

# JAGUAR: TIME SHIFTING AIR TRAFFIC SCENARIOS USING A GENETIC ALGORITHM

Bryan Petzinger, Federal Aviation Administration, Atlantic City Int'l Airport, NJ 08405

Robert Oaks, General Dynamics Information Technology, Mays Landing, NJ 08330

Nicole Nelson, Federal Aviation Administration, Atlantic City Int'l Airport, NJ 08405

## Abstract

This paper describes the redesign of the Federal Aviation Administration's implementation of a genetic algorithm used for time shifting flights in air traffic scenarios. Time shifted scenarios are used in testing decision support tools that predict the potential loss of separation between aircraft. This paper describes the improvements that resulted when this application was redesigned and coded in Java. The improvements described in this paper include the following:

- Maintainability improved as a result of a modular design using object-oriented techniques.
- Usability improved as a result of more efficient logging techniques, configuration methods, and user interfaces.
- Quality of the solution improved as a result of a more accurate method for calculating of aircraft-to-aircraft conflicts.
- Timeliness for obtaining a solution improved as a result of using modern software engineering techniques, such as distributing the fitness function across multiple processors and caching fitness scores.

## Introduction

A conflict probe is a decision support tool used by air traffic controllers to predict aircraft-to-aircraft conflicts<sup>1</sup>. Air traffic scenarios based on recorded live data are essential for the development, testing, and evaluation of a conflict probe. When analysts create these scenarios it is necessary to modify the recorded

data in order to introduce conflicts and encounters<sup>2</sup> that do not exist in the live data.

In order to control the characteristics of the conflicts and encounters introduced into a modified air traffic scenario, the Federal Aviation Administration developed a software application that uses a genetic algorithm<sup>3</sup> (GA) to time shift individual flights in the scenario so that the distribution of aircraft-to-aircraft conflicts and encounters meets user-defined constraints.

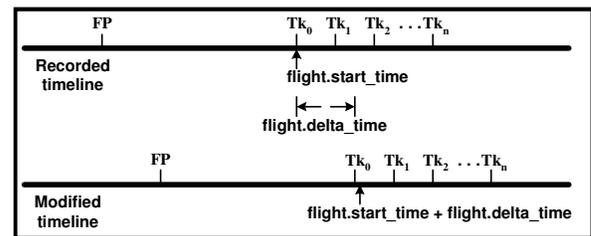


Figure 1. Time Shifting

Figure 1 graphically depicts time shifting. This figure shows the recorded timeline for a flight containing a flight plan (denoted FP) and a number of track points (denoted  $Tk_0, Tk_1, \dots, Tk_n$ ). This timeline could contain other information such as flight plan amendments, interim altitude messages, etc. The figure also shows a modified timeline with the flight's events shifted in time by a single flight specific value (denoted in the figure as  $flight.delta\_time$ ).

<sup>1</sup> A conflict is a situation in which a pair of aircraft fails to have the separation standards administered by the Federal Aviation Administration. These standards are typically five nautical miles horizontally and 1000 or 2000 feet vertically, depending on the aircraft's navigational equipment.

<sup>2</sup> An encounter is a situation used by analysts for evaluation purposes and may be defined as a situation in which a pair of aircraft come within a defined separation distance, typically 25 to 30 nautical miles horizontally and 4000 to 5000 feet vertically.

<sup>3</sup> A genetic algorithm is a search heuristic that mimics the process of natural evolution. The developers used [1], [2], [3], and [4] for this implementation.

In order for analysts to study the efficiency and accuracy of conflict probe tools used in the air traffic environment, it is necessary for scenarios to contain conflicts and encounters. These events do not occur often in recorded traffic data because air traffic controllers already resolved potential conflicts before they take place. The purpose of time shifting flight data in a scenario is to create time overlap of flights flying at similar routes. This will likely generate conflicts and encounters in the altered scenario data.

## The Original Implementation - *Cat*

The developers implemented the GA as an application (called *Cat*) in 2002 using the Oracle® Pro\*C/C++ Precompiler, which is a programming tool that enables the user to embed SQL<sup>4</sup> statements in a high level programming language, and the GNU<sup>5</sup> g++ compiler, which is a \*nix-based<sup>6</sup> C++ compiler. Although this is an Object-Oriented programming language, this implementation utilized only a few Object Oriented Design features and *Cat* was essentially a C program compiled on a C++ compiler. The developers initially used this implementation to determine whether a GA could solve the problem in a reasonable amount of time [5]. The developers then interfaced the application with existing tools and processes [6][7].

