how well a chromosome satisfies these constraints, a function is used to provide a score, fitness, for each chromosome. The fitness of a chromosome defined in Cat can be any real number between 0 and 1. The closer to 1, the more "fit" the chromosome is, or the closer it is to fulfilling all constraints. Every constraint is evaluated individually for each chromosome.

Each chromosome is evaluated by determining the number of conflicts and encounters that occur if the time shifts (genes) are applied to the scenario. Cat simulates the flights using efficient approximation methods, so each incident found in the pseudo-scenario is an approximation of what would occur in a true time-shifted scenario. Each incident is described by a set of parameters that detail what is happening between flights, such as separation distance, encounter angle, etc. This information is used to determine how many incidents are within the constraint's parameters. A count value is incremented for each incident that satisfies the current constraint.

Once all incidents are evaluated, the count values are used. The constraints have a target range, which is the range in which the number of incidents needed to be in the scenario to satisfy the constraint in the desired solution. If the count value falls within the target range, the chromosome is assigned a value of 1 for the given constraint. If the value falls out of the target range, the fitness is given by equation (3) (Oaks and Paglione, 2002).

$$f(count_i) = \begin{cases} \frac{count_i}{loBound_i} & count_i < loBound_i \\ 1 & loBound_i \leq count_i \leq hiBound_i \\ \frac{hiBound_i}{count_i} & count_i > hiBound_i \end{cases} \quad (3)$$

where  $count_i$  is the number of incidents that satisfy the current ( $i^{\text{th}}$ ) constraint,  $loBound_i$  is the lower bound and  $hiBound_i$  is the upper bound of the target range for the current constraint. This equation shows that the fitness value of the constraint will decline in a linear fashion as the count value moves away from the target range.

Once a fitness value has been determined for each constraint, the total fitness value of the chromosome is calculated using equation (4).

$$Total\ Fitness = \frac{2^{\sum_{i=1}^N f(count_i)w_i}}{2^{\sum_{i=1}^N w_i}} \quad (4)$$

where  $N$  is the total number of constraints,  $f(count_i)$  is the fitness of the chromosome for each individual constraint, and  $w_i$  is the weight for each constraint, assigned by the user. The weight

value gives more importance to a constraint, forcing the program to give a high priority to solving constraints with a higher weight value. Thus, if the count is within the constraints,  $f(count_i)$  would be evaluated as  $w_i$  times 1, which is the maximum value. If the count is outside the constraints, the resulting  $f(count_i)$  would be smaller than  $w_i$  times 1 based on the ratios in equation (3). The exponential calculation places more emphasis on a chromosome fulfilling a constraint (Oaks and Paglione, 2002).

The GA proved to be more efficient than randomly selecting the times, compressing the times, or randomly selecting and compressing the times in one program (Oaks, 2002). As constraints are added to define the desired solution in more detail, the time for the program to run to completion and the number of generations increased.

## **2.2 Island Model and Jaguar**

As constraints were added, Cat's average completion time increased (Petzinger et al., 2011). The island model was proposed as a way to increase the speed of the GA. The island model is a method in which multiple GAs run in parallel, each on its own computer. Periodically, the GAs share their population information with each other, helping each other avoid becoming stuck in local fitness plateaus (Petzinger et al., 2011). The island model was implemented in Cat, allowing the GA to run more quickly. At the same time, a study was performed to determine optimal parameters for Cat to see if the completion time could be further reduced. In the study, the Queen Bee model, Sigma Scaling, and Two Point Crossover methods were found to be the most efficient in reaching an acceptable fitness in the least number of generations (Petzinger et al., 2011).

In an attempt to make Cat more modular and increase its speed further, a rewrite of the code base was performed. This new program, called Jaguar, was written in Java and implemented a caching scheme that stored old chromosomes in an attempt to prevent unnecessary re-evaluation, since evaluation of the fitness is the most time consuming part of the algorithm. Also, a distributed mode was created so that the fitness calculations could be distributed to multiple machines, reducing the workload of one machine, and in theory, speeding up the evaluation process (Petzinger, Oaks, and Nelson, 2011). The caching mechanism was found to be less effective than theorized, taking up too much memory in RAM. In addition, the distributed mechanism was unstable; the machines often lost connection with the master box, halting progress of the algorithm. For these reasons, work on Jaguar was discontinued.

## **3. New Implementation: JavaCat**

JavaCat is the current implementation used for generating conflict-based scenarios. JavaCat builds upon the GA implementation that Cat and Jaguar have implemented. It makes use of object-oriented design as well as improved algorithms for solving the problem.

### **3.1 Changes to Basic GA Structure**

JavaCat structures the basic elements of the GA, the genes, chromosomes, and population, in a similar way to Cat. However, following modern object-oriented design, the elements are now different classes. This design allows for a more concise definition of gene, chromosome, and population, with each containing functions specific to their roles.

In JavaCat, the gene element is a class, and the class elements are the time-shift values of the aircraft, as well as its aircraft identification number (ACID). Cat assumed that the list of time-shifts matched a sorted list of ACIDs. By having the gene class structured in a class with both ACID and time-shift, the assumption has been removed, making the data more clearly represented. Equality functions are also added, allowing for the genes to be sorted in a list, and easy checked for duplicates.

Chromosomes are also made into a class instead of the array-based structure implemented in Cat. Each chromosome contains the entire set of genes, the ACIDs and associated time-shift values for an entire air traffic scenario, in a list. The chromosome class is designed to be able to hold the list of gene objects, along with fitness value and a string value containing results of evaluation. With the class-based approach, more functionality added easily. At present, function such as list sorting for the genes, list sorting for the chromosomes, the ability to access genes directly in a list-based fashion, and the storage of the fitness and evaluation results within the chromosome have all been added. This allows for a more clearly defined way of accessing the genes during the mutation stage of the algorithm, as well as for easy comparison of the chromosomes when sorting them by fitness.

The population, the group of potential solutions (chromosomes), is also made into a class in JavaCat. By making the population a class, additional information can be contained within the population for statistical analysis and for the reproduction process of the GA. Within the population class, a list of chromosome objects exists, as well as the population size value, the average fitness, and the standard deviation. An updateStats method is implemented to update the average fitness and standard deviation values stored within the class based on the fitness stored in each chromosome object. Methods are also implemented for removing chromosomes at the end of the list, as well as for adding them and retrieving them if necessary.

### **3.2 Changes to Reproduction**

In the new implementation, object-oriented design is used often in redesigning this process in the GA. This section describes how selection, mating, crossover, and mutation are implemented in JavaCat, and how they are different from Cat's original implementation.

#### **3.2.1 Selection**

Selection is the process of identifying chromosomes that will move on the mating process. In reproduction, this is the stage in which a mate is selected. Cat implemented many different selection techniques including tournament selection, rank selection, Boltzmann selection, Sigma

scaled, and fitness proportionate. JavaCat implemented the Sigma scaled selection, which is found to be a good selection strategy in Cat (Oaks and Paglione, 2002).

When implementing the selection portion of the reproduction process, an interface is created called *SelectionStrategy*. This interface allows multiple versions of the selection process to be implemented without radical changes to the code. The GA is setup to allow the user to select which selection strategy to use for a given run. This provides flexibility to the user, and makes maintaining the algorithm easier. The object-oriented design facilitates easy extension of the code base as future developers need only provide a new class file which provides a new selection strategy.

### 3.2.2 Mating

Mating, or parentage, is when chromosomes are matched with each other for crossover. JavaCat implements same the two versions of Queen Bee mating that Cat implemented, which are determined to be the two best methods of selection (Petzinger et al., 2011).

In JavaCat, a *MatingStrategy* interface is created to allow multiple mating strategies to be implemented and chosen by the user. Two different mating methods are implemented in JavaCat, which are Queen Bee Top and Queen Bee Random. Both Queen Bee mating algorithms have the chromosome with the highest fitness set as the “queen bee”. This “queen bee” chromosome becomes the mate to multiple other chromosomes. The implementation of Queen Bee Top and Queen Bee Random in JavaCat is similar to the implementation in Cat, but setup to take advantage of the object-oriented design.

### 3.2.3 Crossover and Mutation

In JavaCat, crossover is implemented so that two portions of the chromosomes are swapped instead of just one. This is called Two Point Crossover, which is the strategy in Cat that is the determined to be most effective (Petzinger et al., 2011).

