AFRL-IF-RS-TR-2001-287
Final Technical Report
January 2002

# EVOLVING COMMAND, CONTROL, COMMUNICATIONS, COMPUTERS, AND INTELLIGENCE, SURVEILLANCE AND RECONNAISSANCE (C4ISR) DECISIONS

Natural Selection, Incorporated

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

20020308 053

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-287 has been reviewed and is approved for publication.

APPROVED:

STEVEN M. ALEXANDER, 1st Lt., USAF
Project Engineer

FOR THE DIRECTOR:

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFSB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

| REPORT DOCUMENTATION PAGE | Form Approved OMB No. 0704-0188 |
|---|---|

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | JANUARY 2002 | Final Apr 00 - Apr 01 |

**4. TITLE AND SUBTITLE**
EVOLVING COMMAND, CONTROL, COMMUNICATIONS, COMPUTERS, AND INTELLIGENCE, SURVEILLANCE AND RECONNAISSANCE (C4ISR) DECISIONS

**5. FUNDING NUMBERS**
C - F30602-00-C-0048
PE - 62702F
PR - 459S
TA - BA
WU - 01

**6. AUTHOR(S)**
V. William Porto and Lawrence Fogel

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Natural Selection, Incorporated
3333 North Torrey Pines Court
Suite 200
La Jolla California 92037

**8. PERFORMING ORGANIZATION REPORT NUMBER**
N/A

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Air Force Research Laboratory/IFSB
525 Brooks Road
Rome New York 13441-4505

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
AFRL-IF-RS-TR-2001-287

**11. SUPPLEMENTARY NOTES**
Air Force Research Laboratory Project Engineer: Steven M. Alexander,1st Lt., USAF/IFSB/(315) 330-4304

**12a. DISTRIBUTION AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*
This report details the investigation, development, and prototype demonstration of evolving C4ISR capabilities of aircraft in an arbitrary environment. Natural Selection, Inc. (NSI) was tasked to show how the application of the Valuated State Space Approach and evolutionary programming (EP) could be used to produce optimal behavioral plans, and develop a set of requirements for implementing this capability within an existing (or near-future) simulation engine. As a step toward the creation of an actual prototype system, a small prototype optimal planning mechanism was created using computationally intelligent techniques. A long-term goal is to develop a generalized, extensible evolutionary software system which can efficiently and effectively encompass all desired hierarchical levels of planning for air combat entities, and can be used as a tactical decision aid or training mechanism.

**14. SUBJECT TERMS**
Valuated State Space, Evolutionary Programming, Evolutionary Computation, Genetic Algorithms, C4ISR Decision Support, Cognitive Process Modeling, Optimization

**15. NUMBER OF PAGES**
60

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

## Table of Contents


| | | |
|---|---|---|
| 1.0 | Introduction | 1 |
| 1.1 | Objectives | 1 |
| 1.2 | Tasks | 2 |
| 2.0 | Overview and Rationale | 3 |
| 3.0 | Background | 4 |
| 4.0 | Review of Existing Simulation Engines | 8 |
| 5.0 | Technical Considerations | 14 |
| 6.0 | Proposed Software Implementation | 17 |
| 6.1 | Robust Software Design | 18 |
| 6.2 | Prediction Mechanisms | 20 |
| 6.3 | Mutation Operators | 21 |
| 6.4 | Scoring Function | 23 |
| 7.0 | Prototype Simulation and Sample Experiments | 23 |
| 7.1 | Prototype Simulation Scoring Function Components and Implementation | 24 |
| | 7.1.1 Damage Assessment and Survival/Kill Probabilities | 24 |
| | 7.1.2 Resource Utilization | 26 |
| | 7.1.3 Strategic Influence | 26 |
| | 7.1.4 Constraints | 29 |
| 7.2 | Test Experiments | 30 |
| 8.0 | Conclusions and Recommendations | 37 |
| 8.1 | EP/Simulator Problem-Specific Recommendations | 38 |
| 8.2 | Software Extensions | 38 |
| 8.3 | Hardware Requirements | 39 |
| 8.4 | Testing | 39 |
| References | | 41 |
| Appendix A: | Overview of Evolutionary Programming | 43 |
| Appendix B: | The Valuated State Space Approach | 46 |

## List of Figures


| | | |
|---|---|---|
| Figure 1. | Flow Diagram of EP/ModSAF processing | 6 |
| Figure 2. | Processing diagram of EP/JANUS software system | 8 |
| Figure 3. | Example of a temporally linked task frame set for controlling ModSAF vehicles containing 4 task frames. | 15 |
| Figure 3. | Processing diagram of EP/ Generic simulator software system. | 18 |
| Figure 4. | Initial plan for test scenario after on generation of evolution | 31 |
| Figure 5. | Evolved plans for basic scenario after 17 generations. | 32 |
| Figure 6. | Evolved plans for basic scenario after 44 generations. | 33 |
| Figure 7. | Initial scenario with one rectangular no-fly zone added. | 34 |
| Figure 8. | Modified scenario with one rectangular no-fly zone, after 9 evolutionary generations. | 35 |
| Figure 9. | Modified scenario with one rectangular no-fly zone, after 206 evolutionary generations. | 36 |

## 1.0 Introduction

This report details the investigation, development, and prototype demonstration of evolving C4ISR capabilities for the control of aircraft in an arbitrary environment. Natural Selection, Inc. (NSI) was tasked to show how the application of the Valuated State Space Approach and evolutionary programming (EP) could be used to produce optimal behavioral plans, and develop a set of requirements for implementing this capability within an existing (or near-future) simulation engine. As a step toward the creation of an actual prototype system, a small prototype optimal planning mechanism was created using computationally intelligent techniques. A long-term goal is to develop a generalized, extensible evolutionary software system which can efficiently and effectively encompass all desired hierarchical levels of planning for air combat entities, and can be used as a tactical decision aid or training mechanism.

## 1.1 Objectives

Accomplishing this involved achieving a number of objectives, each leading to the development of a useful software tool capable of demonstrating the aforementioned capabilities. These objectives are described below:

1) Survey and examine the range of simulation engines currently available in the simulation and modeling world for potential use for C4ISR support. Identify the state of the art with respect to non-rule-based intelligently interactive combat simulation and how existing deficiencies impact the capabilities of these systems.

2) Define the necessary requirements for the design, development and implementation of software which uses evolutionary programming and a suitable simulation engine for the optimal planning of air combat vehicles in a specified environment. Specifically, determine all necessary components required to interface an existing simulation engine, as well as the hardware/software, and communications protocols required to support such a system.

3) Provide recommendations for implementing such a system, the potential tradeoffs and possible uses for this automated planning system. Identify a course of action that will facilitate adaptation of a chosen simulation engine such that it can be used to generate optimal, truly intelligently interactive, air combat plans.

All of these objectives have been met successfully as a result of work performed during this effort.

## 1.2 Tasks

There were three global tasks in this project. In the first task, a survey of Air Force C4ISR and related training activities was conducted. This task consisted of two interrelated components. The first of these entailed identifying the aspects of air combat planning operations currently in place in the Air Force, and assessing which of these aspects would benefit from the application of evolutionary optimization techniques to derive optimal plans. The second part of this task involved reviewing the set of existing simulation models currently being used by the military forces, and identifying those that are suitable for use in an evolutionary planning framework. These two parts are highly coupled as the availability and choice of the simulation engine is critical to the success of the software system. In turn, the aspects of air combat planning define what parameters of concern and level of simulation are necessary to generate C4ISR decision aids.

The second phase of this project involved development of a small test/prototype program to generate optimal plans for aircraft entities within an environment with typical constraints and threats. For this prototype program, modeled threats consisted of SAM sites. Additionally, a no-fly zone was included to represent typical combat environments wherein specific countries or areas therein prevent transit through parts of their airspace.

The third and last task was centered around developing recommendations, requirements, and a general plan of action for implementing an autonomous evolutionary planner for planning air combat operations. Such a combat planner could be used as a tactical aid as well as in a training environment wherein combat personnel could train against a calibratable, intelligently interactive adversary.

In addition to these tasks, a paper and presentation were made at the Aerospace SPIE conference held in Orlando, Florida. The purpose of this presentation was to inform the audience on the nature of the current state of the art as well as provide them with a small demonstration of the possibility of generating air combat plans using evolutionary computation.

This report is comprised of sections describing 2) overview and rationale, 3) review and analyses of applicable simulation engines, 4) technical considerations pertinent to the problem, 5) potential software implementation, 6) prototype simulation software, and 7) conclusions and recommendations. Appendix A contains an overview and algorithmic details of evolutionary programming.

## 2.0 Overview and Rationale

Creating optimal intelligently interactive behaviors for both C4ISR and air combat in general is a formidable task. There are multiple interacting aspects, all of which must be taken into account simultaneously. By definition, the mechanisms involved must operate without human intervention. They should also be fully adaptive to realistic dynamic environments and not just static simulations. In typical applications, available information is limited by visibility and communication capability. In addition, there is often a difference between perceived reality and actual truth due to the information filtering processes inherent in a typical battlefield environment.

In the vast majority of previous research studies, planning is undertaken for a single vehicle at a time. Multiple vehicle scenarios are either ignored or treated as a set of separately controlled individual vehicles. However, in combat, autonomous vehicles must act in concort with other members of the same team to enhance the effectiveness of the cohesive whole. Coordination of efforts is an exceedingly important mechanism. Without it aggregate behaviors cannot be truly exploited, leading to suboptimal solutions. This results in the expenditure of more resources to solve any given problem than should be necessary.

The search space of possible alternative paths and combinations thereof is immense. Because of this, exhaustive analysis is, in most cases, impractical. What is needed is a way to efficiently search the space of possible tactics to find one of sufficient value in time for it to be useful. When exhaustive analysis is clearly impossible, a common approach is to turn to heuristics. However, heuristics often prove useful only under certain specialized circumstance.

Another approach is to attempt to solve the problem using mathematically tractable approximations. This typically generates the right solution to the wrong problem. For example, steepest descent is prone to failure if there are multiple optima. Linear programming is often used even when the constraints are known to be nonlinear. Complex problems are decomposed into simple ones so that these can be treated separately for convenience. The aggregate of these local optimizations leads to a global optimum only if the component problems are truly independent, which is rarely the case. Statistical procedures generally presume stationarity, but the real-world is nonstationary. In fact, these and other heuristics are rules. If the rules that solve the specific problem are known *a priori*, the best procedure is to use them. If they are not known the rules chosen relative to a previous problem may often stand in the way of finding a better solution to the current problem.

Until recently, simulated autonomous entities were either semi-automated (where an operator is required to exert some degree of supervisory control over the vehicles), or constructively controlled using heuristics (defined *a priori* by a 'knowledgeable expert'). Requiring human operators to interact with 'semi-intelligent' autonomous entities is both expensive and inefficient. Aside from the required personnel costs, the resulting simulations are highly dependent upon the skill of the human involved. In any case, these entities are not truly autonomous, as the operator is a necessary part of the control process.

Heuristic control can be equally problematic. Increasing intelligence levels through application of heuristics quickly makes these vehicles just 'smart' enough to accomplish routine tasks but generates easily predictable (and defeatable) behaviors. The heuristics employed always reflect a human's opinion, which is not necessarily optimal for more than one circumstance. In addition, heuristics are fixed rules that are not adaptable to realistic dynamic environments.