For a given problem, a GA requires a chromosome, (i.e, a potential solution that can be encoded as a bit stream) and a population (i.e, a number of potential solutions). As implemented for this problem, the GA defines a potential solution as a sequence of integer times that can be represented by the tuple:

$$\langle \Delta t_1, \Delta t_2, \dots, \Delta t_n \rangle$$

Each delta time in the tuple represents the flight specific time shift value for each flight in the scenario. A number of these potential solutions comprise the GA's population.

A GA also requires a fitness function. For this, the developers designed a function that returns a value between 0.0 and 1.0 representing how well the potential solution solves the problem. The developers designed this function so that a value of 1.0 represents a solution that meets all of the constraint bounds. The original application implemented this fitness function using socket programming techniques so that the evaluation of the fitness of each potential solution was multi-processed. See [8] for a detailed description of these techniques.

The developers modified the original application in 2008 to eliminate the embedded SQL so that it could input its data in flat files<sup>7</sup>. This was done to eliminate the application's dependence on the Oracle® Pro\*C/C++ Precompiler.

Although analysts have successfully used this application for many years, the number of iterations required for the GA to reach an acceptable solution has increased dramatically. This has been due to (1) the use of larger scenarios containing more flights over longer time periods, which caused the search space to grow, and (2) the addition of new constraints and the tightening of existing constraints, which caused the solution space to shrink. Additionally the procedural nature of the implementation was not conducive to adaptation and resulted in improvements greatly increasing the complexity of the code.

## The Island Model – *Cat Distributed*

Eventually the decision was made to re-implement *Cat* with a modular design, distributed fitness function, and several optimizations that would not be feasible to add to the original implementation. During the design phase of the re-implementation a method for distributing *Cat* was discovered that led to a prototype, and eventual upgrade of the original system.

This upgrade (called *Cat\_dist*) provided the capability to run independently on multiple computers across the network, and then periodically share their solution set with each other via a file on a shared network drive. This process is known as an

---

<sup>4</sup> SQL is an acronym for Structured Query Language, a programming language for managing data in relational database management systems.

<sup>5</sup> GNU is a recursive acronym for "GNU's Not Unix!" See <http://www.gnu.org/>.

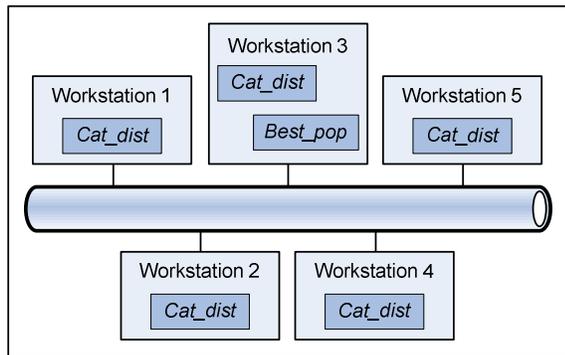
<sup>6</sup> \*nix refers to indicate all operating systems similar to Unix; specifically in this case any Linux operating system.

---

<sup>7</sup> In the context of this paper, a flat file is an ASCII-formatted file containing one record per line.

Island Model in a GA context, and is depicted graphically in Figure 2. Following the evolution metaphor; an Island Model is an implementation of a GA where several instances of the GA are run in parallel, each instance representing an island. Periodically these islands experience migration, i.e. the sharing of their solution sets. See [9] for a description of this interim solution along with a description of a Design of Experiment performed to determine the optimum values for many of the GA's parameters.

The main benefit provided by the Island Model is that as the GA explores the solution space it tends to reach a local optimum, which causes the fitness to plateau. The stochastic processes in the GA allow the algorithm to break out of this local optimal eventually, but it often takes a relatively long time. The Island Model addresses this by running multiple instances of the GA in parallel each will typically plateau at different times. By staggering the times at which plateaus occur infuses the stagnant population with a means to break through a fitness plateau [1].



**Figure 2. Example of a distributed Cat run**

Figure 2 shows an example of five instances of *Cat\_dist* being run on five workstations with the best population (denoted *Best\_pop*) being shared on Workstation 3. The five instances run independently of each other. Periodically each instance compares its current population with the saved best population. If the saved best population is better than the local population the best solution is copied and vice versa.