Like the other portions of the GA, crossover is also setup to have an interface, called *CrossoverStrategy*. This interface allows other methods of crossover to be implemented easily, and the user is able to choose whichever strategy they wish, once the strategies are implemented. The interface also contains a default method in which mutation is performed. Mutation is implemented so that each gene is checked to see if it should be mutated. If mutation is to occur, a random time shift is generated from a Gaussian distribution, uniform distribution, or a truncated Gaussian distribution as determined by the user. For example, in the experiment performed, the truncated Gaussian distribution is used, with limits of -3600 and 3600 seconds ( $\pm 1$  hour).

### 3.2.4 Elitism

Elitism is also implemented in JavaCat. Elitism is a method of carrying a certain amount of chromosomes over to the next generation if no chromosomes in the new generation are any better than the elites. It also preserves the highest fit chromosomes from generation to generation. In Cat, elitism is defined as a percentage of chromosomes in the population. For JavaCat, the user specifies the exact number of chromosomes that will be considered elites instead of a percentage, preventing the need to round the elite chromosome quantity up or down.

### 3.3 Changes to Evaluation

There are two major changes to the evaluation process in JavaCat. Two established evaluation tools are used to assess conflict properties in the time-shifted scenario. Also, the calculation of the constraint fitness is updated by using a more robust function.

#### 3.3.1 Evaluation Tools

In JavaCat, the evaluation function is changed dramatically. First, the in-house tools, *TrackCPDetector* and *PopupIncidentDetector* are used to find all incidents and their characteristics in the time-shifted scenarios. These programs are more accurate than the original approximation method implemented in Cat. The output of these programs is used to evaluate each scenario when determining the fitness.

*TrackCPDetector* is a program that detects all incidents contained in a scenario (Crowell et al., 2011). It iterates through all of the spatially-filtered flight pairs (flight pairs that can potentially have incidents), and finds all incidents for each flight pair. For every incident, it determines all of the information about the event, such as distance, encounter angle, and determines whether the flights are transitioning in altitude or they are level. This information is then stored in an object that the program can use to access the information.

*PopupIncidentDetector* is a program that evaluates all incidents found in *TrackCPDetector* and determines if the incident is a popup incident or not. A popup is an incident that occurs within a given threshold from either the start of the aircraft's track, the start of a recorded clearance message, or the exit of an inhibited airspace not modeled for aircraft-to-aircraft conflicts (Paglione, Oaks, and Ryan, 2004). A popup incident could also occur if either aircraft is less than a specified altitude away from a cleared interim or hold altitude at conflict start (Paglione, Oaks, and Ryan, 2004). These events are difficult for a CP to predict accurately, and are not used in the testing procedures for the CP. Once the popups are found, the incident objects are then evaluated to calculate the fitness of the scenario.

#### 3.3.2 Calculating Constraint Fitness

The structure of evaluating the constraints is redesigned in JavaCat to have an object-oriented approach, similar to the GA reproduction structure. The incident object from *TrackCPDetector* and *PopupIncidentDetector* is passed to the constraints package of JavaCat. In this package, an abstract class forms the framework of each constraint class. To reduce the number of classes needed for development, each class has parameters specific to its overall type, allowing the declaration of all 39 potential constraints the team uses in Cat with only five base classes. The base constraint types are Adherence, Angle, Popup, Separation, and Vertical Transition.

An XML structure is defined for specifying constraint types and the values for the constraint. JavaCat builds constraint classes by reading the XML file created and using the base classes implemented in the constraints package. These classes are then stored in a list to allow the program to make use of them when evaluating the data returned from *TrackCPDetector* and *PopupIncidentDetector*.

The incident object is passed into the base constraint classes. If the incident satisfies that constraint, the count value for that constraint is incremented. The classes return their count values, which is then used as a parameter for calculating the fitness in the abstract constraint class. The weight and fitness for the constraint, along with additional information for diagnostics, is returned once all of the data are processed for that scenario. In JavaCat the weight and fitness

values for each constraint are used in a similar fashion as Cat, with equation (4) being used to calculate the total fitness of the population. This information is then stored within the current chromosome class.

The abstract constraint class in JavaCat implements a new fitness function for evaluating the fitness for each chromosome. Cat implements the linear decline function (equation (3)) when the count value is outside of the target range. For JavaCat, a custom function is created to give more information to the GA about how far away it is from the desired solution. The derivation of the new fitness function for JavaCat is detailed below.

The new fitness function is designed to be a function that represents displacement from the desired target range. The target range is defined by a high and low bound. In the new fitness function,  $x$  is defined as the distance outside of the target range. This distance is calculated by using the count value (the number of events that satisfy the current constraint). If the count value is above the high bound, then it is the distance between the count and high bound; if the count value is below the low bound, then the distance is between the count and low bound; and if the count value is in the range, then the distance is 0.

With the displacement from the desired target range defined, the function can be derived. Let  $x \geq 0$  be the distance from the desired target area. Let  $f(x) \in [0 \dots 1]$  be the fitness function. The fitness function must meet the following conditions:

- $f(0) = 1$
- $\lim_{x \rightarrow \infty} f(x) = 0$
- $f(x)$  is strictly monotonic outside of the target range

The function is designed to be smooth outside of the target range, so  $f(x) \in C^1$ , meaning fitness and its derivative are continuous.

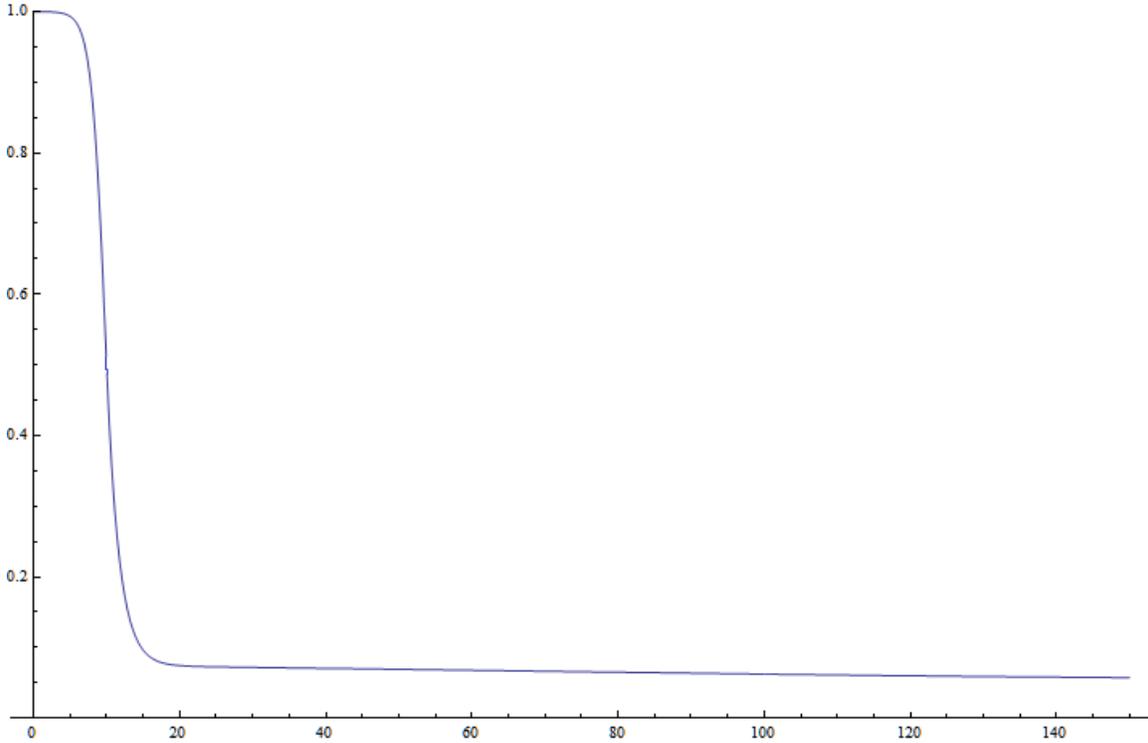
Based on these conditions, a function  $g(x)$  is initially devised. Specific boundary conditions are chosen to change the behavior of the fitness for distances near and far from the target area. 10 is chosen to be near and 100 is chosen to be far, giving  $g(10) = 0.5$  and  $g(100) = 0.0625$ . Thus,  $g(x)$  is defined in equation (5).

$$g(x) = \begin{cases} \frac{1}{1 + e^{x-10}} & x \leq 10 \\ ae^{-bx} + cx + d & 10 < x < 100 \\ \frac{1}{8 \log_{10} x} & x \geq 100 \end{cases} \quad (5)$$

where the constants  $a$ ,  $b$ ,  $c$ , and  $d$  approximately satisfy the requirements for  $g(x)$  when:

$$\begin{aligned} a &= 151.4444117076111 \\ b &= 0.5875213285038344 \\ c &= -1.357170255947662 \times 10^{-4} \\ d &= 0.07607170255947662 \end{aligned} \quad (6)$$

This equation produces the curve shown in Figure 9.