In contrast, the evolutionary programming algorithm (Fogel 1964, Fogel et al., 1966, Fogel 1995) is a most general optimization technique. The *only* "rules" are problem-independent iterative mutation and selection. Those components of randomness that are found to be of value are retained to benefit further generations of solutions. Evolutionary programming is an inherently elegant and potent technique simulating the mechanisms of phenotypic evolution: iterative mutation and selection to generate organisms that tend toward optimal behavior with regard to their environment and given payoff function. It is not limited by prescribed rules or heuristics, therein making it most suitable for optimizing path plans for autonomous vehicles, especially in dynamic environments.

Prior to this research artificial intelligence systems failed to provide truly intelligent computer simulated vehicles. In contrast, results of research at Natural Selection, Inc. using evolutionary programming to control simulated entities have shown that intelligently interactive behaviors can be evolved [Porto and Fogel, 1997].

## 3.0 Background

Evolutionary programming [Fogel et al. 1966, Fogel 1995] can be used to generate optimal behavior that controls the simulated entities for a predetermined amount of time into the future. An objective function in the form of a Valuated State Space and normalization function [L. Fogel 1995] scores each prospective behavioral plan. Re-optimization of this information is performed periodically using the latest available information about the sensed environment (i.e., number and positions of entities, terrain, damage assessment, etc.). This results in the generation of dynamically adaptive behavioral plans. By predicting the positions of all simulated vehicles, behaviors which anticipate actions of enemy, friendly (and potentially, neutral) team members can be evolved.

Natural Selection, Inc. has successfully demonstrated the use of evolutionary computation for generating behavioral plans for simulated combat entities. This evolutionary process has been combined with two different simulation engines, ModSAF and more recently, JANUS. The first research produced an evolutionary program implemented around the simulation engine ModSAF, that provided a basis for optimal low-level control of autonomous vehicles [Porto and Fogel, 1996]. Evolutionary programming was used to optimally control computer generated forces (CGFs) on two opposing teams in highly dynamic environments. Tactical courses of action were learned adaptively for individual vehicles as well as for higher-level aggregations (i.e., platoons). Evolutionary updates of behavioral plans incorporated dynamic changes in the developing situation and the sensed environment.

The version of this integrated EP/ModSAF software (as developed for STRICOM) plans high level behaviors/paths in view of the given scoring function that includes several factors. The process acts by interrupting the normal processing flow of the simulation engine, essentially stealing CPU cycles to perform the optimization tasking. First, the simulation engine (ModSAF) is temporally stopped to obtain a snapshot of the current environment. Next, while the simulation is paused, the evolutionary program uses the current physical environment (i.e., sensed enemy positions, terrain features, weather, etc.) to predict the actions of the opponent forces into the near future. Past behaviors can be used to approximate state information relative to purpose and intent. With this information, the evolutionary program generates a set of 'parent' solutions to the planning problem, given the desired mission goals (e.g., survival, task accomplishment, minimization of fuel used, etc.). From these 'parent' solutions, 'offspring' solutions are generated which are variants of the parent solutions.

After scoring all of these solutions, the least-fit are culled (finite resources limit the number of retained solutions) and the process is iterated by the generation of new 'offspring' solutions. When a sufficient number of iterative cycles or specified time has elapsed, the best of these solutions is input back into the simulation engine that is awakened from its paused state. The evolved solutions are then used by the simulation engine as plans for the specified vehicles over the desired time period. The process is repeated indefinitely or until the desired goals are accomplished. Figure 1 below represents this process flow pictorially.

During each EP/ModSAF cycle the (modified) graphical user interface within ModSAF displays the paths generated throughout the evolutionary process. However, the vehicles do not actually move on the screen during this planning phase – this is only a graphical guide for ascertaining (visual) fitness of the solution set. When restarted, ModSAF can be used to display the new behavioral paths within its own graphical framework.

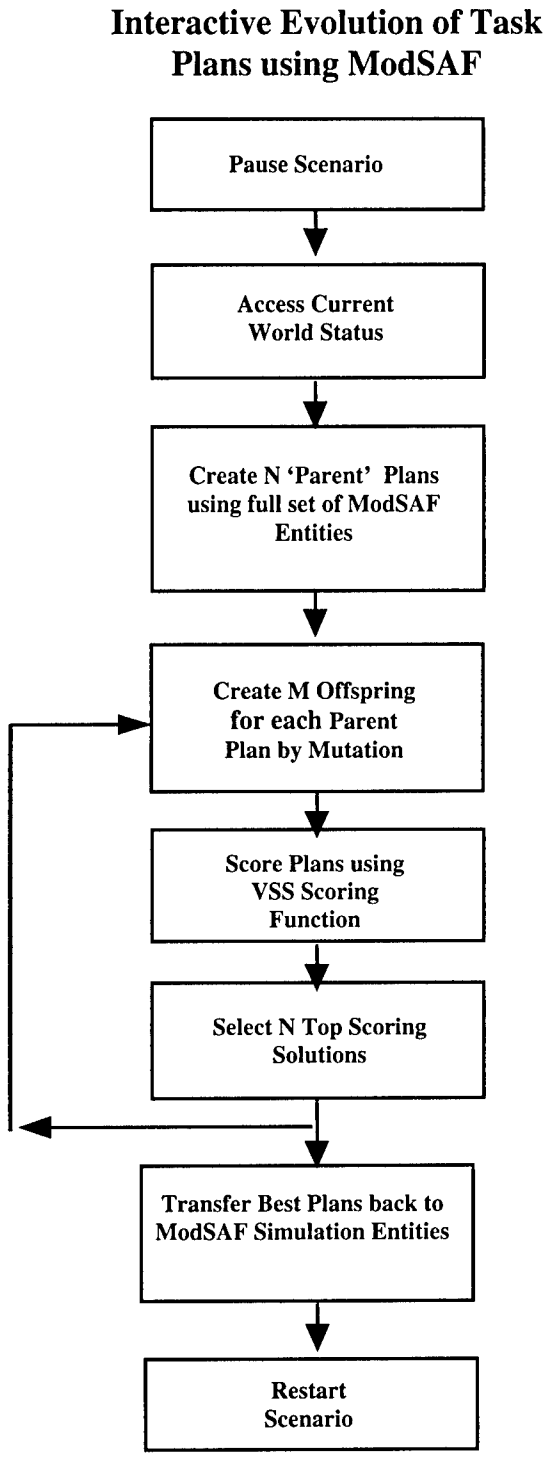


**Figure 1**. Flow diagram of EP / ModSAF processing.

This preliminary study using ModSAF resulted in a second research project in which JANUS was used as a simulation engine. In addition to the major differences between JANUS and ModSAF, several other important improvements were made. These include

1. providing for additional robustness against several failure modes
2. incorporating additional fidelity to the internal evolutionary models.
3. implementing the software in a true parallel processing framework.
4. refining the probabilistic $P_{kill}/P_{survival}$ measures
5. incorporating several soft-factors into the simulation (e.g., motivation)

Instead of 'stealing' CPU cycles directly from the single processor running ModSAF, the JANUS-based approach uses parallel processing mechanisms to permit offloading evolutionary computation processing tasks if/as necessary. This is an important consideration since when used in a training mode, any delay in processing directly effects the end-user's leading to dissatisfaction. A client-server mechanism was developed in which separate data servers act as information messengers only, relieving the main CPU running the simulation engine of both heavy EP-related computational costs as well as making it less prone to interruption failure due to interruptions. Standard internet sockets and protocols are used to transmit information to and from each data client program. This design also isolates the individual components from each other, allowing them to act asynchronously and independently from each other. Thus, if one client is terminated for whatever reason, other clients are not affected.

By revising the implementation, it allows the evolutionary programming software to run as a completely separate process on virtually any computer (provided said computer has standard internet socket communication capabilities and sufficient memory to run a basic EP client process). In fact, the design allows for any number of independent evolutionary programs to be run on one or more computers. Thus, in true parallel processing fashion, these independent processes can be instantiated based solely upon the available computational resources (up to host platform communication socket limits). Figure 2 indicates the functional components and relationships between each component. Note that although only one EP client process is shown in the figure, any number of EP client processes can interface independently with the EP server process. A separate graphical user interface (GUI) is provided for each EP client process to facilitate access to and control of user parameter selections.

Finally, the new implementation incorporates a set of soft-factors which are important to the measure of success on a battlefield. Intelligence level is controlled through adjusting the number of alternative courses of action reviewed during each evolutionary epoch (time between updates of behavioral plans to the simulation engine). This is accomplished through adjustment of population dynamics (e.g., number of parents, number of offspring per parent, number of generations between epochs). Similarly, measures of motivation, physical and mental ability, and risk taking propensity are also modeled.

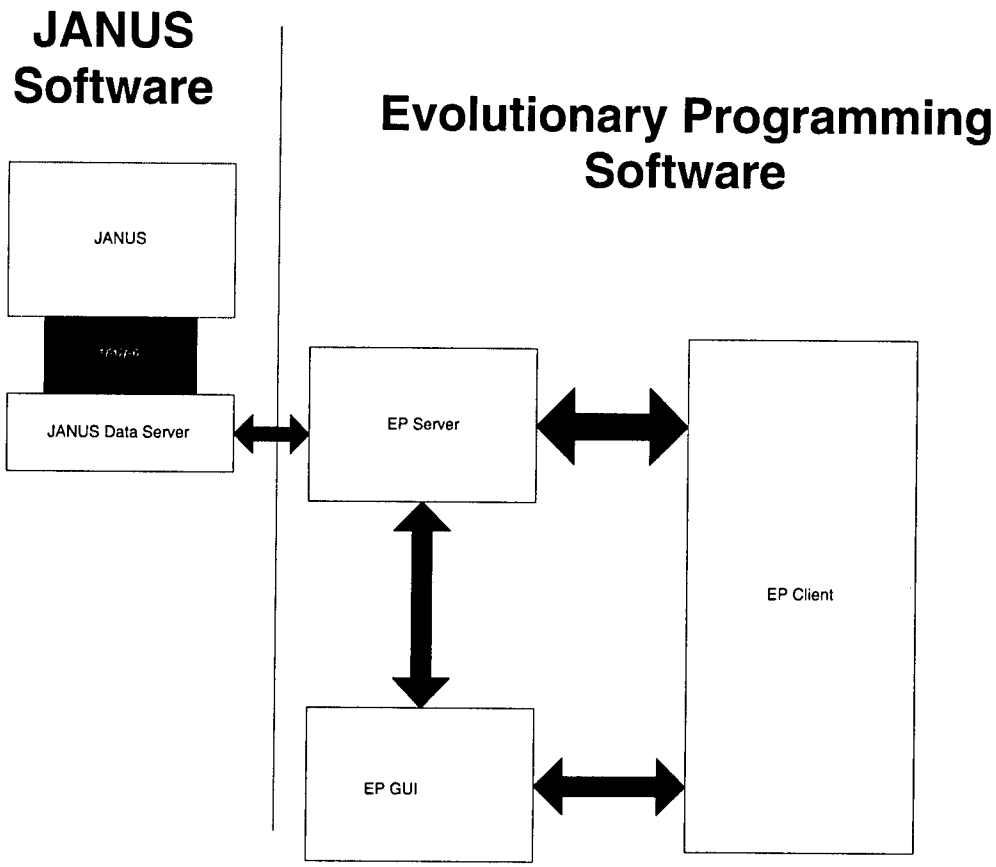


**Figure 2**. Processing diagram of EP / JANUS software system.

## 4.0 Review of Existing Simulation Engines

A survey was conducted of Air Force C4ISR and other training simulations in order to identify those that can be enhanced by incorporating an intelligently interactive opposing force. This requires that:

1. The simulation model entities at a suitable level of complexity and abstraction, midway between engineering simulations (will the missile hit its target), and campaign level simulations (primarily concerned with attrition, sustainment, and the deterrence of escalation).