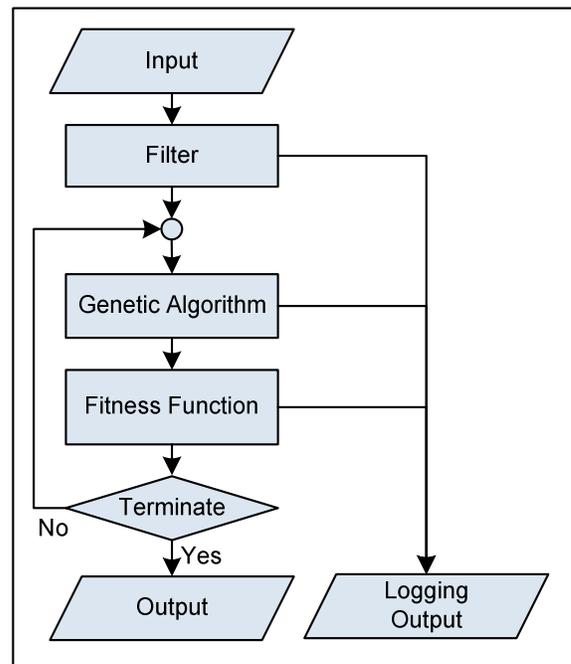
This upgrade prolonged the utility of the original application, but the developers desired a more robust

and extensible solution using modern object-oriented design techniques [10].

## The Reimplementation – *Jaguar*

The improvements implemented in the reimplementation include:

- Maintainability of the application by using a modular design based on object-oriented techniques,
- Usability of the application with implementation of more efficient logging techniques, configuration methods, and user interfaces,
- Quality of the solution with the use of a more accurate method for the calculation of aircraft-to-aircraft conflict rather than approximation, and
- Timeliness for obtaining a solution through optimizations such as caching fitness scores and filter flight pairs as well as distributing the fitness function across multiple processors.



**Figure 3. Overall Jaguar Flow**

Figure 3 depicts the flow of the program implementation. The program begins with recorded flight data as input. The data is then passed through a

filtering process to reduce the search space. The filtering process is discussed in detail later in this paper.

The GA generates delta time values for each flight in the scenario. Delta times for the first generation are either supplied as input to the program by the user, or randomly generated. Subsequent generations are created from the current generation using processes designed to mimic evolution. Each generation is evaluated by a fitness function, which rates how well each individual in the population solves the problem. An individual is considered a solution if it meets some minimum fitness value, defined by the user. If a solution has been found, the application returns the solution and exits. If a solution has not been identified, the GA is run on the new population.

*Jaguar* uses the same overall process used by *Cat* as it has proven to be effective in practice; the implementation differences result in a better, more modular design and specific optimizations and performance enhancements. These improvements are extensive enough that it was more practical to rewrite the code from scratch, rather than trying to modify the original code.

### ***Watchmaker Framework***

*Jaguar* uses The Watchmaker Framework<sup>8</sup>, which is open source software that provides an extensible, high-performance, object-oriented framework for implementing platform-independent evolutionary/genetic algorithms in Java. Watchmaker provides a solid foundation for implementing the GA with a set of modular components.

### ***Conflict Probe***

The fitness function is the component of the GA which evaluates a given solution and assigns it a value in the range 0.0-1.0. The fitness function used by *Cat* and *Jaguar* evaluates a solution by modifying the tracks of each flight in a scenario by the time shift value specific to that flight. The resulting conflict and encounter properties are evaluated and compared against the user defined constraints.

The evaluation of conflict and encounter properties is known as a Conflict Probe (CP). The accuracy of the CP is directly responsible for the accuracy of the solution generated by *Jaguar*.

Implementation of a CP is non-trivial and as a result the CP used by *Cat* is a simple approximation of what is used in the field. A key advantage of the modular design of *Jaguar* is the use of a much more accurate CP that was developed in-house for analysis. The in-house algorithm, Track Conflict Probe (TCP), results in higher quality solutions generated by *Jaguar* in comparison to *Cat*.

A drawback to utilizing the improved CP is the increase in amount of time to run. Therefore, making performance optimizations and enhancements to *Jaguar* are even more important to compensate for the extra time spent determining aircraft-to-aircraft conflicts and encounters.

Both the approximation used by *Cat* and the in-house algorithm used by *Jaguar* determine encounters and conflicts through flight by flight comparison. When determining encounters and conflicts for a unique flight pair, calculations are performed on a set of track points, paired by their common track time.

The calculations performed on a set of track points are based on the distance of the two aircraft at the current track time in comparison to the separation parameters. These same calculations are performed in both the approximation and TCP; determining the distance vertically and laterally.