**Figure 9. The plot of the function  $g(x)$ , where the  $x$ -axis is the distance ( $x$ ) from the target range, and the  $y$ -axis is the fitness value**

This equation is a slightly weaker result than the initial requirement, since  $g(0) \approx 1$  and it has extreme mid-range behavior with the sharp curve. However, the secant line from  $f(0)$  to  $f(100)$  is shown in equation (7).

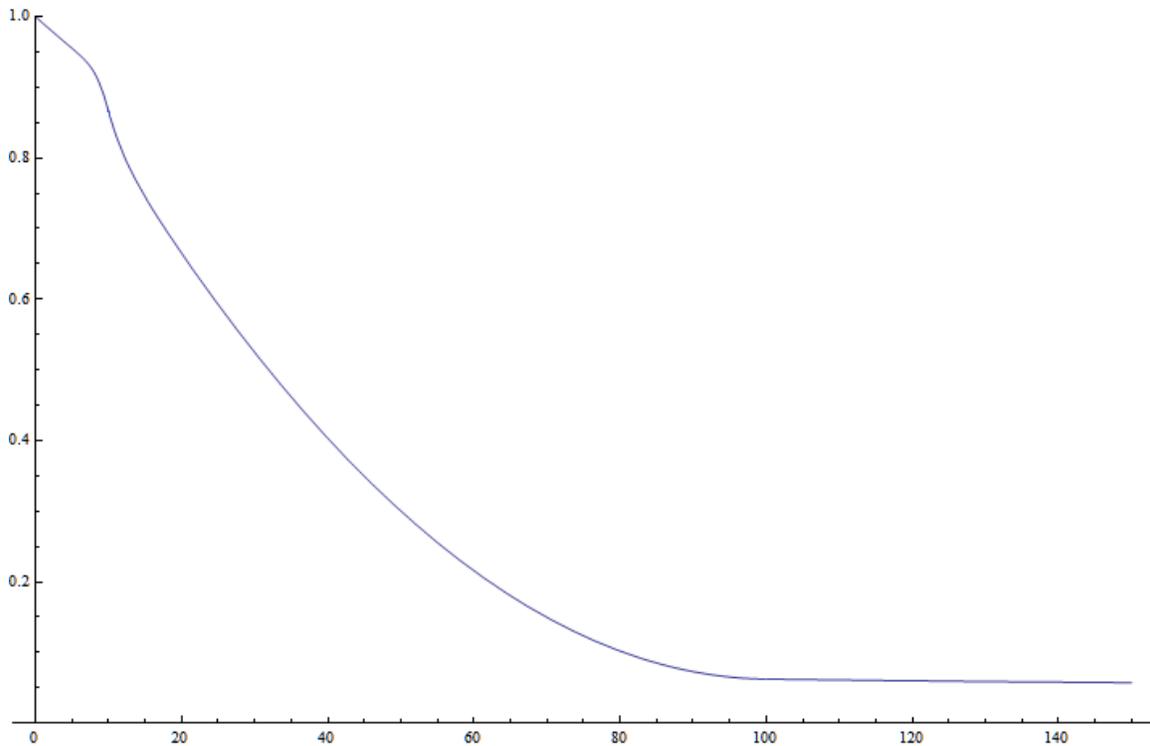
$$s(x) = 1 - \frac{3}{320}x \quad (7)$$

This equation can be used to remedy  $g(x)$  to give a better mid-range curve that consolidates the functions.

With  $g(x)$  and  $s(x)$  defined,  $f(x)$ , the fitness function for the constraint, can be defined as shown in equation (8).

$$f(x) = \begin{cases} \left(\frac{x}{100}\right)g(x) + \left(1 - \frac{x}{100}\right)s(x) & x < 100 \\ g(x) & x \geq 100 \end{cases} \quad (8)$$

This function's plot is shown in Figure 10.



**Figure 10. Plot of  $f(x)$ , the fitness function when outside of the target range, where the  $x$ -axis is the distance from the target range, and the  $y$ -axis is the fitness value**

This function does not reach a fitness of 0.5 when  $x$  is 10, but the function is found to still be acceptable, and it meets all of the desired properties. Refer to the Appendix for a more detailed derivation of this function.

### **3.4 Population Updates Outside of Reproduction**

Typically, upon completion of the reproduction cycle GAs perform evaluation and then begin reproduction again. In JavaCat, a stage is introduced that allows the population to be updated outside of the reproduction process. This notion is introduced to allow the island model, which is a way of updating a population outside of reproduction, to be implemented into the program. It allows the developer to develop new actions for the GA to do after evaluation, but before starting the reproduction process again. During this phase of the GA in JavaCat, the program can continue to the next iteration to skip this process altogether, or it could perform an action, like island migration. A *LifeGoals* interface is created to give the framework to allow the population to be updated in this manner, following the same pattern used for the different elements of reproduction. The user can select the option the program will use in the configuration file, allowing the flexibility of choosing which method the GA will use.

Implementation of the island model is what causes JavaCat to be identified as a distributed GA. This model allows JavaCat to run on multiple machines independently of one another. This model is a metaphor associated with the migration of animals to different geographical areas, in which evolution in these areas can be different and independent of the original area. The process of moving a portion of the population on one island to another is referred to as island migration.

Each instance of JavaCat begins with a different subset of solutions (populations) and runs a GA on each of them. After a set amount of generations determined by the user, all instances of the GA connect to a shared file. The shared file contains a population from the previous GA that is interacting with it. If the file is empty, the GA will populate the file with its current population information and move on. If not, the GA locks the file, blocking the other GAs from using it until file updating is complete. The top half of population saved in the file is compared to the bottom half of the population for the GA, and the chromosomes with the better fitnesses values are kept by the GA. The GA then saves its population to the file, releases the lock, and moves on. The current method used for this sharing of chromosomes matches the previous Cat implementation. Future research will explore alternate techniques to adjust both frequency of migration and how the chromosomes are mixed and shared upon migration.

## 4. Original Experiment

This section describes an experiment carried out to investigate the effects of four factors on the performance of JavaCat. The experimental approach and the definition of the factors being studied here are discussed in Section 4.1. The results are presented in Section 4.2.

### 4.1 Approach

A GA has many factors that can be adjusted. Each factor affects the GA, and the factors typically interact with one another nonlinearly (Mitchell, 1998). The goal of the experiment described in this section is to observe any potential interaction and non-linear effects of the factors.

Research objective: determine the influence and interaction of certain factors on the performance of JavaCat.

Four factors are investigated in this experiment: population size, period of migration, number of islands, and Air Route Traffic Control Center (ARTCC). These factors are selected for investigation based on previous experience with Cat and general knowledge of GAs.

#### 4.1.1 Experimental Design

Population size is a large contributor to the speed required to process each generation. As the population size increases, it takes longer for the GA to complete its evaluation of the population. A larger population, however, allows for more solutions to be checked at one time which may decrease the number of generations required to arrive at a solution. The goal is to find a population size that minimizes both the total duration of the run and the number of generations per run.

Period of migration, referred to as epoch in Petzinger et al. (2011), is how often the population in the islands undergoes migration. The period sets how often this occurs, using an interval specified in terms of the number of generations. The most frequent migration period is every generation. However, a period that is too small may limit the benefits of parallelism in each island's independent GA to the sharing of potential good solutions across all the islands. The goal of the experiment is to determine the most reasonable values for this parameter.

Number of islands is the number of parallel GAs running at one time. In an island model setup, each GA is considered an island, whose population remains isolated from the other GAs except for information exchanged during migration. In this experiment, each GA runs on its own physically separate computer system. The parameter "number of islands" defines the number of GAs that will be involved in migration. In theory, as the number of islands increase, the number of generations and duration of the GA should decrease; however, only through experimentation can this be verified.