2. The simulation can be adapted to operate without human intervention, for having a man-in-the-loop makes the simulation uncalibratable and adds excessive lag.

3. The simulation operates on an available computer platform at least as fast as real-time, that is, it can generate outcomes of any particular maneuver/tactic fast enough to search for alternative courses of action before a final solution is required. If this is not the case, the evolutionary program (or any other

optimization method for that matter) cannot execute a sufficient number of generations to discover significantly better maneuvers/tactics.

4. The simulation is written in such a manner that an external program can access both the state information that defines possible courses of action as well as the measures that appear in the mission as stated as the form of a Valuated State Space™. Here combat effectiveness is judged in terms of a number of parameters such as, targets killed by priority, their timeliness, munitions expended, acceptable degree of vulnerability, amount of force at risk, as well as lost opportunity costs. Success of the mission therefore involves broad considerations that go beyond simply time-on-target. Direct access to the code is not necessary if the simulation engine provides an existing convenient interface to obtain these parameters and input new optimized information back to it.

5. The simulation must not be a legacy system, classified, or one that is still in development. To create a truly usable evolutionary optimization system it must be implemented on an existing system that is also expected to be utilized in the relevant future. Legacy systems are, most often, inappropriate as they are generally scheduled for replacement in the near future. In order to minimize development costs and maximize flexibility for use on a range of Air Force projects, use of a classified simulation system is not realistic.

6. The program manager in charge of the simulation must be willing to provide a copy that can be modified. It was found that program managers offer a variety of reasons for not allowing this to occur. Access to source code is highly important to the success of this project although it often tends to present a problem with many simulation development teams. In fact, the majority of people contacted with regard to their simulation engines would either not release the source code or required their own development teams to place the required hooks into their simulations. Given the cost (both time and money) of using a third party as an intermediary, these simulation engines are not economically feasible for our use.

A large number of simulation systems were reviewed for use as part of an evolutionary optimization system. Contact was made with Bob Mayne regarding AFMSS/CLOAR, a mission planning segment that is federated with Joint Interim Mission Model (JIMM). A number of pilots select the mission routing/scheduling plan, viewing it on AFMSS with CLOAR providing appropriate intelligence information. This portion of the simulation is classified. After the plans have been established they are executed on JIMM taking from one minute to an hour depending upon the complexity of the required simulation. Obviously this simulation is unsuitable. JIMM is much like Suppressor (a non interactive simulation) written in C++ for Unix but it is reported to be difficult to get up and running and even then not necessarily reliable.

JIMM is the merger of key capabilities of the Air Forces Suppressor model into the Navy's Simulated Warfare Environment Generator (SWEG) model. It is primarily a mission level analysis model but can also be used as a scenario generator for testing and training purposes. It is a Distributed Interactive Simulation (DIS), event-stepped, object-

oriented, general purpose conflict simulation and may be used as a missile prototype for a next generation mission model. It can participate in a network with other simulations and man-in-the-loop systems and represent multi-sided conflicts. Based upon the information that this simulation is frequently unreliable and difficult to instantiate the software on a new system, this simulation package is not suitable for our use.

Col. Thomas Ardern reported that all the training combat simulations used at Maxwell Air Force Base are manned on both sides. Therefore they are unsuitable.

Mike Davis, program manager for AWSIN referenced Jim Daly of Raytheon. He, in turn, reported that the OPFOR within AWSIN is manually operated by personnel who attempt to simulate enemy behavior in a realistic manner. There is no measure of their degree of optimality. The blue force has even less freedom, in that it must follow the given ATO and doctrine. AWSIN is the official Air Force Theater-Level war gaming model/simulation for training. It can include day and night operations under limited weather conditions over smooth earth. This two-sided simulation includes aircraft, airbases, surface-to-air missiles, short-range air defense systems, ships and radar sites. Measures are ordinarily taken with respect to the success of individual aircraft, munitions consumption and the manner in which the scenario develops. There is no calibration of the friendly force demonstrated capability. This simulation can treat air attack, air defense, air interdiction, air-land battle, air superiority, air traffic control as well as command and control operations, but is not suitable for enclosing in an evolutionary framework. AWSIM can be coupled with JQUAD, a joint electronic warfare model that combines other intelligence models. It is a two-sided game with probability driven random variables. It is interactive because it requires human controllers operating in a complex language in instructions and formats. For example, there are 36 different kinds of missions, each with presets.

Dave Hoagland of WPAFB referenced their current development of Combat Automation Requirements test bed (CART) within the Air Force Research Laboratory, Human Effectiveness Directorate. Within constructive simulations, sensitivity analyses are usually conducted on key subsystem attributes by selectively varying their levels and measuring the results in terms of mission performance. Unfortunately, consideration of the crew interface is usually avoided in such requirement generation efforts. Here, man-in-the-loop is used partly because of an inability to properly model human behavior in an interactive setting. CART intends to advance the state of the art in human modeling. This simulation is in C++ and is HLA compatible, however, it is not suitable for an evolutionary framework because of the man-in-the-loop requirement.

Major Scott Fox reported on Thunder, a stochastic, two-sided analytical simulation of campaign-level military operations. The simulation was developed in the 1980's to examine the utility and effectiveness of air and space power in the context of theater-level joint warfare. Thunder is written in SINSCRIPT and is the predecessor to STORM to take advantage of "new technology and simulation, algorithms, and processing to enhance analytical utility". In essence the mission is specified, the simulation runs faster than real time but is still slow because of the duration and complexity of theater operations. This rule-based simulation includes the variability found in actual combat, as

a result each initial condition scenario is processed twenty times in order to determine the most likely outcome and other statistics. It offers certain reasonable measures of effectiveness, but these are neither weighted nor combined. The simulation can work on a SUN workstation after being compiled in C. STORM is a future simulation that will be neither intelligently interactive nor calibrated. It is being developed in support of JWARS. It is not a candidate for the evolutionary framework because it is currently under development. However, if in the future this simulation becomes a reality, our design will be capable of interfacing with it.

Lt. Col. Laurie Talbot monitors JCAT for the Air Force Agency for Modeling and Simulation. This is a man-in-the-loop, *n*-player game, wherein each player can have a different weapon system. It can operate in real time or up to ten times real time with sub-routines directing weapons to targets in a preset manner. The program is in C++ and has already programmed measures of effectiveness, but no way to combine these or to determine when the combat is over. Being man-in-the-loop it is unsuitable for an evolutionary framework.

Technical information was also obtained regarding the Advanced Tactical Combat Model (ATCOM) developed by Boeing, Philadelphia. This is a stochastic, force-on-force constructive combat simulation. The term "constructive" means that the computer constructs the outcome from the initial conditions and force direction, this as opposed to the virtual simulation wherein the man-in-the-loop determines the outcome by estimating various outcomes during the game. In ATCOM, interactions between the opposing forces are simulated over a digital terrain, taking into account weapons and sensor performance, combat vehicle characteristics, weather and tactics. This simulation is man-in-the-loop, requiring gamers/tacticians to make tactical inputs during the combat simulation. It provides a high fidelity simulation of detection and communications with six degree of freedom dynamic models of the aircraft and fundamental equations of motion for the ground vehicles. Again, because this simulation requires man-in-the-loop, it is not suitable for an evolutionary framework.

The Extended Air Defense System (EADSIM) was developed by Teledyne-Brown. They own the software and will not allow changes to be made unless they make them. The Extended Air Defense Test bed is an outgrowth of EADSIM that is reported to be flexible, difficult to operate and very costly. Here the subject is too far from Air Force C4ISR and the program cannot be readily modified to insert the hooks to access the required parameters.

The Virtual Strike Warfare Environment 7 is intended to support the next generation of Joint Strike Fighter. The intent is to develop a prototype engineering environment and process for evaluating Joint Strike Fighter Mission Effectiveness. There are many participating organizations but this simulation is still under development and therefore not suitable for the present purpose.

BAE Systems produced a Tactical Strike Coordination Manager (TSCM), a joint force level planing tool that enables rapid execution of the ATO and near real time monitoring of ground and airborne tracks. It displays the theater of operation and selectively permits

overlay of targets and threats, the friendly Order of Battle, zones developed manually or from the Airspace Control Order (ACO) and weather. Although developed primarily for Navy use, planners can use this to develop integrated strike packages for Air Force or coalition tactical aircraft, Tomahawk land attack missiles and UAV's, or to process the ATO for execution. This simulation was demonstrated to Lt. Steven Alexander at the BAE facility in Rancho Bernardo. It was considered a strong contender to the modification of JANUS to include air capabilities. TSCM is understood to be the property of BAE Systems. Their cooperation would be essential to incorporating the required hooks and providing an evolutionary framework that would optimize the tactics. In conversations with BAE personnel they were adamant that their source code remain in house and only their developers would be allowed to insert the required interface hooks to the parameters of interest. Hence, due to the additional cost of this third-party interface, this simulation is deemed unsuitable.

The Air Campaign Planning Tool (ACPT) was developed under DARPA, Air Force, and Navy sponsorship to support the Joint Forces Air Component Commander (JFACC) in the development of CNOOPS. It consists of a large collection of routines that are called upon in the process of planning. Situation Analysis Tools provide access to relevant information including intelligence reports and help the campaign planner build a rationale for the air strategy based on national political theater and military objectives as presented by NCA. Although this takes into account enemy capabilities, it is not interactive in real time.

Natural Selection, Inc. has been under contract to DARPA to provide automated decision support to JANUS. Before this, a variety of specially qualified experts were required to exercise this training program, each expert directing the tanks, artillery, and other specialized weapons so that the students could learn to operate the friendly force in a realistic manner. This is costly and it is often difficult to have the right experts at the time and place required. The automated capability has now been demonstrated. The mission of each "expert" is specified, the evolutionary program optimizes tactics given the initial conditions and available capabilities. The speed of performance depends on the number of platforms (up to several thousand), the extent of the mission statement (number of parameters and level of detain of the Valuated State Space™ and normalizing function), and the population size/number of generations used in the evolutionary program. This capability was demonstrated to Lt. Steven Alexander in terms of ground warfare, then later in terms of inserting air combat aircraft. JANUS is indeed a flexible simulation that has been used extensively for Army training and can be readily modified to simulate air land battles. There are numerous entity types within JANUS and it is easily extensible to additional entities. However, the plans executed within JANUS are typically point-to-point moves wherein no turning information (kinematics, loading, etc) are currently modeled. Although this could be a point of concern in aircraft combat simulation, JANUS could be modified to incorporate these parameters. It was therefore recommended that this simulation be used for our development efforts.

Certain other simulations were considered to be unworthy of further attention. Col. Forrest Crain, Director of DMSO, indicated that his agency has jurisdiction over some

1000 different combat simulations. Many of these are still undocumented in their files. There is no current summary of their usage rate or degree of calibration. Incidentally, he pointed out that JWAR/JSIM, the most recent joint simulation will cost over one billion dollars. Such an expenditure cannot be justified in terms of its projected worth. Thus far it has not met its requirements and may receive no further funding.