The difference between the two implementations lies in what the algorithms do when they find a track point that is within the separation parameters representing an encounter or conflict, respectively. The approximation keeps track of the total number of points for an aircraft pair that are representative of an encounter or conflict. However, this logic does not include the ability to merge conflicts that is applied in the field. TCP includes the logic to merge conflicts together, i.e. if two conflicts occur in a close enough timeframe to each other they become represented by a single conflict.

In contrast to merging conflicts together, the approximation algorithm also does not consider the case where a single aircraft pair can have multiple conflicts. Since the approximation simply counts the number of points in conflict, the points could be

---

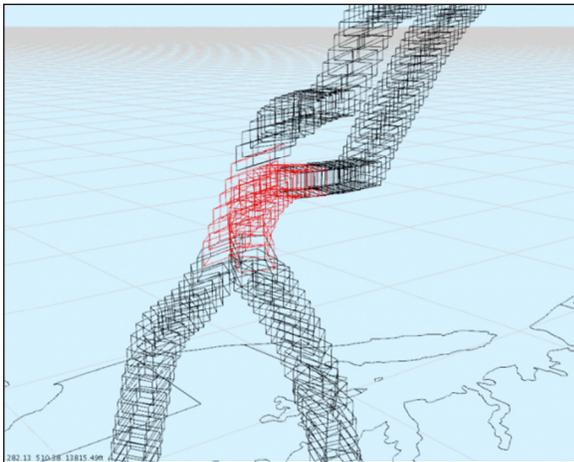
<sup>8</sup> See <http://watchmaker.uncommons.org/>

spaced anywhere throughout the time the flights overlap. TCP stores multiple conflicts when the points in conflict are outside the window of time to merge into a single conflict.

Although the mathematical computations are the same between the approximation and TCP, the results of the algorithms vary. The differences between the algorithms result in different conflict and encounter counts which skew the results generated by the GA. Results generated by a suite of in-house tools may produce a different outcome in terms of encounter and conflict counts when the approximation is applied through the execution of *Cat*. On the contrary, since *Jaguar* takes advantage of using TCP, one of the in-house tools, the GA produces the same results as later runs of the in-house tools.

### Spatial Filter

If the tracks of any two flights are overlaid, independent of time, and they never violate the minimum conflict/encounter separation standards, then no matter how they are time shifted they will not be affected the conflict/encounter properties of the scenario. With this in mind, a spatial filter was created for *Jaguar*; for each flight a bounding box is created around each pair of adjacent track points which represents the minimum conflict/encounter separation. Flights are then paired, and if their respective bounding boxes do not intersect the pair is eliminated from the search space.



**Figure 4. Bounding Boxes** - a graphical representation of how the filtering works; bounding boxes are drawn around each pair of track points for two tracks, intersections are shown in red.

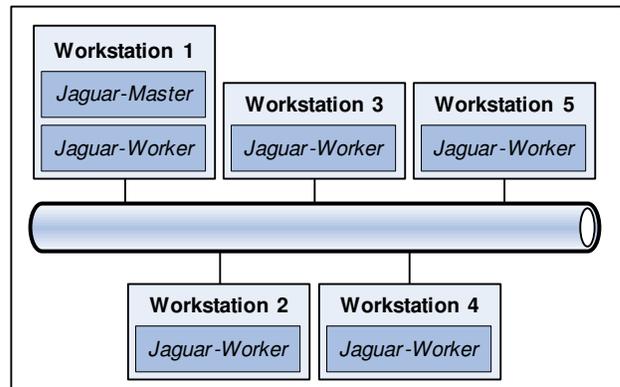
### Cache

Time shifts are relative to a pair of flights, and as the fitness converges over time the same flight pair-time shift combinations appear with increasing frequency. This property led us to implement a cache for the fitness function, which in effect trades some memory for increased speed.

*Jaguar* uses the *guava-libraries* for compute cache. This cache is keyed by an object containing two flights and their relative time offset, mapped to a Java List. If the key is contained in the map the associated value is returned, otherwise the value is computed based on the fitness function and that value is then stored in cache. The cache has a maximum number of entries, and entries are expired based on a Least Recently Used (LRU) eviction strategy, which is a cache eviction strategy that discards the least recently used items first.

### Distributed Fitness Function

*Jaguar* uses a Master-Worker architecture where the GA is run on a Master node which distributes the fitness function computation to Workers.



**Figure 5. Example of a Jaguar Run**

Figure 5 shows an example of a typical *Jaguar* run. In this example, the Jaguar-Master is running on Workstation 1, while five Jaguar-Workers are running, one on each of the workstations.