The ARTCC is the physical center from which the traffic data are collected. This factor is a blocking factor, since flights from the same ARTCC are expected to have similar flight paths and patterns of behavior (Petzinger et al., 2011). Blocking is often used to reduce variability in the experiment and achieve greater precision (Pox, Hunter, and Hunter, 2005). Chicago (ZAU) and Denver (ZDV) are selected for this study by based on a cluster analysis technique performed in a

previous study, in which the clusters are formed using Ward’s clustering model in JMP®<sup>2</sup> (Young, Fabian, and Paglione, 2013). Only two of the five ARTCCs from this study are chosen for the JavaCat experiment since five ARTCCs would have exceeded our resource budget for the experiment. ZAU and ZDV are chosen to include one eastern ARTCC and one mid-western ARTCC, which have different traffic characteristics.

The experiment is designed using principles from statistical design of experiments (DOE). The design includes three levels of each factor and combines factor levels to define different experimental runs. The factors and levels used are shown in Table 1.

**Table 1. Factors and levels used in the experiment**

<b>Factor</b>	<b>Levels</b>		
Population size	20	60	100
Period of Migration	25	125	225
Number of Islands	5	10	15
ARTCC	ZAU		ZDV

A full factorial design requires 54 runs. A custom design consisting of a balanced set of 20 runs is used here in order to balance available resources and strength of the model. Because the GA process relies on random number generation, unique random seeds are used in each run and the entire experiment is replicated three times to characterize the variation in performance that naturally occurs between runs due to the nature of the GA search algorithm.

Due to limitations on the resources available, there are nine 6-core and six 8-core workstations available to use for the experiment. To balance performance differences between the workstations, the island groupings are setup so that a ratio of three 6-core workstations to two 8-core workstation is constant for each group of islands (5, 10, and 15). The goal of balancing the number of 6-core to 8-core workstations is to help reduce the run to run variation from the performance of the workstations.

### 4.1.2 Scenarios

A flight scenario with about 1,000 flights is desired for this experiment. 1,000 flights is a reasonable sample size that sufficiently represents conflict properties while keeping the scenario small enough for the GA to process quickly. This allows enough runs to be completed for a suitable experiment. The subset of flights are from a time period during the day where the volume of flights in the air is high, and there is no ramp up or ramp down period in the number of flights.

Data from OPSNET<sup>3</sup> is obtained and examined to determine on which day the NAS experienced the least amount of delays with the greatest amount of flights. The month of May is chosen, due to the expectation that May typically has tamer weather with sufficient traffic throughout the month. Since more recent data are desired, the data from May 2015 is examined.

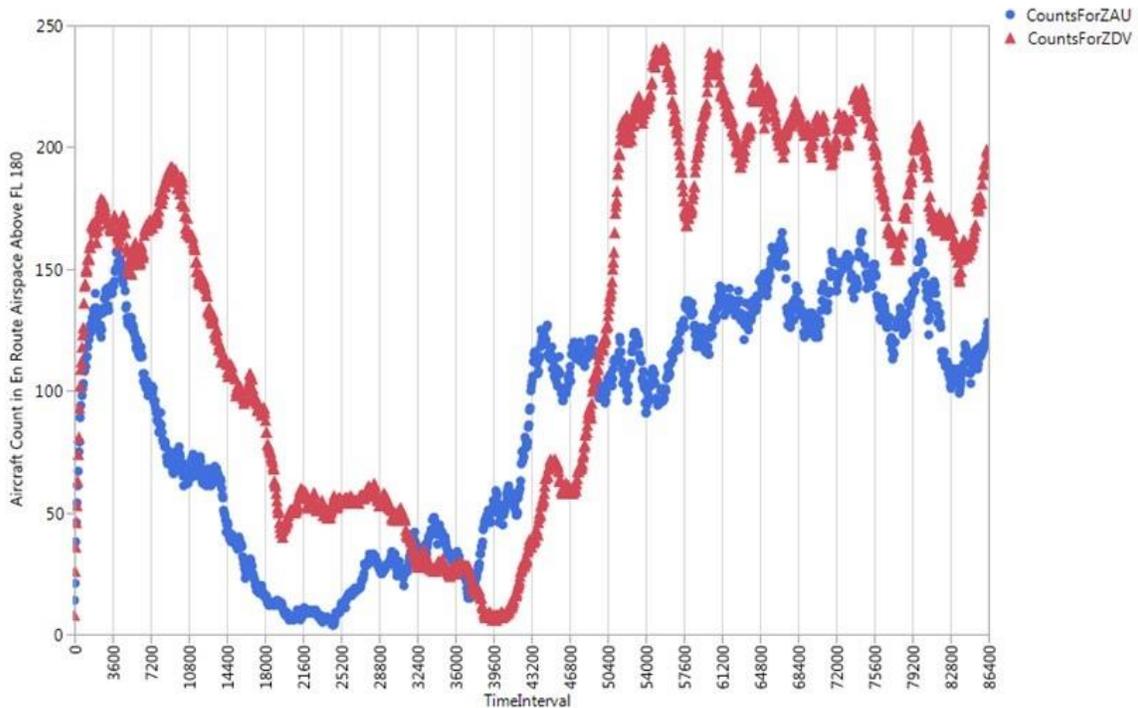
<sup>2</sup> ANG-C55 frequently uses JMP®, a commercially available software tool that provides the user with the capability to perform simple and complex statistical analyses. See <http://www.jmp.com>

<sup>3</sup> Operations Network (OPSNET) is the official source of NAS air traffic operations and delay data. The OPSNET website is <https://aspm.faa.gov/>

The number of operations and the total delay time throughout each day in May is surveyed to determine which day would be sufficient for the experiment. May 1, 2, and 24 are found to have the least amount of delay, with May 2 having the least amount of delay.

The total number of operations for each day in May is also examined. May 1 is found to be having the greatest number of operations in the NAS. Since May 1 also has one of the smallest delay days in the NAS, May 1 is chosen for the day to collect data from. Reducing the delay in the NAS reduces noise that could appear in the experiment. The large number of operations ensures that there are enough flights flying in the NAS to get a modest-sized scenario for testing.

Once the date is determined, the data for the two ARTCCs are gathered from NASQuest<sup>4</sup>. Once this is complete, the full 24-hour scenarios are processed and loaded into ANG-C55's database for analysis. The records are queried in JMP® to count the number of flights throughout the day for both ARTCCs. Flights below 18,000 feet, the floor of en route airspace, are ignored since this experiment focuses on flights in en route airspace. Figure 11 shows the total number of flights above 18,000 feet for ZDV and ZAU.



**Figure 11. Total number of flights throughout the day on May 1 in ZAU**

Using this data, the peak times for each ARTCC can be determined. The interval between 57600 seconds (1600 UTC) and 84800 seconds (2300 UTC) is selected for the scenario as it encompasses the entire peak traffic time, with no ramp up or ramp down, satisfying the requirements of the scenario. The first 1050 flights from this interval are identified and used. Then, any flight that has an altitude below 18,000 feet is removed from the list of flights.

<sup>4</sup> NASQuest is the FAA's data repository for the Common Message Set (CMS) from all 20 en route centers. It collects CMS data from either the legacy Host Computer System (HCS) if still in operation or ERAM.

Approximately 1000 flights are chosen from the remaining data. These flights are then processed using software tools developed by ANG-C55 to create the scenarios.

After creating the scenarios, the program *TrajectoryConflictProbe* is ran. This program uses generated trajectories from the scenarios to predict potential conflicts and encounters, and to create alerts for them. This process is similar to how the operational conflict probe operates, and results in a list of conflicts and encounters and their associated properties that are indicative of what air traffic controllers would have observed on this particular day and airspace. At any given instant of time for a specified window of time in the future, the software is effectively simulating what would have happened if controllers did not resolve predicted conflicts. The result is a list of pseudo conflicts and encounters that are evaluated and the resulting properties set as target constraints for the GA.

Thus, the process of evaluating the output of this program is used in another program, *CatInputFileGenerator*. This program generates an input file for Cat that defines the constraint bins for desired output scenario. The program is updated to produce this file in XML format for JavaCat. At this point, scenario generation is complete and JavaCat is run.

## 4.2 Results

Three full replications of the experiment are run over a period of 6 weeks and the results are compiled using JMP® statistical software. The following subsections describe the underlying statistical model, response variables, and synthesized analysis.