Another possible candidate simulation is ModSAF (Modular Semi-Automated Forces). ModSAF models the physical processes of a number of entities simultaneously. ModSAF incorporates heuristics for emulating low-level vehicle behaviors. These heuristics are used to manipulate vehicles at various hierarchical levels (unit, platoon, etc.) while adhering to realistic physical constraints. A computer operator (or automated system) is used to generate high-level behaviors, leaving the detailed low-level modeling and planning to the imbedded heuristics inside ModSAF. This realistically approximates a chain of command, where command leaders provide high-level instructions and let the smaller details be handled in some routine manner[1].

Although ModSAF is currently at the end of its development cycle (soon to be replaced by OneSAF), it is designed to model a wide variety of entities in land, sea, and air environments. ModSAF (as well as most other CGF simulation engines) effects control of entities via parameterized structures that specify actions for discrete periods of time. In terms of game theory, this temporal aspect becomes critically important as behaviors for each agent rely upon a finite-time period for planning as opposed to the usual open-ended approach typically used in robot path planning. Instead of describing paths as points in *N*-space, a set of temporally linked, discrete-time task frame segments are used to specify the duration, velocity vector, and all other pertinent parameters. An entity is controlled by instructing it to perform a sequence of desired tasks, each for a prescribed period of time. Hence, predictive planning is driven by time, not position. Each team plans for a specified 'lookahead' time into the future. The sum of all task frame time durations in a behavioral plan is a constant, set equal to this 'lookahead' time.

Previously, ModSAF v2.0 and v3.0 were used (as described above) as part of an evolutionary planner. Both versions exhibited suitable traits but were prone to failure due to unreliable execution of input plans. The new OneSAF prototype is currently being tested as a beta version. Natural Selection, Inc. has obtained a copy of this simulator and will be evaluating it over the course of the next few months. Additional hardware is required to operate OneSAF as it requires 256MB RAM memory, over a Gigabyte of storage space, and the most recent version of Linux to operate. A version has also been developed which should run under Windows NT4.0 if additional software components (i.e., X windows, communications protocol upgrades, etc.) are also installed. Due to the unreliability of WindowsNT as a stable multi-tasking O/S, Linux is obviously the preferred choice if OneSAF is selected for use in the future.

It is important to note that both ModSAF and OneSAF do not allow the user to identically replicate results of a scenario, hence some other method of obtaining statistically

[1] A detailed description of the implementation of this algorithmic approach can be found in [Porto and Fogel, 1996].

significant results must be used. This may require replication of an experiment many times to account for potential statistical variance in the underlying simulation engine[2].

## 5.0 Technical Considerations

Since the choice of simulation engine is crucial to the success of the evolutionary optimization software, several technical aspects must be considered. The basic requirements have been described above (i.e., the simulator must run at a suitable level of abstraction, hooks into the required parameters must be available, documentation must be available, etc.). However, many other aspects must be considered as part of optimization of Air Force C4ISR tasking.

The choice of representation for tasking is pertinent to the C4ISR optimization problem. The representation must be capable of incorporating all salient aspects of the behaviors to be modeled in an efficient manner. Within JANUS, for example, tasks are carried out as a set of temporally linked moves (line segments) with a velocity vector specifying direction and velocity. The state of the entity (e.g., firing permission) is held as a separate construct which is not necessarily coupled to each move. NSI's extension to this concept augments these line segments with a set of parameters relevant to the action and time the action is being performed. This is very similar to (and derived from) the task frame constructs used within ModSAF (and OneSAF).

In ModSAF, each vehicle is controlled via a sequence of temporally linked 'task frames' which specify the path, velocity, firing permissions, targeting systems, and other pertinent parameters. A set of task frames acts as a time-ordered sequence of behavioral plans specifying how, where, when, and what each vehicle will do now and in the predicted future (Figure 3). This provides a mechanism for specifying desired (i.e., optimal) behaviors through a given period of time. ModSAF, however executes certain sub-behaviors within each taskframe using one or more fixed finite-state machines (FSMs). Evolution of the number of task frames, the specific types, and all parameters in these task frames for each vehicle can be performed as an independently scheduled task. By coupling the action type with a set of pertinent parameters, more complex behaviors can be controlled and optimized. From an Air Force C4ISR combat perspective, the addition of parameters within such a task frame may allow optimization at different levels of hierarchy (i.e., subjugating lower-level behaviors to heuristic control mechanisms). For example, low-level navigation around an obstacle may be performed using a set heuristic wherein the higher level tasking (e.g., travel to a specific point in space) is open to full optimization.

[2] ModSAF utilizes the system clock in some parts of the scheduling algorithm, hence exact replication of scenarios may not be possible. Thus, repeating each test a sufficient number of times is necessary in order to obtain accurate statistics with respect to overall system performance. Modification of this random number generation mechanism to use a deterministic pseudo-random number generator would rectify this problem but may cause unknown effects in multiple terminal, interactive simulations.

| Task Frame 1 | Task Frame 2 | Task Frame 3 | Task Frame 4 |
|---|---|---|---|
| Type: Move<br>Duration $\tau_1$<br>$Vx_1$, $Vy_1$, $Vz_1$<br>Fire Control Set 1<br>Aux. Parameters | Type: Move<br>Duration $\tau_2$<br>$Vx_2$, $Vy_2$, $Vz_2$<br>Fire Control Set 2<br>Aux. Parameters | Type: Attack<br>Duration $\tau_3$<br>$Vx_3$, $Vy_3$, $Vz_3$<br>Fire Control Set 3<br>Aux. Parameters | Type: Halt<br>Duration $\tau_4$<br>$Vx_4$,$Vy_4$, $Vz_4$<br>Fire Control Set 4<br>Aux. Parameters |

**Figure 3**. Example of a temporally linked task frame set for controlling ModSAF vehicles containing 4 task frames. Control proceeds (from left to right) through each successive task frame.

The evolutionary process must be able to accurately predict where and when each entity will be in the (limited) future. This can be accomplished through running the simulation to completion stage, or by modeling the simulation itself using abstractions of behavioral results. Using the simulation itself is often infeasible as it 1) consumes considerable CPU time, and 2) is often not capable of being re-set to a specified starting time. Hence, creating a model of the simulation itself is often the only realistic option. Planning and prediction must take into account available (or anticipated) terrain, intervisibility, and communication information. Problems arise, however, when the heuristic low-level planning mechanisms deviate significantly from the evolved high-level plans. This is a problem encountered interfacing the evolutionary planner with ModSAF, which generates low level behaviors using a set of hard-coded finite state machines. Discrepancies typically occur when entities navigate through difficult terrain or around barriers, leading to both timing and position errors. Although these low-level actions are controlled by hard-wired heuristics inside ModSAF, their resultant behaviors are not easily predictable. Access to the heuristic rule sets is limited at best, and the number of rules precludes realistically utilizing them in the path planning prediction process. Solving this problem involves creating and evaluating plans that are not only desired but are also actually realizable. Thus ModSAF (and presumably OneSAF) will require additional effort to incorporate features which minimize any such discrepancies.

JANUS and many other simulators model movement as simple point-to-point actions. This may be inappropriate for air-to-air combat, since it neglects the physical dynamics of turning to achieve the new heading and velocity. For air strike planning this may not be of major concern as the optimal plans are executed by the pilot who most likely will have direct control of vehicle turns. Incorporating turning dynamics into any simulation results in a more realistic model at the expense of considerably more computational time. Depending upon the end application and level of abstraction, these features can be added or omitted from the evolutionary optimization software without difficulty.

Of equal importance to the representation is the capability of the model to incorporate environments that accurately reflect real-world situations. Many of these features are common to both ground-based and air-based combat simulations. These include general weather conditions, time of day, terrain features, etc. Weather may require augmentation to specify conditions at various altitudes (a capability not currently implemented within JANUS or ModSAF/OneSAF). Non-homogeneous weather conditions could be implemented within the EP-based model by stratifying the altitude domain. Two or more

discrete layers would model the applicable altitudes, with discontinuities at boundaries. This method has been successfully implemented in ocean-based simulations wherein sound speed profiles rely on similar stratification modeling. This can be implemented as needed depending upon the degree of fidelity required.

Threats can be modeled using various entity types, both fixed and movable. Surface to Air (SAM) sites were added to the JANUS simulation by creating a new entity with specified (x,y,z) position, no movement capability, limited ammunition (missiles), and specified kill radii. Both hemispherical and cylindrical probability of kill models have been successfully included. Since these entity types are easily created and instantiated with different parameter sets, many different threat types can be included in the simulation.

Constraint modeling is also important to the optimization process. Two main constraints are the terminal points of a flight, and the inclusion of potential no-fly zones. In air-combat/strike planning it is often important to constrain the flights to return to the point of origination. Within the EP model NSI accomplished this by setting the terminal way-point to be equivalent to the starting point, and not permitting it any freedom of movement during the mutation processes. Thus, round-trip travel was proscribed for all aircraft in the simulation. Other possible implementations would allow terminal points to vary within a fixed set of points describing preferred landing sites. If this constraint is relaxed, additional mechanisms should be incorporated to ensure that the aircraft entities land in appropriate areas (and not just terminate their routes abruptly as land vehicles can).

Similarly, there are several methods of incorporating other (hard and soft) constraints such as no-fly-zones into the simulation. One, albeit unorthodox, method involves modification of existing terrain features such that they pose an impenetrable area within the flight envelopes of the designated aircraft. A better method and one which has been successfully implemented within the EP/JANUS software suite involves incorporating a binary mask of terrain features added to the existing terrain map. This mask is then used to define a penalty function which decreases the fitness score of entities that fly through it. The chord segment distances through the penalty zone are aggregated and used to either completely negate (all or nothing) the strike plan, or to decrease the score proportionally to this traversal distance. Typically the penalty function is constructed to allow some penetration for initial plans which are subsequently refined through the mutation process. After a sufficient number of iterations, final solutions exhibit plans that navigate around the no-fly zones. Implementation of specific mutation operators that augment this process can significantly speed the convergence process. Similarly, a heuristic mechanism can be used to 'correct' and refine those plans around the specified no-fly zones. Since the penalty function can be tailored with continuous or discrete functions, any number of soft/hard constraint models can be implemented as desired.

Another required feature is the ability of the simulation engine to access plans from the evolutionary optimization engine in a timely fashion. In order to accomplish this without degrading the simulation performance, the simulation engine should be capable of implementing a fast interface mechanism, such as shared memory. Shared memory segments are physical memory (RAM) allocated by the operating system to a process.

This RAM is then directly accessible by the originating and other designated processes. File based data transfer mechanisms are considerably slower, relegating them infeasible for this purpose. The shared memory can be used as a reflective memory wherein all state information normally stored in the simulation software is replicated within the shared memory. A simple memory copy is utilized, resulting in little or no computational impact on the simulation process. In addition to the shared memory segments, some mechanism must be in place to transfer data to and from the shared memory. The requirements for this mechanism is to be able to 1) attach to the shared memory segment, and 2) transmit data asynchronously bi-directionally to and from this shared memory. Additionally this interface must be sufficiently fast and robust as not to impact the performance of the host computer running the computer. Since shared memory is only accessible by processes executed on the same computer, the data transmission mechanism must also reside on this host computer. Ideally it should be a completely separate process for the sake of robustness. A simple client-server mechanism such as that shown in figure 2 accomplishes all of these goals. This interface process can act as a client to the simulation engine and as a server to all other processes (e.g., the EP optimizer).