### Constraints

In the original implementations of *Cat*, the constraints were specific and not extensible. In *Jaguar*, constraints are reflectively created based on

the class names and parameters defined in an XML-structured file, which is designed to be flexible. The constraints follow the Composite Design Pattern so that they can be constructed in an elaborate tree structure with different branches weighted differently or as a basic list by adding all the constraints to a single composite constraint. Each constraint requires a constraint target, which allows for different evaluation strategies. For example, the target can be a single value or a range of values or it can be a binary evaluation (i.e., 1 if the target is met, 0 otherwise) or assigned a value based on a user defined function (i.e., linear, exponential, etc) difference from the target.

## Conclusion

The Federal Aviation Administration supports the development of decision support tools (DSTs) through the implementation of NextGen; providing new technology to accommodate the growing demand of air traffic. Analysis of DSTs is conducted through the use of recorded air traffic data. However, recorded air traffic data often does not contain conflicts and encounters because air traffic controllers have rerouted traffic to avoid such occurrences.

The goal of this paper is to describe an algorithm developed by the Federal Aviation Administration that meets user constraints for occurrences such as conflicts and encounters. The initial implementation of the algorithm solved the problem in a timely fashion until the problem became more complex. At this point, the large number of iterations required to find a solution resulted in a redesign. The new redesign solved the problem of timeliness by implementing the island model. However, the problem still existed that the implementation was not flexible. With flexibility in mind, the team used the logic of the previous implementations to develop Jaguar. Jaguar maintains speed but improves the flexibility, allowing the implementation of new constraints and concepts.

The future of Jaguar may include shifting air traffic data in search for different types of occurrences outside of the current conflict and encounter implementation. The current implementation will also undergo thorough solution time and solutions validation. With the flexibility

implemented in Jaguar the options for application have grown drastically.

## Acronyms

<b>CP</b>	Conflict Probe
<b>FP</b>	Flight Plan
<b>GA</b>	Genetic Algorithm
<b>LRU</b>	Least Recently Used
<b>SQL</b>	Structured Query Language
<b>TCP</b>	Track Conflict Probe
<b>XML</b>	Extensible Markup Language

## References

- [1] Belding, Theodore, 1995, "The Distributed Genetic Algorithm Revisited," Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, CA.
- [2] Goldberg, David E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA, Addison-Wesley.
- [3] Michalewicz, Zbigniew. 1996, *Genetic Algorithms + Data Structures = Evolution Programs, Third, Revised and Extended Edition*, New York, NY, Springer-Verlag.
- [4] Mitchell, Melanie, 1996, *An Introduction to Genetic Algorithms*, Cambridge, MA, The MIT Press.
- [5] Oaks, Robert D., August 2002, "A Study on the Feasibility of Using a Genetic Algorithm to Generate Realistic Air Traffic Scenarios Based on Recorded Field Data," AIAA-2002-4767, *American Institute of Aeronautics and Astronautics Guidance, Navigation, and Control Conference*, Monterey, CA.
- [6] Oaks, Robert, and Mike Paglione, October 2002, "Generation of Realistic Air Traffic Scenarios Using a Genetic Algorithm," *21<sup>st</sup> Digital Avionics System Conference*, Irvine, CA.
- [7] Paglione, Mike M., Robert D. Oaks, and Karl Bilimoria, November 2003, "Methodology for Generating Conflict Scenarios by Time Shifting Recorded Flight Data," *3<sup>rd</sup> Annual Aviation Technology, Integration, and Operations Technical Forum*, Denver, CO.

[8] Donahoo, Michael J., and Kenneth L. Calvert, 2001, *TCP/IP Sockets in C: Practical Guide for Programmers*, San Francisco, CA, Morgan Kaufmann Publishers.

[9] Petzinger, Bryan, Robert D. Oaks, Mike M. Paglione, Dr. Christina M. Young, 2011, "Evaluation of a Genetic Algorithm that Modifies Air Traffic Data for Conflict Probe Testing," *30<sup>th</sup> Digital Avionics System Conference*, Seattle, WA, October.

[10] Freeman, Eric, and Elisabeth Freeman, 2004, *Head First Design Patterns*, Sebastopol, CA O'Reilly Media, Inc.

## **Email Addresses**

Bryan Petzinger – [bryan.petzinger@faa.gov](mailto:bryan.petzinger@faa.gov)

Robert Oaks – [robert.oaks@gdit.com](mailto:robert.oaks@gdit.com)

Nicole Nelson – [nicole.nelson@faa.gov](mailto:nicole.nelson@faa.gov)

*31st Digital Avionics Systems Conference  
October 14-18, 2012*