### 4.2.1 Model

The mathematical model developed for this experiment is shown in equation (9). It represents the custom design used, in which all main effects are used and their quadratic, non-linear effects are examined, all levels and factors are crossed, and allowing for only two-level interactions to be examined. This amounts to the four main effects, six two-way interaction terms, and three quadratic terms for the continuous variables. The constant or overall mean effect is represented by  $\mu$ .

Response:

$$R_{ijkl} = \mu + P_i + M_j + I_k + A_l + P_i \times P_i + M_j \times M_j + I_k \times I_k + P_i \times M_j + P_i \times I_k + P_i \times A_l + M_j \times I_k + M_j \times A_l + I_k \times A_l + \varepsilon_{n(ijkl)}$$

where:

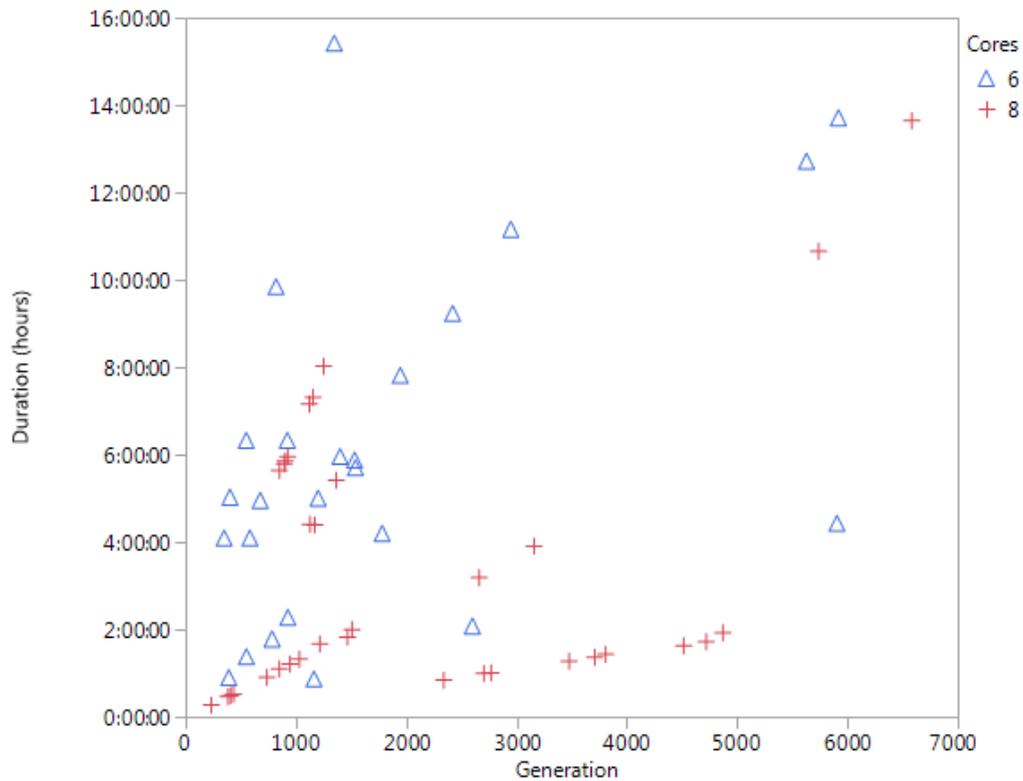
$$\begin{aligned} P_i &= \text{Population Size, } i = 1, 2, 3 \\ M_j &= \text{Migration Period, } j = 1, 2, 3 \\ I_k &= \text{Number of Islands, } k = 1, 2, 3 \\ A_l &= \text{ARTCC, } l = 1, 2 \\ \varepsilon_{n(ijkl)} &= \text{random error, } n = 1, 2, 3 \text{ for all } i, j, k, l \end{aligned} \quad (9)$$

The model assumes that random error  $\varepsilon_{n(ijkl)}$  is approximately independent of the factors, and is normally distributed with zero mean. The model also assumes that all the factors are linearly added as demonstrated in equation (9).

### 4.2.2 Response Variables

The initial response variables studied in the given model are the number of generations and the duration of the GA. The final generation and duration count are recorded upon completion of each run.

These two variables are plotted in Figure 12 to examine their relationship.



**Figure 12. Relationship between duration and generation response variables in the original experiment**

The two variables are not strongly correlated, but increasing generation count is generally shown to have a higher duration. The performance of the workstations had more effect than anticipated, which suggests that workstation type is a significant uncontrolled variable. An attempt is made to reduce this effect by balancing the ratio of 6-core to 8-core workstations, but the effort is not sufficient.

### 4.2.3 Analysis

The experimental data are loaded into the JMP® model and analyzed. The results are presented in two statistical tables. Table 2 provides the results for the generation response variable and Table 3 for the duration response variable.

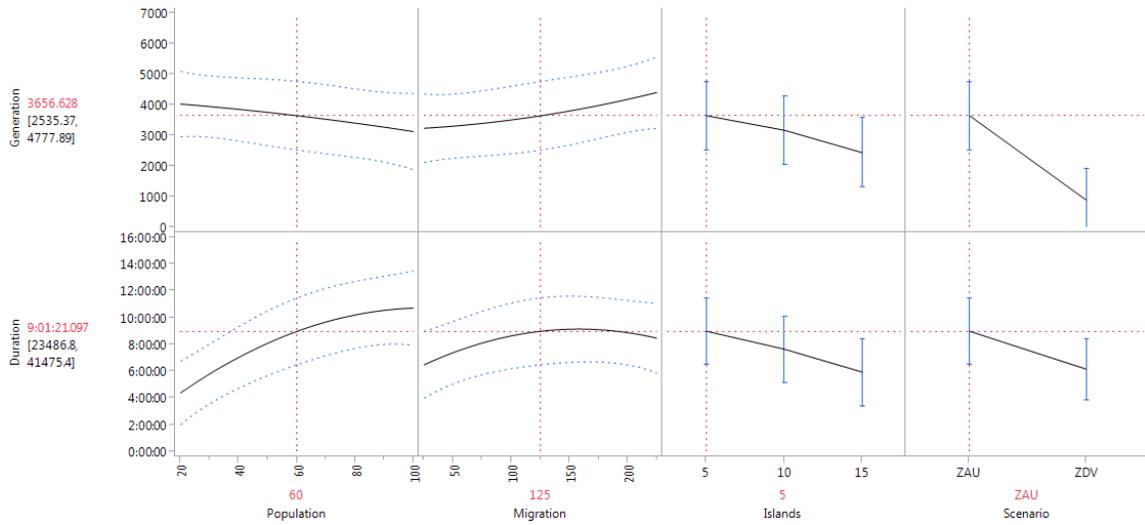
**Table 2. Parameter estimates for generation response**

Term	Estimate	Std Error	t Ratio	Prob> t
<b>Scenario[ZAU]</b>	1139.171	156.7998	7.27	<b>&lt;.0001</b>
<b>Migration(25,225)</b>	458.8994	176.5454	2.6	<b>0.0125</b>
<b>Islands(5,15)</b>	-364.434	176.5454	-2.06	<b>0.0447</b>
Migration*Scenario[ZAU]	263.1415	176.6865	1.49	0.1432
Islands*Scenario[ZAU]	-238.525	176.6865	-1.35	0.1836
<b>Population(20,100)</b>	-219.091	178.1362	-1.23	0.225
Population*Migration	-196.506	195.9	-1	0.3211
Population*Scenario[ZAU]	-165.591	174.7704	-0.95	0.3483
Migration*Islands	135.0966	195.2971	0.69	0.4926
Migration*Migration	178.4766	395.2489	0.45	0.6537
Population*Islands	65.82711	195.9	0.34	0.7384
Islands*Islands	-129.523	395.2489	-0.33	0.7446
Population*Population	-68.9172	398.4481	-0.17	0.8634

Referring to Table 2 for generation count, only the three main factors, scenario, migration, and islands had statistically significant effects at a 5% confidence level (highlighted in bold). In Table 3, the duration response variable, only the main effect of population and the interaction of island and scenario had statistically significant effects at the same level (again highlighted in bold).