Standard internet sockets using TCP/IP or UDP protocols are sufficient for the data transfer interface. Other mechanisms might use UNIX pipes or equivalent concepts on non-UNIX machines. Since internet sockets are fairly universal among all modern computers, this is a more extensible approach and should make the software applicable to simulation engines running on virtually any current or future operating system.

Provided the simulation engine interface is designed as described above, the computer platform/operating system hosting the evolutionary optimizer should be capable of supporting a graphical user interface to allow the user to enter and view parameter settings.

Finally, some method for grouping various simulated entities is required. Group actions are often required as part of military doctrine. Doctrinal constraints require that multiple agents on the same team must work in concert, achieving their mission goals collectively. A planning solution consists of a set of behavioral plans, one for each entity on the team. The scoring function must consider an aggregate of all individual behaviors for a specific team solution. This entails implementing hierarchical models (i.e., a platoon of tanks or a squadron of aircraft), versus representing and evolving solutions for each individual entity. If a chosen simulation itself currently does not incorporate grouping and cannot be readily adapted, the evolutionary program can still be used to create group behaviors and later copy these plans back into the appropriate individual entity parameters within the simulation engine. The net effect will be the same although the visualization within the simulation engine may show a set of identical plans versus a plan for the designated group.

## 6.0 Proposed Software Implementation

The design of the evolutionary algorithm system should result in a platform independent program that is capable of utilizing a variety of simulation engines. This will greatly enhance its applicability as doing so will make the program useful well into the future, and applicable to more than just one simulation engine. To achieve this goal, a design

that utilizes a high degree of object-orientation (for extensibility), multiple independent processes (for robustness), and standardized inter-process communication mechanisms is required. The design shown in Figure 3 meets these requirements.

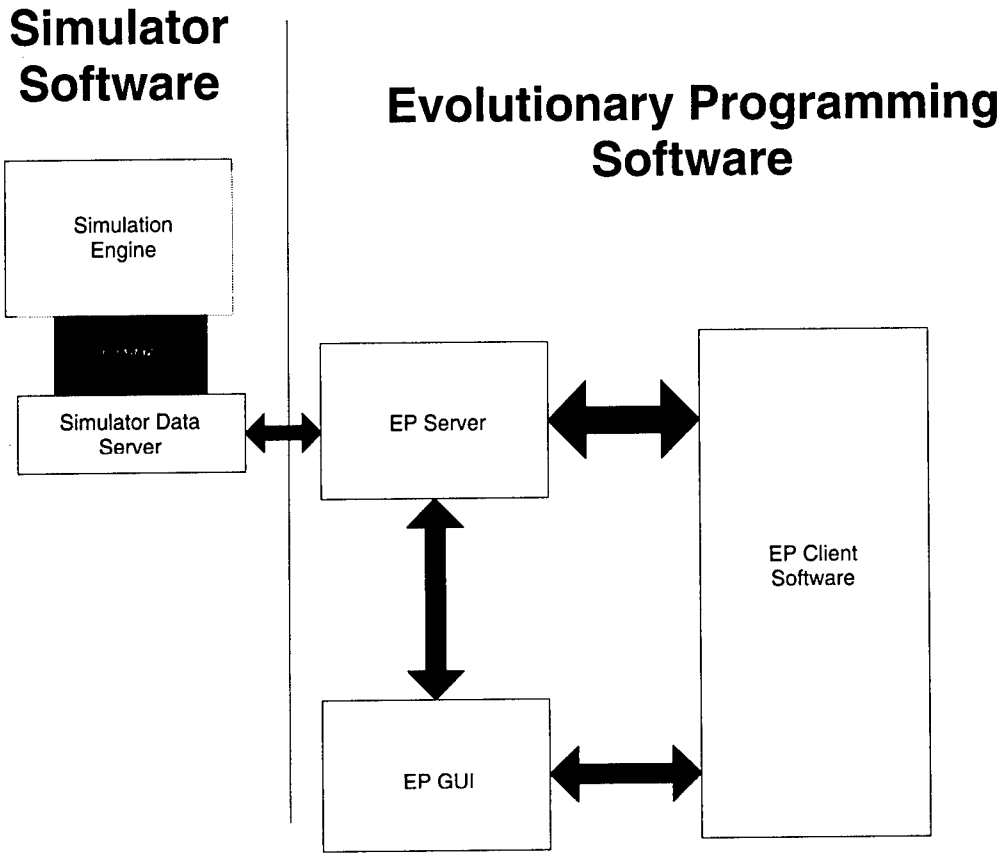


**Figure 3**. Processing diagram of EP / Generic simulator software system.

### 6.1 Robust Software Design

The proposed design approach consists of distributing the EP task(s) as separate processes (clients) that act and communicate asynchronously with the host simulation engine. Although only one EP client process is shown in figure 3, any number of such processes can be instantiated to optimize actions for simulated entities independently or cooperatively. By incorporating and implementing all communication with the host process through standard internet-type socket connections (TCP/IP), virtually any computer can host one or more EP processes independent of the host simulation engine. In this way, off-line computation can be accomplished via other computer assets (similar

to using auxiliary array processors) to offload computationally intensive tasks. One interface program acting as a server process interfaces with the host simulation engine. All EP processes communicate their requests (acquire data, transmit data) to the server, and not directly to the simulation engine, increasing robustness. Thus the EP processes (can) operate independently from any predefined simulation engine. This provides great flexibility for incorporating this same behavior optimization software into other existing and potential future simulation engines.

The server process is either called directly (as a subroutine) from the host simulation engine, or as a stand-alone process that will interface through shared memory (or other suitable interface mechanism) with the simulation engine. To interface with a new specific simulation engine, only the part of the server process that transmits data to and from the simulator needs to be tailored to the particular simulation (e.g., ModSAF). This transmission interface methodology is a standard, robust concept in other programs involving communication with a variety of heterogeneous computers/programs. Communication bandwidth is minimal since 1) transfer of pertinent simulation environment data (i.e., world topography) is only required once per EP process, 2) updates in the scenario data are relatively small in size, and 3) data transferred to the server after evolution are small (i.e., path plans for each entity being evolved). To maintain robustness, both EP client and server processes are designed to reconnect dynamically should socket communications be interrupted accidentally.

The chosen coding language should be selected with respect to speed due to requirements for speed in evaluating solution fitness. The additional computational cost of using JAVA or C++ may seriously impact the performance of the overall system, therefore it is recommended that standardized (platform/OS-independent) ANSI C be used wherever possible. Since it is largely interpretive, JAVA is up to an order of magnitude slower than other languages, and its garbage collection (object memory allocation/deallocation mechanisms) are known to be prone to failure. The C++ language, while providing built-in object-oriented benefits also is typically 15-20% slower than the same software written in standard ANSI C. Because speed of operation directly dictates how much of the search space can be analyzed, any reasonable method to minimize computational costs should be incorporated. No platform-specific constructs should be used (e.g., Microsoft extensions to the ANSI specification). Furthermore, the ease of interfacing with existing (i.e., ModSAF, OneSAF, JANUS, etc.) simulation code reinforces the decision to use ANSI C.

Regardless of the chosen base language, certain object-oriented design principles should be incorporated. Although it requires some additional coding effort, an object-oriented design does greatly enhance extensibility. For example, object definitions may include low level types (eg., XYZPosition, TaskSegment, Mission, MissionGoal, etc.) as well as high level objects (ScenarioData and EPSolution). Each object then incorporates basic set of constructors and operators that can be performed on the object. (NOTE: due to the generic object-oriented design, C code can be easily converted to C++, Objective-C or other object-oriented language if desired in the future).

Standard TCP/IP sockets provide the basis for all interprocess communication. By implementing these within a data-packet framework, a high level of robustness to failure can be incorporated. A data packet consists of a header, the data, and possibly a packet termination symbol. The header contains the packet type (for quick confirmation/rejection of incoming messages) and the number of bytes the data message actually contains. Additional security is obtained by comparing the number of bytes read to the number in the header. Messages which are corrupted or unexpected (unhandled) can be rejected without compromising system performance.

Multi-threaded code should be used to minimize potential bottlenecks in the system. Three separate threads can be incorporated as follows:

1) Receiving Thread – receives, parses, and transfers data inside (from a specified set of designated message types) to the Processing Thread. Mutually exclusive buffering is used to prevent overwriting data. This thread also rejects corrupt/unhandled message types.
2) Processing Thread – utilizes data received from the Receiving Thread, processes it through the evolutionary algorithm, and sets appropriate flags to ensure buffered data is not overwritten.
3) Sending Thread – transfers resulting data from the algorithmic actions inside the Processing thread, buffers these into data packets, and transmits the packetized messages to the appropriate client/server processes.

Since each thread acts independently (although the data is buffered through mutually exclusive flags) and is scheduled by the operating system, each can be set up to ensure communication bottlenecks are eliminated. A consequence of this design is the requirement that the operating system chosen for this implementation be capable of instantiating multiple threaded processes, and that it allows the software designer to set independent priorities for each thread.

### 6.2 Prediction Mechanisms

Prediction of opposing force actions is critical to the success of the optimization system. There are several possible methods of prediction of opposing entity behaviors. The most simplistic is to assume a single-player game wherein the motive behind actions of one team to not directly impact the behaviors the other opposing team. Although unrealistic, this approach allows fast computation where solutions are required in minimal time. Augmenting the state-space to incorporate anticipated (predicted) mission goals for opposing entities allows the optimization process to adapt intelligently to perceived actions. Actions taken in light of known information about the enemy are much more likely to succeed.

This is a much more realistic approach and can be accomplished as follows. First, the mission goals for opposing entities are initialized randomly, or by incorporating whatever information is known about the opposition forces. Within each iteration, the previous state of the opposing entities is compared against the current state (as determined by the 'snapshot of the world' mechanism described in figure 1. Estimates of mission goals for

opposing team members can then be updated using a predictor-corrector mechanism. In this way, state estimates are constantly updated and track the perceived motives of enemy forces temporally. Note that, however, if the estimates are erroneous, actions planned can be highly sub-optimal (i.e., estimate survival is the enemy's highest priority but in truth they are laying a trap and are highly aggressive). Thus this mechanism must incorporate some measure(s) of uncertainty.

In addition to estimating the mission goals for the enemy, prediction mechanisms are required to estimate physical position movements for some time into the future. First, second, and higher order models can be used depending upon what kinematic data is available. A key point in the success of position estimation into the future is to acknowledge the uncertainty of future predictions by de-weighting the 'predicted' actions as the prediction intervals increase. One method for accomplishing this is to use uncertainty covariance ellipses (or other distributions) which grow with time. A simple linear estimation model can be derived which updates these using an extended Kalman Filter (EKF) update cycle. Physical dynamics of the motion models can be incorporated to skew the distributions according to specific characteristics of opposing force entities. Prediction of own-team entities is unnecessary since the plans for these are created (thus known implicitly) or assumed known as a part of information sharing among common team members.