**Table 3. Parameter estimates for duration response**

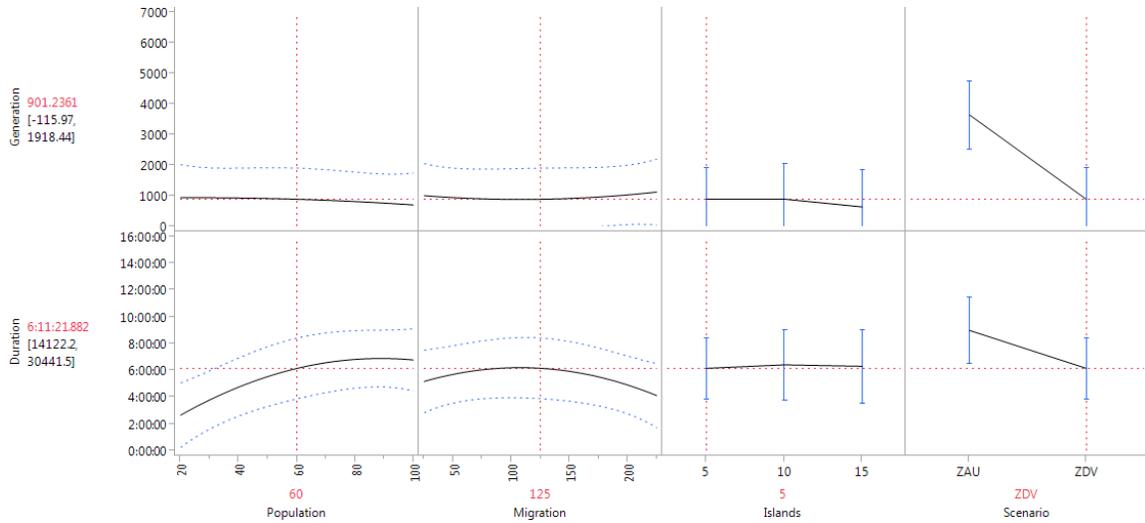
Term	Estimate	Std Error	t Ratio	Prob> t
<b>Population(20,100)</b>	10053.4	1428.94	7.04	<b>&lt;.0001</b>
<b>Islands*Scenario[ZAU]</b>	-2882.69	1417.311	-2.03	<b>0.0478</b>
Migration(25,225)	2773.118	1416.179	1.96	0.0563
Migration*Scenario[ZAU]	2751.703	1417.311	1.94	0.0583
Islands(5,15)	-2610.47	1416.179	-1.84	0.0717
Scenario[ZAU]	2216.918	1257.787	1.76	0.0846
Migration*Migration	-5462.47	3170.534	-1.72	0.0916
Population*Population	-5168.82	3196.196	-1.62	0.1127
Population*Scenario[ZAU]	1992.018	1401.94	1.42	0.1621
Migration*Islands	1972.711	1566.597	1.26	0.2143
Population*Migration	-1053.25	1571.434	-0.67	0.506
Population*Islands	644.9082	1571.434	0.41	0.6834
Islands*Islands	-670.319	3170.534	-0.21	0.8335



**Figure 13. Predictor profiler for ZAU**

JMP® provides a powerful graphical representation of the resulting model, called the predictor profiler. This is illustrated in Figure 13 and Figure 14 for ZAU and ZDV, respectively. They illustrate modeled responses and factors and include confidence intervals. A confidence level implies that a specified percentage of all samples would give an interval that includes the parameter being estimated, with the rest of the samples yielding an erroneous interval (Devore, 2012). The large width of these confidence intervals indicates the imprecision of the model’s predictions, suggesting that the model as presented is not able to capture all of the noise in the GA process. The results of the statistical tests are somewhat consistent with what the predictor profiler is illustrating. A large slope for duration response for population, reflected in a very small  $p$ -value<sup>5</sup> in Table 3. Similarly, a very large slope or effect is observed between scenarios for generation count which is also reflected in a very small  $p$ -value in Table 2.

<sup>5</sup> (Devore, 2012) defines the  $p$ -value as the smallest level of significance at which the null hypothesis would be rejected. If the  $p$ -value is less than the required  $\alpha$  value (0.05), the null hypothesis should be rejected.



**Figure 14. Predictor profiler for ZDV**

However, the confidence level is very large in both ARTCCs for population, migration period, and number of islands. The minimal slope in the number of generations as the population increases shows that the population does not have a significant effect on the number of generations the GA takes to find an acceptable solution, as presented in Table 2. This is not the case in the duration response, which population has a significant effect, as revealed in Table 3.

Overall, all the factors and both responses have wide confidence intervals around the modelled line as illustrated in Figure 13 and Figure 14. While in some cases the effect (shown by a steep slope) is larger than the confidence interval, in many of the plots this is not the case. This is attributed to the large run-to-run variance measured by replicating the experiment three times and using different random seeds values. This also attributes to the variation caused by the mixture of box types used in the various runs. As stated earlier, each run consisted of the same ratio of different performing 6-core and 8-core computer workstations. A priori it is assumed that balancing each run with the same ratio of 6 and 8-core workstations would be sufficient to control the variance caused by their strict performance difference. However, it is observed that this is not the case. 6-core and 8-core boxes would benefit quite differently in the runs and it is somewhat arbitrary on how this occurred in the experiment. Many times 6-core boxes would benefit from the sharing of the faster 8-core boxes yet solve the GA faster with fewer generations. The opposite can also occur by the significantly faster 8-core boxes. Thus, a smaller and targeted supplemental experiment is proposed that only used either 6 or 8 core boxes, eliminating this uncontrolled source of variation.

## 5. Supplemental Experiment

The original experiment described in Section 4 is only able to provide a partial answer to the original objective. The results exhibit a significant amount of noise and there are unexpected results from competition due to the different processing power in each island.

Of particular interest is how the number of islands affects the time and number of generations required to arrive at a solution. To better understand this effect, another experiment is performed in an attempt to attenuate the potential influence from using machines with different cores and processor speeds. In addition, it also eliminates other factors to focus on the effect from number of islands.

The experimental approach is discussed in Section 5.1, along with the definition of the factors being studied. The results are then presented in Section 5.2.

### 5.1 Approach

The goal of this experiment is to reduce the noise from factors such as different CPU types and traffic from different ARTCCs in order to better investigate the effects of the number of islands on the number of generations. Nine computers are used in this experiment, all the same type of 8-core CPU style workstation. The main factor tested is the number of islands used per GA run, of three, six, and nine.

The ZDV traffic scenario is chosen because the ZDV runs in the previous experiment exhibited the lowest variance in responses. A population of 20 is used for all runs, along with a migration period of 125 generations. These numbers are chosen since the experimental runs using traffic from ZDV had the lowest variance between runs.

The runs for this experiment are setup so that each grouping had nine replications. This results in a total of 27 runs for this experiment. The order in which the groups are run is varied, and spanned all possible combinations of the islands. This allows for a sizable number of replications based on the variance in the data.

### 5.2 Results

Nine replications of this experiment are completed over a period of one week and the results are compiled using JMP® statistical software. The following subsections describe the underlying statistical model, response variables, and synthesized analysis.

#### 5.2.1 Model

The mathematical model developed for the supplemental experiment is a simplified as compared to the one in the original experiment, and is shown in equation (10). It represents a custom design which has only the main effect and a quadratic effect. The constant or overall mean effect is represented by  $\mu$ .

Response:

$$R_k = \mu + I_k + I_k \times I_k + \varepsilon_{n(k)}$$

Where:

$I_k$  = Number of Islands,  $k = 1, 2, 3$

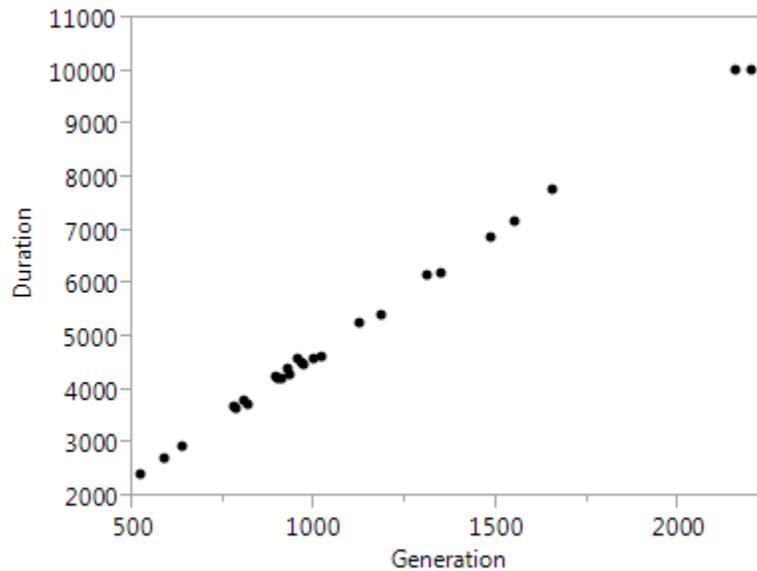
$\varepsilon_{n(k)}$  = 1, 2, ... for all  $k$

(10)

The model assumes that random error  $\varepsilon_{n(k)}$  is approximately independent of the factors, and is normally distributed with zero mean. The model also assumes that all the factors are linearly added as demonstrated in equation (10).