### 6.3 Mutation Operators

Regardless of representing the parameter state space as simple route segments or more complex 'taskframes', the set of mutation operators must be able to generate a range of variants of parent solutions. These mutation operators should be able to take small, medium, and large jumps in the solution space to prevent stagnation and entrapment on local optima. Assuming a simple route segment model containing a velocity vector and terminal stage operation (i.e., wait, turn, fire, etc.) potential mutation operators found useful include

1) Add a new route segment randomly within the set of existing segments. Generate the new velocity vector randomly based upon a Gaussian or other appropriate distribution around the adjacent segment directions. A maximum number of route segments can be incorporated to constrain the resulting solution.
2) Delete an existing route segment chosen at random from the set of segments in the parent solution. A minimum number of route segments can be incorporated to constrain the resulting solution.
3) Randomly modify the parameters within an existing route segment (i.e., velocity magnitude, velocity direction, end state, etc.). The range of modifications can be similarly restricted to ensure physical bounds are realized for each parameter of concern (i.e., tanks cannot fly).
4) Merge portions of two existing solutions to create an offspring solution.
5) Merge portions of an existing solution with a solution created randomly to create an altered offspring solution.
6) Utilize a combination of the above mutation operations to create variants which have an increasing range of deviations from the parent solutions.

Unless otherwise specified, random selection of parameters indicates selection with equal probability for all outcomes. The following operators are directly applicable to more complex 'taskframe' constructs. Mutation operators should created to take into account the temporally variable nature of each individual task frame.

1) Add a task frame segment randomly by reducing the time durations of the adjacent segments. Generate the new velocity vector randomly based upon a Gaussian distribution around the current direction.
2) Add a task frame segment randomly by reducing the time durations of the adjacent segments. Generate a new velocity vector to smooth paths between two adjacent vertices.
3) Randomly remove time from a task frame segment and add it to an adjacent task frame segment without modifying the velocity vectors.
4) Delete a randomly chosen task frame segment.
5) Modify the velocity vector of a randomly chosen task frame segment using a Gaussian distribution centered around the current direction.
6) Modify the parameters of a randomly chosen task frame segment (firing permission, targeting parameters, etc.).
7) Modify the type of a randomly chosen task frame segment (e.g., from 'move' to 'assault').
8) Merge solution 'taskframes' from two or more solutions to create a new offspring solution.
9) Merge solution 'taskframes' from an existing solution and a randomly created solution to create new offspring solutions.

Another possible option is to incorporate randomized heuristics into the solution set. These may be derived from known solutions used successfully in previous scenarios or derived randomly. For example, obstacle avoidance can be accomplished as a low-level task by incorporating a randomized heuristic. When an obstacle is encountered (i.e., a no-fly zone, SAM site) the heuristic randomly chooses a direction, and augments the current plan with one or more additional tasking/route segments to the left/right of the impediment. Additionally, a 'back-up' step can be implemented within this approach. These methods have been found quite useful in optimizing solutions for low-level maneuvering around obstacles in land-based scenarios.

A self-adaptation mechanism to optimize selection of mutation operators can also implemented. Self-adaptation mechanisms can speed the convergence of evolutionary search algorithms. A threshold for each mutation operator is initialized and subsequently updated as a function of its cumulative effectiveness in improving the fitness of a solution. A mutation operator's effectiveness in improving the fitness of a solution path is then measured by the ratio of the number of times it improves the fitness to the number of times it is applied.

Making the number of mutations per offspring a function of the relative fitness of the individual vehicle (its fitness score) increases the convergence speed of the evolutionary algorithm. Individual entities on a team with behavioral plans demonstrating better fitness are mutated fewer times on average than those with poorer fitness. The average number

of mutations for each vehicle's plan in an aggregated solution is calculated independently for each entity in the team solution.

### 6.4 Scoring Function

In order to judge the fitness of any solution a suitable scoring function must be formulated. This fitness function should be a metric that evaluates the solution as a whole. Functional components of concern are

1) Probability of survival
2) Probability of kill
3) Attainment of mission specific goals
4) Resource utilization

Probability of kill and survival are calculated by integrating probabilities sampled at a set of discrete points throughout the projected scenario plans. Projected positions (and hence potential threat areas) of opposing entities are used for these calculations. Mission goals are defined as attainment of certain time-space requirements (i.e., the aircraft must be at a specific location at a specific time). Other applicable (previously tested using ground-based scenarios) entity mission goals include successful area defense, area attack, and pursuit.

Resource utilization is required in the scoring function to permit optimal use of limited resources. These typically include fuel and weapons, but could be potentially augmented to incorporate other limited assets of value.
Penalty functions for avoiding constraint bounds (i.e., no-fly zones) can be modeled using a variety of functional types depending upon the requirement for hard or soft constraints. Hard constraint penalty functions can utilize a unit step function, whereas sigmoid or other continuous functions are more suitable for preferentially avoiding physical areas and for generating solutions with semi-permeable 'buffer' zones around impenetrable objects.

The payoff function is then constructed to evaluate the worth of each behavioral plan using individual scores each entity (with respect to the environment). These results are aggregated by team to create an overall fitness score. Solutions for multiple members on a team are first weighted with respect to their assigned priorities, then aggregated. Thus it is possible (though not desirable) to evolve a high-scoring solution which contains behavioral plan for one entity which is excellent and plans for other vehicles which are significantly worse. In general, however, the evolutionary process will drive solutions toward those with high fitness for all entities simultaneously. A scoring function based upon the Valuated State State Approach accomplishes this in a most general manner.

## 7.0 Prototype Simulation and Sample Experiments

Due to the time and budget constraints of this project, generation of a full simulation model for Air Force C4ISR combat entities was not possible. However, a small-scale demonstration package was constructed using modified components from existing

EP/JANUS software, and incorporating additional graphical display software for visualization of solutions.

A set of simplistic experiments was conducted to evaluate the modified capabilities of the modified EP/JANUS algorithm. In all of these experiments, the focus of the mission was 1) attack one or more designated, prioritized targets, 2) avoid threat zones. Hard constraints imposed on all test experiments consisted of mandatory return to the starting location. Soft constraints consisted of 1) minimization of flight duration, 2) minimization of fuel used (Note: this is separable from flight duration since altitude changes are mutable parameters), and 3) avoidance of no-fly zones defined in the environment.

Threats (SAM sites) were modeled as non-movable entities with fixed a amount of ammunition (missiles) capable of shooting down anything traveling within a defined circular envelope around the site. Both cylindrical and hemispheric threat models were defined with the probability of kill proportional to the chord length traversed by the aircraft through the kill radius. Since threat zones are parameterized objects, any number of these can be implemented within a scenario, each with potentially different ranges, kill probabilities, number of missiles, etc.

### 7.1 Prototype Simulation Scoring Function Components and Implementation

The scoring function for the prototype problem consists of the several components aggregated into a modified Valuated State Space (VSS). These components represent measurable factors of interest to the problem at hand, in this case, air strike planning. To measure these factors the simulation is typically not run through to the point where end results can be ascertained. This is due to three main factors 1) it is computationally expensive, 2) the simulation software often cannot be reset in time back to any desired starting point, and 3) accurate statistical analyses of results would require multiple runs due to the stochastic nature of the simulation. Hence measurements are modeled with approximation functions and these results are used to accumulate the resulting fitness score.

For this simulation experiment, five components pertinent to the air strike planning task. These parameters are 1) self damage, 2) damage inflicted on the enemy, 3) fuel used to complete the tasking, 4) end distance from a designated goal position, 5) cumulative survival as a measure of team strength, and 6) influence over the defined battle area. In this particular test example, since these parameters do not entail various sub-parameterizations the VSS can be reduced to a weighted sum of the individual components. Individual performance measures are defined below in more detail.

#### 7.1.1 Damage Assessment and Survival / Kill Probabilities

Self damage and damage done to the enemy are measured as the cumulative damage inflicted upon each member of the self (controlled) team. Damage is measured probabilistically along discrete time steps. The simulation time is discretized, and known and projected positions are estimated through to the end of the desired simulation time. Each entity in the simulation has a known set of munitions, each with a pre-defined probability-of-hit table (with respect to range, velocity, and other environmental

parameters). Probability tables can be modified as required based upon known characteristics of a weapon and delivery system. Typically these parameter tables indicate higher probabilities of hitting targets at closer ranges. Thus, plans which traverse closer to (or directly above) targets obtain a higher score. For this experiment NSI utilized the current physical parameters defined in JANUS v6.34 for USAF A8 jet aircraft and its associated munitions systems. The SAM sites (enemy forces) used the parameter set defined for Soviet SA-13 missiles. Maximum ranges for both of these weapon systems were, however, set independently (overriding the JANUS parameterization) to 5km and 20km, respectively. Intersections of these probabilities (with respect to range) are used cumulatively (as a product) to calculate instantaneous damage to and damage done to each entity in the simulation. The current JANUS model utilizes four quality states for an entity, undamaged, mobility damaged, firepower damaged, and complete destruction. The number of hits acquired by an entity and the position of the hit are used to determine the degree of damage. The instantaneous probability of survival (equivalently, the probability of being killed) for each entity are calculated using the 'complete destruction' damage calculations. These then are aggregated over the total discretized simulation time to determine the net outcome (with respect to survival/damage) of the planned behaviors.

With each sequential discretized time step, a cumulative probability of survival is calculated for each entity in the scenario. Cumulative probabilities at each time step $t_k$ are calculated from the instantaneous probabilities as follows:

$$Psurv(j) = \prod_{i=0}^{Ne} [1.0 - \alpha(i,j) * \hat{P}kill(i,j)]$$

$$Pkilled(j) = 1.0 - Psurv(j)$$

$$\overline{P}killed(j)_{k+1} = \overline{P}killed(j)_k + \overline{P}surv(j)_k * Pkilled(j)$$

$$\overline{P}surv(j)_{k+1} = 1.0 - \overline{P}killed(j)_{k+1}$$

$$\overline{P}surv(j)_0 = 1.0$$

where $N_e$ is the number of enemies,

$\hat{P}kill(i,j)$ is the instantaneous probability of the i-th entity killing the j-th entity

$\alpha(i,j)$ = 1 if entity i is shooting at entity j
= 0 otherwise

Psurv (j) is the instantaneous probability of the j-th entity surviving

Pkilled(j) is the instantaneous probability of the j-th entity being killed

$\overline{P}surv(j)_k$ is the cumulative probability of survival for the j-th entity at time $t_k$

$\overline{P}killed(j)_k$ is the cumulative probability of the j-th entity being killed at time $t_k$

This results in a cumulative probability for each team member entity at the end of the predicted time interval. A normalized sum is generated from these terms, which can be weighted by the specified priorities for each 'home team' entity or opposing team entity as specified for the team solution. By accumulating these weighted terms, a team probability of survival (and probability of killing the opposing team forces) is available for use in the VSS scoring function. Since cumulative survival for a team is an aggregate over all entities on the same team the score is normalized by the number of entities on that team.

### 7.1.2 Resource Utilization

Measurements of fuel usage are calculated based upon the fuel usage per unit time parameters within the JANUS data files. Note, however, that these data tables do not currently recognize differences in fuel usage due to elevation changes (acceleration up to a specified altitude) and turning maneuvers. This value is then normalized by comparing it to the maximum values stated in the data tables for the specific entities.

Distance to the end goal position is a measure of how close the final plan comes to achieving a physical goal. The difference between the ending position and targeted goal is normalized based upon a pre-set range parameter (defined specifically for each potential ending goal position). Ultimately this may become a set of goals, within a temporally prioritized sequence, but in the current version of the software only the ending position is used (and only if a goal-position specific mission objective is selected). In the prototype tests since the mission was not dependent upon a end-position goal, this component was ignored.