### 5.2.2 Response Variables

Due to the mixture of 6-core and 8-core CPU workstations in the original experiment, the generation and duration response variables are not strongly correlated. These two variables are examined again in the supplemental experiment where all islands are using the same CPUs. After the results are gathered, generation and duration data are plotted against each other as shown in Figure 15.

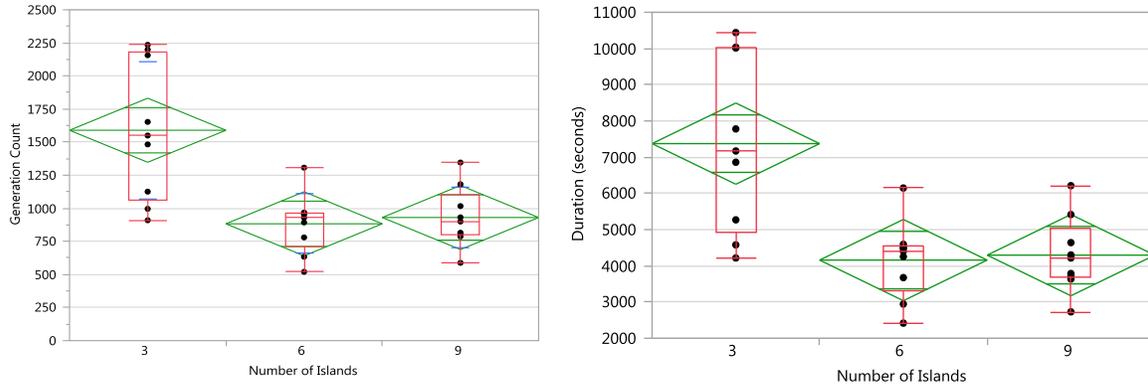


**Figure 15. Relationship between duration and generation response variables in the supplemental experiment**

In this experiment, the correlation between generation and duration is much stronger. The effect of different performances of workstations is eliminated with the use of similar workstations for all islands.

### 5.2.3 Analysis

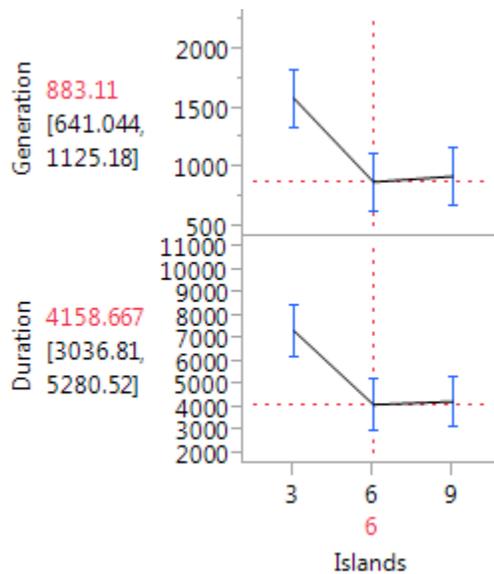
The results of this experiment are loaded into the JMP® model and analyzed. The count of generations at completion is plotted against the number of islands simulated, as shown in Figure 16.



**Figure 16. Box plot and confidence interval diamonds for number of generations and duration**

Figure 16 presents the experimental results for the each of the 9 runs for each of the 3, 6, and 9 island configurations. Stacked for each of these island configurations, the filled dots (black) represent the result for each run. Similarly for each island configuration the box plot (red) maps the 75th, 50th (median), and 25th percentiles. The diamonds (green) are centered on the sample mean with height proportional to the upper and lower bounds of a 95th level confidence interval. The number of generations it took to find an acceptable solution is much higher when the number of islands is set to three as compared to when it is set to six or nine. The variance between the replications is lower than the previous experiment but is still high, even though the only actual difference between replications is the random seed value. This suggests that random seed is a significant source of variance, one that cannot be easily removed from the experiment.

The predictor profile shown in Figure 17 is generated in JMP® using the results gathered from the experiment.



**Figure 17. Predictor profiler for supplemental experiment**

The predictor profiler shows that there is a statistically significant difference between three and six islands as well as three and nine islands. However, there is no significant difference between six and nine islands. The confidence intervals are still large in this experiment due to run-to-run variance.

## 6. Conclusions

A new, Java-based genetic algorithm implementation called JavaCat is developed. This program is used for creating time-shifted flight scenarios for Conflict Probe evaluation. CP evaluation requires scenarios that contain loss of separation events between aircraft. The CP needs to be able to predict these events accurately, and create an alert for each one it predicts. Scenarios are used to test the CP, but are limited since real-world traffic data does not contain such events. Time-shifting flights to induce these events using actual recorded traffic data allow the noise to be present in the scenarios to make them more indicative to real-world events. JavaCat finds the time-shift values needed to create such scenarios with these events by using a genetic algorithm to search through many potential solutions, finding the one that matches the original user defined scenario characteristics.

The redesigned program implements an object-oriented design, allowing for easy maintenance and development of new strategies for the GA making it highly extensible. Due to this modular design, new constraints can be easily added when new requirements are introduced. A new fitness function is developed for determining how well the solution satisfies a constraint. JavaCat also uses libraries developed by ANG-C55 for detecting separation events and popups, making the evaluation more accurate as compared to Cat, which uses an approximation method.

The island model for genetic algorithms is also implemented in JavaCat. The island model theoretically allows for larger problems to be solved due to the distributed GA it introduces. This feature is also implemented in an object-oriented fashion to allow easy additions to the program for different migration strategies that can be introduced and studied in the future. A new portion of the GA has been introduced to accommodate this feature, and to allow other population updates to occur outside of the reproduction process.

An experiment has been performed to study how certain factors affect the performance of JavaCat. The factors studied are population size, migration rate, number of islands, and ARTCC. The results suggest that the population size has a significant effect on the number of generations and duration of the GA. The number of islands and migration rate also has a significant effect on the number of generations the GA runs. In addition, the results indicate that the run-to-run variance is high, which makes it difficult to make precise conclusions on the results. The results are complicated by the mixture of workstation types, with different islands having different performances during evaluation of the population.

A second experiment evaluates how the number of islands affects the GA performance. In this experiment, the ARTCC, migration rate, and population size are fixed. Similar workstations are used to reduce the complications observed in the first experiment. This experiment focuses on a single factor and includes nine replications in order to account for natural variance in the runs. A significant difference between three and six islands and three and nine islands is detected, but the difference between six and nine islands is not significant. The run-to-run variance is still high, but on average, increasing the number of islands from three islands to six or nine islands reduces the number of generations and duration to find an acceptable solution.

Overall, JavaCat's improved implementation of the GA combined with the knowledge gained from the two experiments shall provide the FAA with a valuable tool in generating future air traffic scenarios for testing air traffic control automation.

## 7. Recommendations and Future Work

The original experiment used a mixture of 6-core and 8-core workstations in the island groupings. Mixing these workstations caused complications, which made analyzing the results more difficult. The results indicate limited correlation between generation and duration. Due to the large variance and complications from workstation mixtures, a second experiment focused on the number of islands. In future experiments, similar workstations should be used for each island. This will ensure that each island has similar performance when evaluating the population for each generation.

In the second experiment, the number of islands used for the island model of JavaCat did have an effect on the duration and number of generations needed to find an acceptable solution. An improvement in duration and number of generations is observed when increasing the number of islands from three to six, but no improvement from six to nine. A new experiment could be performed to see if there are diminishing returns by increasing the number of islands to something higher than nine, but this would require more computer resources.