Certainly utilization of other limited resources can be included into the simulation scoring function. These quantities include water, food, human physical capacities, and perhaps potentially restricted operations such as electromagnetic transmissions.

### 7.1.3 Strategic Influence

An influence parameter is also calculated for the team entities. The influence value for a team is calculated by determining the areas over which the entities have potential for inflicting damage, regardless of enemy positions and capabilities of inflicting damage upon the enemy. Influence values are modeled using a derivative of a heat-equation boundary value problem in classical thermodynamics. In a nutshell, influence is (non-linearly) proportional to the aggregation of force at the discretely sampled points in time. The distribution of entities in (x,y,z) position space is used in these calculations.

This technique is based roughly upon a map weighting technique (Zobrist 1969). The algorithm starts by allocating a grid array (with the same dimensions as the map) and

initializing this array to zero at all locations. The granularity of the grid array is arbitrary and is typically constructed from convex polygonal (e.g., quad, hex, octogon, etc. ) cells, but can also be created from any other discretized geometric base shape. Finer grid meshes result in higher accuracy at the expense of significantly higher computational times. A positive value is placed at each friendly entity grid location and a negative value at each enemy entity location. Neutral entity locations can be set to positive, zero, or negative values based upon their modeled 'expected/anticipated degree of cooperation'. The algorithm then iterates, adjusting the values at each location in the array by the values at its neighbor positions, until it reaches a steady state (or a specified termination criterion is reached). One rough measure is to increase the value at each position it by one for each friendly entity neighbor and decrease it by one for each enemy entity. At the steady state, the resultant map values indicate the degree of control the teams have over the mapped combat environment.

The sign of the value indicates which side has some degree of control. Values near zero mean that no team controls the territory (i.e., no-mans-land or a 'front' zone). Large values indicate there is a strong control in that area by one of the teams. To determine the starting value for each entity type, each point is set to the average of its four previous neighbors, then scaled with a thermal conductivity (based upon terrain characteristics). Points which have constraints on them must remain untouched and boundary areas at map edges should have non-zero constraint values.

The matrix convolved upon the topographical map can be partitioned into quadrants, hexagons, or any other suitable discretization based upon computational requirements.

A value can be placed upon on each type of terrain, and each type of objective, depending on how useful it is to each of the strategies applicable to the designated mission. In tank warfare, for example, a forest might be set to a high defensive value, a low offensive value, and a moderate movement value. By creating new maps that have a value within each discretized grid cell it is relatively easy to evaluate strategic influence. If these values are then combined with entities currently on the topographic map, it is possible to evaluate how well or poorly defended an area is, where weak and strong points are, etc.

A tank, for instance, may add a significant offensive value to it's own grid cell, and due to the range of it's weapons, may add a slightly lower offensive value to all grid cells within it's level of force (LOF) and range (possibly modified by terrain and probability of hitting as modified by range). If another tank is beside it then the two LOFs and ranges would very likely cross, and create fields of fire with overall greater values. An enemy tank within this field will correspondingly reduce the values by it's defensive values. This is analogous to building up a histogram of the battlefield to determine positions of relative strengths, weaknesses and balance points.

Simplified heat transfer equations have been implemented for the NSI influence model, with each entity being represented by a temperature constraint on a point of a flat plate. If a single point is constrained to a certain temperature, it will influence other points of the plate over time, eventually reaching a steady state. With only one point constrained,

the entire plate will reach a constant temperature in the steady state. With several point sources, both positive and negative, the steady state solution will define areas of influence, and define a picture of controlled areas. The following is a simple example of how a few entities might affect their surroundings:

| Initial: | Steady State: |
|---|---|
| 0 0 6 0 0 | 0 3 6 3 1 |
| 0 0 0 0 0 | -3 -1 2 4 3 |
| 0 -6 0 4 0 | -5 -6 -1 4 4 |
| 0 -4 0 0 0 | -5 -4 -2 5 5 |
| 0 0 -6 6 0 | -4 -5 -6 6 5 |

A contour map produced using this data represents a front at a zero level. Using the heat-transfer analogy, by changing the thermal conductivity of any point on the plate, it is possible to modify the ability to influence it. For example, imagine a layered, composite plate constructed of various (not-necessarily contiguous) insulating properties, overlaid together to represent terrain on a map.

If a particular map section composite has higher overall higher heat conduction properties, it will transmit more heat than a section made of with overall higher insulation coefficient. The concept is to define how well each terrain type conducts influence. Relative to ground-based combat, mountainous terrain might be made act as an insulator, forests might have very high heat conductivity, plains and roads set to other intermediate conductivity values. Thus, a tank positioned in the mountains will not be able to command as large an area as one in the middle of a field.

For example, imagine a tank sitting in the middle of a 5x5 grid, with the all edges constrained to a level of zero. The map on the left contains the terrain data in terms of thermal conductivity (a function determining how well heat (military strength) can be transferred through it). Let the integer values '4' represent forest, and the integer values '9' represent open areas. The influence of a strength '8' entity on a specified terrain map would be:

| Terrain Map | Influence Map |
|---|---|
| 9 9 9 9 9 | 0 1 2 1 0 |
| 9 9 9 9 9 | 1 3 4 3 1 |
| 9 9 9 9 9 | 2 4 8 4 2 |
| 4 4 4 4 4 | 0 1 2 1 0 |
| 4 4 4 4 4 | 0 0 0 0 0 |

Thus the influence map shows the area the entity commands as well as the relative degree of control. This approach works well with supply lines by setting the conductivity for roads to be relatively high. Using this technique, ALL entities along or near a road will increase the power of any other entity along that road. If an enemy puts entities on a road square, the line to the remaining forward entities is obviously broken, and the influence from rear entities must now flow around the road (over land or other terrain).

On a topological map, areas of influence are analogous to hills & valleys. The evolutionary optimization algorithm then develops plans that take advantage of these level zones and hills and valleys. If for example in the optimization process, the mission is to destroy the enemy forces, plans which 'fill' up the 'holes' and 'valleys' (created by the presence of the enemy in the environment) in the landscape will score high.

Alternatively, if the mission objective is to capture strategic points with the least resistance, plans which find 'passes' in the enemy influence 'mountains' will score high.

### 7.1.4 Constraints

No-fly zones are also modeled, using rectangular sections superimposed upon the terrain grid. These zones are implemented as three-dimensional objects that extend past the flight limits of all aircraft (thus the algorithm could not just develop plans which flew over the no-fly zones). A continuous function that uses a sigmoidal shape defines the penalty for intrusion through the zone. The sigmoid function is defined as

$$f(x) = \frac{1.0}{1.0 + e^{-kx}}$$

where k is a parameter controlling the linearity (slope) around the zero crossing. The overall penalty is thus directly proportional to the time/distance traversed through the zone, permitting multiple transits with aggregate penalties. The form of the sigmoid function is convenient as it is asymptotic to (+/-) 1.0 and can be easily scaled and translated to the range [-0.5,+0.5].

Though the simulation software allows multiple entities in any combination to be used, due to available time and funds, only one aircraft was used for these prototype tests. This also made analysis easier. Positive scores were assessed for plans that traversed over the targets (the assumption was made that for these experiments, if the aircraft flew within a predefined distance (500 meters) of a target it could strike it with $P_{kill}$ proportional to the distance from the target. Aircraft were also modeled with weapons that could dispense up to 10 munition loads, each targeted to an individual (though not necessarily unique) target. Total kill scores were aggregated as the product of $P_{kill}$ multiplied by the designated target priorities.

Finally, based upon the maxim of parsimony, a small factor is incorporated that rewarded simpler solutions (with fewer task segments) over those with more complicated paths. This factor is calculated as

$$f(x) = 1.0 - \frac{N}{M}$$

where $N$ = the number of task segments in a plan
$M$ = the maximum allowable number of task segments in any plan

## 7.2 Test Experiments

Figure 4 displays the basic scenario at initialization stage with two targets (indicated by the symbol 'T'), three SAM sites (indicated by the symbol 'S') overlaying the basic topography (indicated by the various shades of grey). Circles around the SAM sites indicate the range the missiles launched from a SAM site can attack aircraft. The symbol 'A' indicates the starting airbase for the single aircraft flying the mission. The no-fly zone is not part of this example. In green are the altitudes of the target sites, whereas the purple digits (center left near the target) indicate the altitude of the aircraft's planned path. The best initial solution is indicated by the red segmented path. This path represents the best path generated after one iteration of the evolutionary program. Note that although it satisfies the constraints (return to the starting base) it only visits one target site, and does not visually appear to be optimal.

At the top left of this plot, there is a set of performance figures for the current iteration. Shown in yellow are the individual scoring components for self damage (inflicted by the SAM sites), enemy damage (based upon nearness to and weapons damage to the target(s), fuel usage (minimized parameter), distance to the goal point(s) (i.e., target sites), cumulative survival (based upon integration of probabilities of survival over the entire planned path. Additionally the influence parameter is calculated by determining the areas over which the aircraft has potential for inflicting damage. Finally the overall fitness score is displayed as the cumulative sum of the weighted individual components just described. The 'nomove' term is unused in these scenarios as it is applicable to ground-based entities only.