Currently, JavaCat has only the reproduction strategies that are found to be optimal in Cat in (Petzinger et al., 2011). The other selection and mating strategies could be implemented easily due to JavaCat's modular design. Also, crossover currently assumes a two-parent model, where only pairs of chromosomes swap genes. Studies have shown that more than two parents can be mated together for crossover for improved results, called multi-parent recombination in (Tsutsui, Yamamura, and Higuchi, 1999) and (Eiben, Raué, and Ruttkay, 1994). This would require a multi-parent mating scheme and new crossover strategy, which again could easily be implemented in JavaCat due to its modularity.

The parameters for the GA in JavaCat all remain constant throughout the entire evolution cycle. Some studies have proposed that the probabilities of crossover and mutation should be adaptive. In adaptive genetic algorithms (AGAs) the probabilities of crossover and mutation vary based on the fitness values of the chromosomes (Srinivas and Patnaik, 1994). This could lead to a quicker convergence rate to an acceptable solution since the higher fitness chromosomes are more protected from a high mutation rate, while lower fitness chromosomes have a higher mutation rate to find a better solution. The mutation portion of JavaCat would have to be updated to follow a similar object-oriented pattern to selection, mating, and crossover to allow this change.

The fitness function of JavaCat is one attempt at improving how quickly the algorithm finds an acceptable solution. There is a significant improvement in the fitness function implemented in JavaCat compare to the legacy Cat software. The fitness function provides information to the GA on how close it is from finding a solution that satisfies each constraint. This function has not been studied intensively, and future research could investigate different fitness functions and how they can improve GA performance.

Currently, there are 39 different constraints that can be defined for the desired solution from JavaCat. The GA currently tries to solve for every constraint, with some that are weighted for more importance. An experiment and analysis should be performed to see which properties best impacts the CP's predictive performance. If the GA tries to solve for constraints that do not influence the CP's performance, unnecessary complexity is present in the system. Since JavaCat is used for making scenarios to test the CP, this is not only an important consideration in the tool's overall effectiveness but also could help in improving the GA's performance by reducing unnecessary or redundant constraints.

## 8. References

- Crowell, Andrew, Fabian, Andrew, Young, Christina M., Musialek, Ben, and Paglione, Mike M., "Evaluation of Parameter Adjustments to the En Route Automation Modernization's Conflict Probe," FAA Technical Note DOT/FAA/TC-TN12/2, Atlantic City, NJ, December 2011.
- Devore, Jay L. *Probability and Statistics for Engineering and the Sciences*, 8th Edition. Belmont, CA: Duxbury Press, 2012.
- Eiben, Agoston, Raué, P. E., and Ruttkay, Z., "Genetic algorithms with multi-parent recombination," *Parallel Problem Solving from Nature-PPSN III*, Springer Berlin Heidelberg, 1994. pp. 78-87.
- Jung, Sung H., "Queen-bee Evolution for Genetic Algorithms," *Electronics Letters*, Volume 29, Issue 6, 20 March 2003, p. 575-576.
- Mitchell, Melanie, *An Introduction to Genetic Algorithms*, Cambridge, MA: The MIT Press, 1998.
- Oaks, Robert D. and Paglione, Mike M., "Generation of Realistic Air Traffic Scenarios Based on Recorded Field Data," *Air Traffic Control Association (ATCA) 46th Annual Conference Proceedings*, Washington D.C., November 2001.
- Oaks, Robert D. and Paglione, Mike M., "Generation of Realistic Air Traffic Scenarios Using a Genetic Algorithm," *Digital Avionics Systems Conference (DASC)*, Irvine, CA, October 2002.
- Oaks, Robert D., "A Study on the Feasibility of Using a Genetic Algorithm to Generate Realistic Air Traffic Scenarios Based on Recorded Field Data," *American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference*, Monterey, CA, August 2002.
- Paglione, Mike M., Oaks, Robert D., and Ryan, Hollis F., "Methodology for Evaluating Regression Testing a Conflict Probe," *Digital Avionics Systems Conference (DASC)*, Salt Lake City, UT, October 2004.
- Paglione, Mike M., Oaks, Robert D., and Summerill, J. Scott, "Time Shifting Air Traffic Data for Quantitative Evaluation of a Conflict Probe," *American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference*, Austin, TX, August 2003.
- Paglione, Mike M., Oaks, Robert D., Ryan, Hollis F., and Summerill, J. Scott, "User Request Evaluation Tool Daily Use Time Shifting Trajectory Prediction Accuracy Degradation Study," Internal Report, William J. Hughes Technical Center, Atlantic City, NJ, December 1999.

- Paglione, Mike. M., Oaks, Robert D., and Bilimoria, Karl D., "Methodology for Generating Conflict Scenarios by Time Shifting Recorded Traffic Data," *American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference*, Denver, CO, November 2003.
- Petzinger, Bryan, Oaks, Robert D., Paglione, Mike M., and Young, Christina M., "Evaluation of a Genetic Algorithm that Modifies Air Traffic Data For Conflict Probe Testing," *30th Digital Avionics Systems Conference (DASC)*, Seattle, WA, October 2011.
- Petzinger, Bryan, Oaks, Robert D., and Nelson, Nicole, "Jaguar: Time Shifting Air Traffic Scenarios Using a Genetic Algorithm," *31st Digital Avionics Systems Conference (DASC)*, October 2012.
- Pox, George E. P., Hunter, J. Stuart, and Hunter, William G., "Comparing Two Entities: Relevant Reference Distributions, Tests and Confidence Intervals," in *Statistics for Experimenters: Design, Innovation, and Discovery*, 2nd ed. Wiley, 2005.
- Srinivas, Mandavilli., and Patnaik, Lalit. M., "Adaptive probabilities of crossover and mutation in genetic algorithms," *Systems, Man and Cybernetics*, Vol. 24, Issue 4, 1994, pp. 656-667.
- Tsutsui, Shigeyoshi, Yamamura, Masayuki, and Higuchi, Takahide, "Multi-parent Recombination with Simplex Crossover in Real Coded Genetic Algorithms," *Proceedings of the genetic and evolutionary computation conference*. Vol. 1. 1999, pp. 657-664.

## Appendix

Section 3.3.2 described the conditions that are desired for the new fitness function. Based on these conditions, a formula  $g(x)$  is devised. A logistic function is used for when the count value is close to the target range. The logistic function is shown in equation (11),

$$\frac{1}{1 + e^{x-x_0}} \quad (11)$$

where  $x_0$  is the midpoint of the sigmoid, which is defined to be 10, since the fitness shape of the fitness would satisfy the requirements. An inverse logarithm is used for count values that are far from the target range. This function is shown in equation (12),

$$\frac{1}{k \log_{10} x} \quad (12)$$

where  $k$  is a constant set to 8. This value allows a gradual decline as the distance from the target range increases, and is designed for larger values. These two functions need to be connected together with another function for the combination to be a continuous function. These functions are connected with equation (13),

$$ae^{-bx} + cx + d \quad (13)$$

where  $a$ ,  $b$ ,  $c$ , and  $d$  are all constants.

With the values for the cutoffs for each equation defined in Section 3.3.2, equation (5) is defined. To prove that it is continuous, the derivative of equation (5) is determined, and shown in equation (14).

$$g'(x) = \begin{cases} -\frac{e^{x+10}}{(e^x + e^{10})^2} & x \leq 10 \\ -abe^{-bx} + c & 10 < x < 100 \\ -\frac{\ln(10)}{8 \ln^2(x)} & x \geq 100 \end{cases} \quad (14)$$

where the constants  $a$ ,  $b$ ,  $c$ , and  $d$  approximately satisfy the requirements for the  $g(x)$  as shown in equation (6). This equation is a slightly weaker result than the initial requirement, since  $g(0) \approx 1$  and it has extreme mid-range behavior. However, the secant line from  $f(0)$  to  $f(100)$ , as defined in equation (7) is used to lift  $g(x)$ , to get a more gradual curve.

With  $g(x)$  and  $s(x)$  defined,  $f(x)$ , the fitness function for the constraint can be defined, as shown in equation (8). To test whether the function is still continuous, the derivative is determined, and is shown in equation (15).

$$f'(x) = \begin{cases} \frac{(g(x) - s(x))}{100} + \frac{x(g'(x) - s'(x))}{100} + s'(x) & x < 100 \\ g'(x) & x \geq 100 \end{cases} \quad (15)$$

This function meets all of the desired properties.