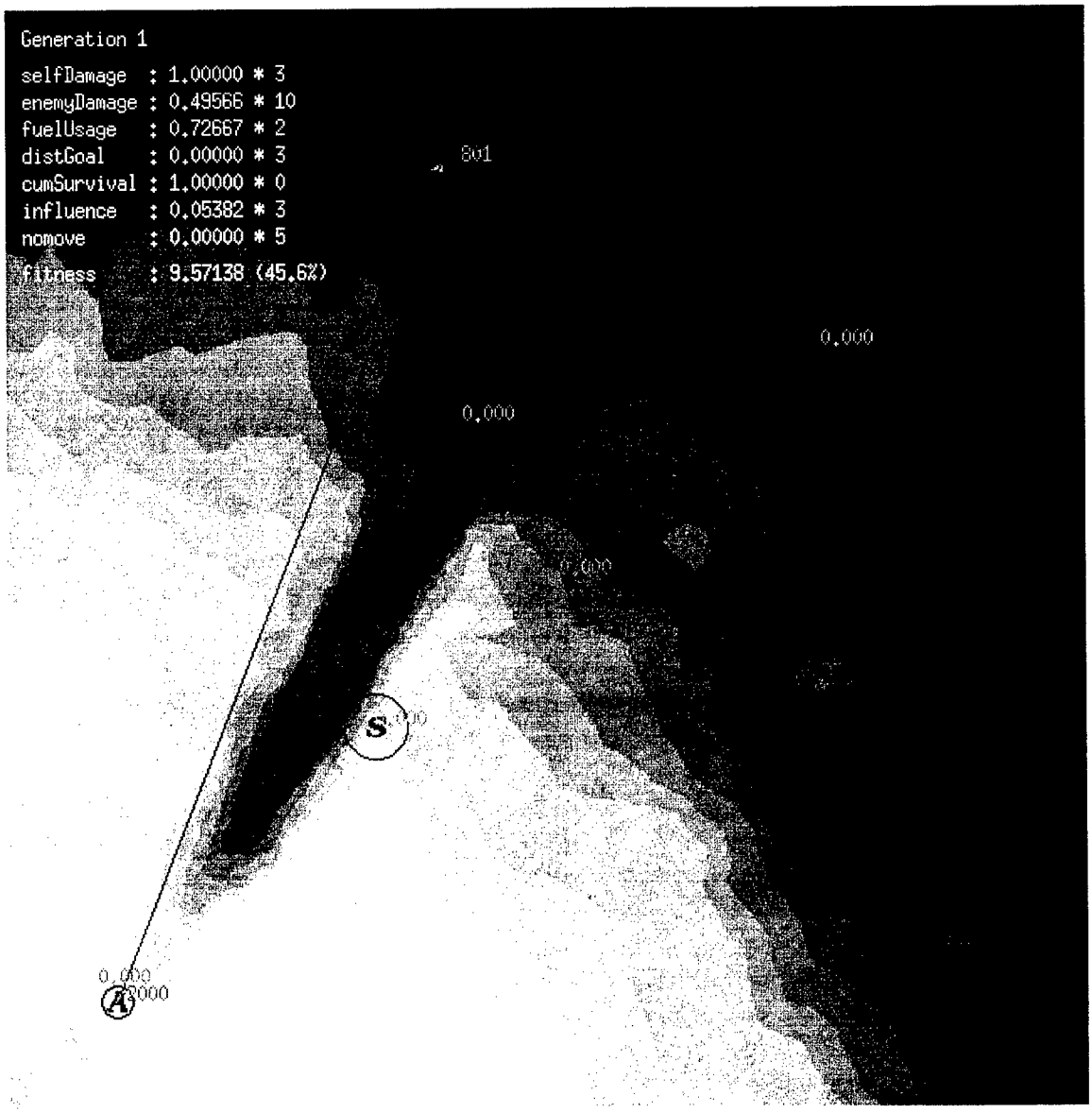


**Figure 4**. Initial plan for test scenario after one generation of evolution.

In figure 5 is the same scenario shown after 17 generations of evolution. The best path is again shown in red. In light-grey are the solution paths from the entire population. Again, purple numbers adjacent to route terminal points are indications of the altitudes for the individual aircraft path plan segments. Note that by generation 17 the software has evolve a solution which evades all SAM sites (no intersections within their circular kill radii), flies over one target and is quite close to flying directly over the second target. The cumulative fitness score indicates this is a better solution than the initial score shown in figure 4 above.

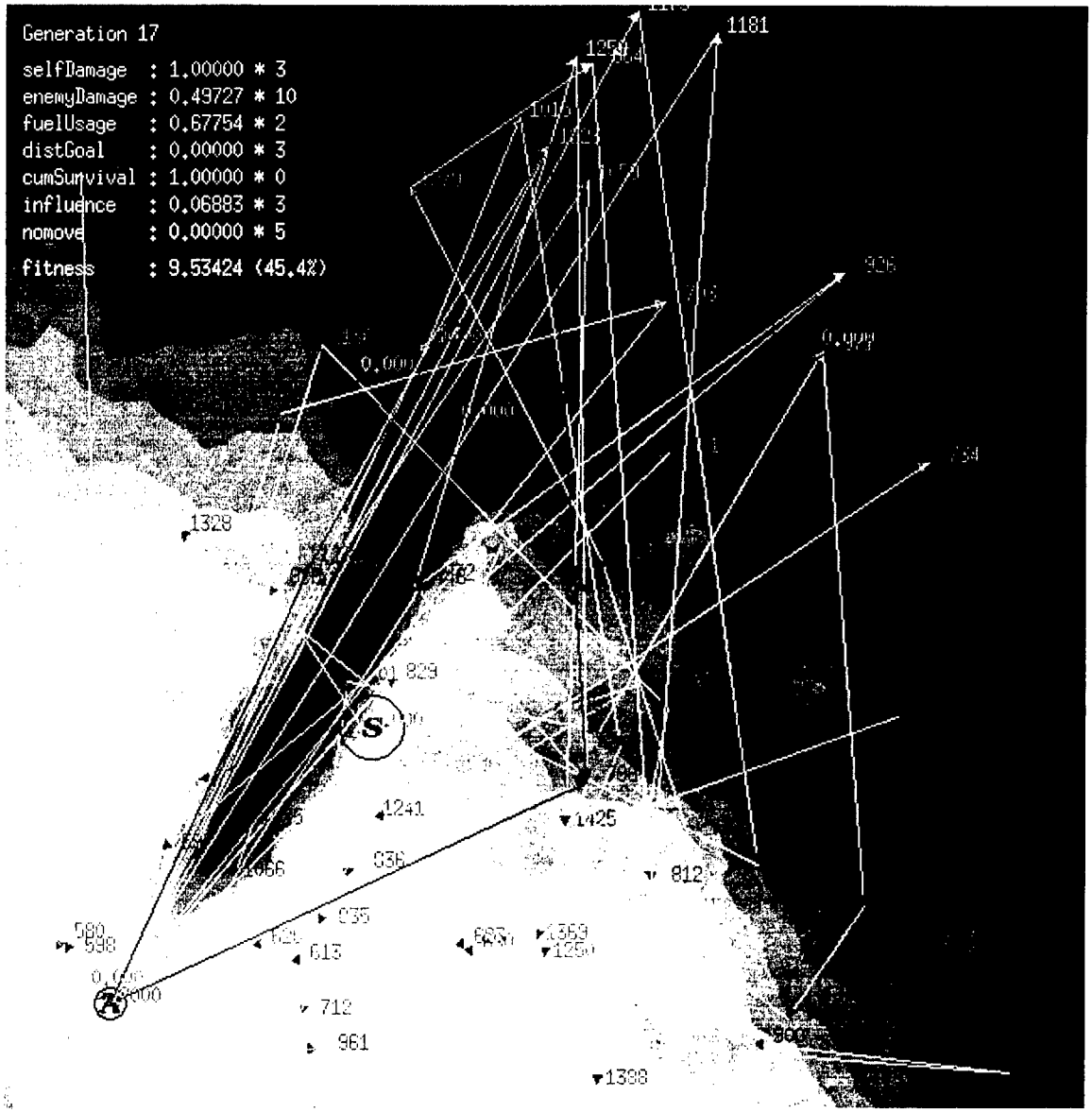


**Figure 5**. Evolved plans for basic scenario after 17 generations.

Figure 6 displays the final best solution after 44 generations of evolution. Note that the path now flies directly over both targets, and does not intersect any of the SAM site circular kill zones. Also shown, in light-grey, is the second-best solution from the population. Notice the small deviation (to the right middle of the plot) from the best plan (in red). The majority of these two plans are coincident, indicating the close similarity between the two solutions in fitness.

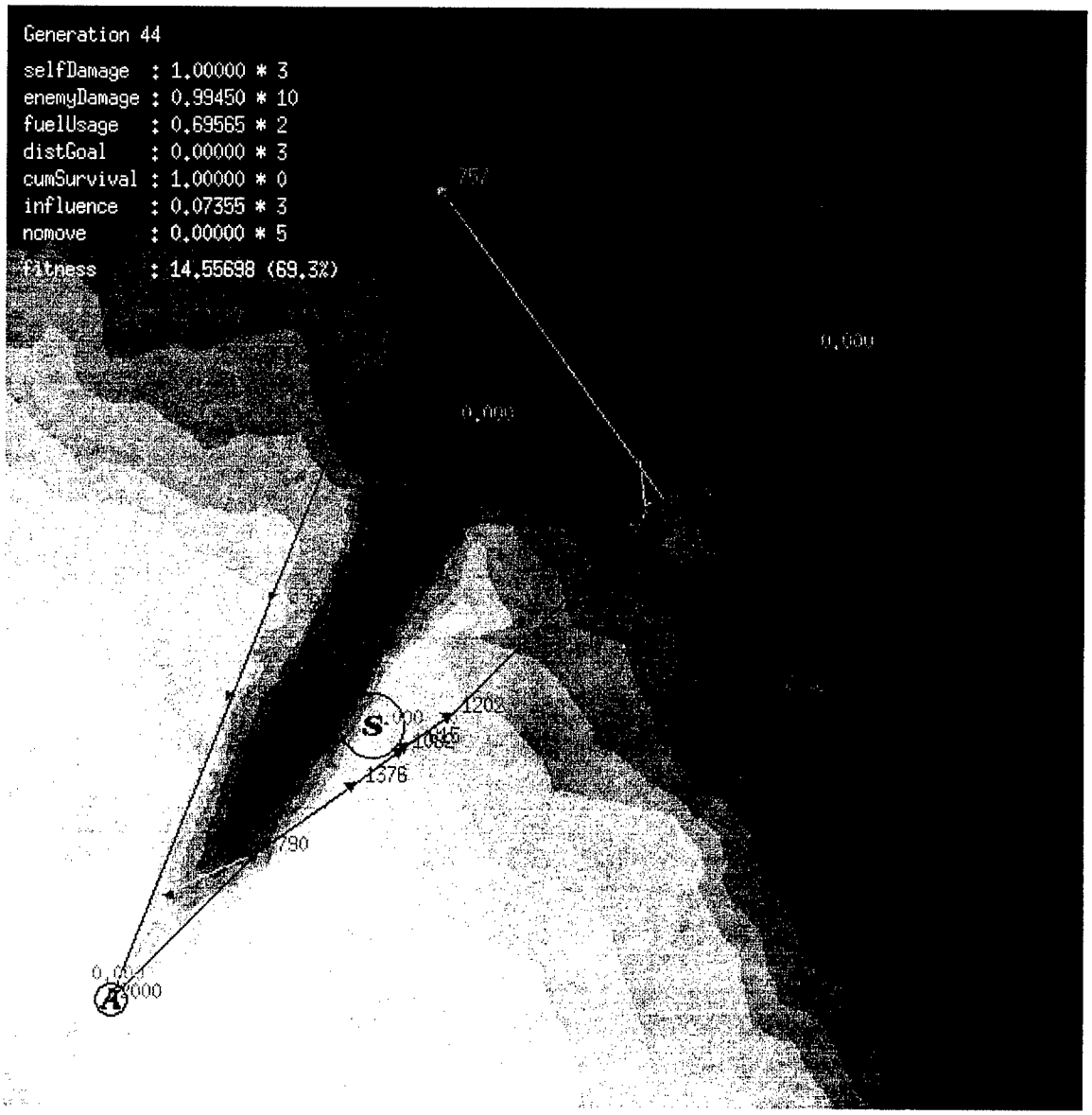


**Figure 6**. Evolved plans for basic scenario after 44 generations.

The fitness score for this best solution is noticeably higher than the best solutions shown in the previous plots. If the software is left to iterate over a few more generations, the expected improvements include tightening up the path segments such that they minimize threat exposure and fuel usage. Some evidence of this is already evidenced in the difference between the best plan in this plot and the best plan from the plot in figure 5.

Results of another set of test experiments are shown in figures 7, 8, and 9 below. In these experiments, the initial conditions were kept identical to the example shown in figures 4, 5, and 6, above, but an additional constraint was added to the scenario. To make the experiment more realistic, a rectangular no-fly zone (shown in red at the lower left side of the scenario topography) was added to the problem. Aircraft paths that intersect this no-fly zone are highly penalized to guiding the evolutionary process to derive solutions that avoid this area. Both the addition of and position (close to the origination point) of the

no-fly zone constraint makes this a considerably more difficult problem than the unconstrained example. For clarity, in figure 7, no initial solution paths are shown.

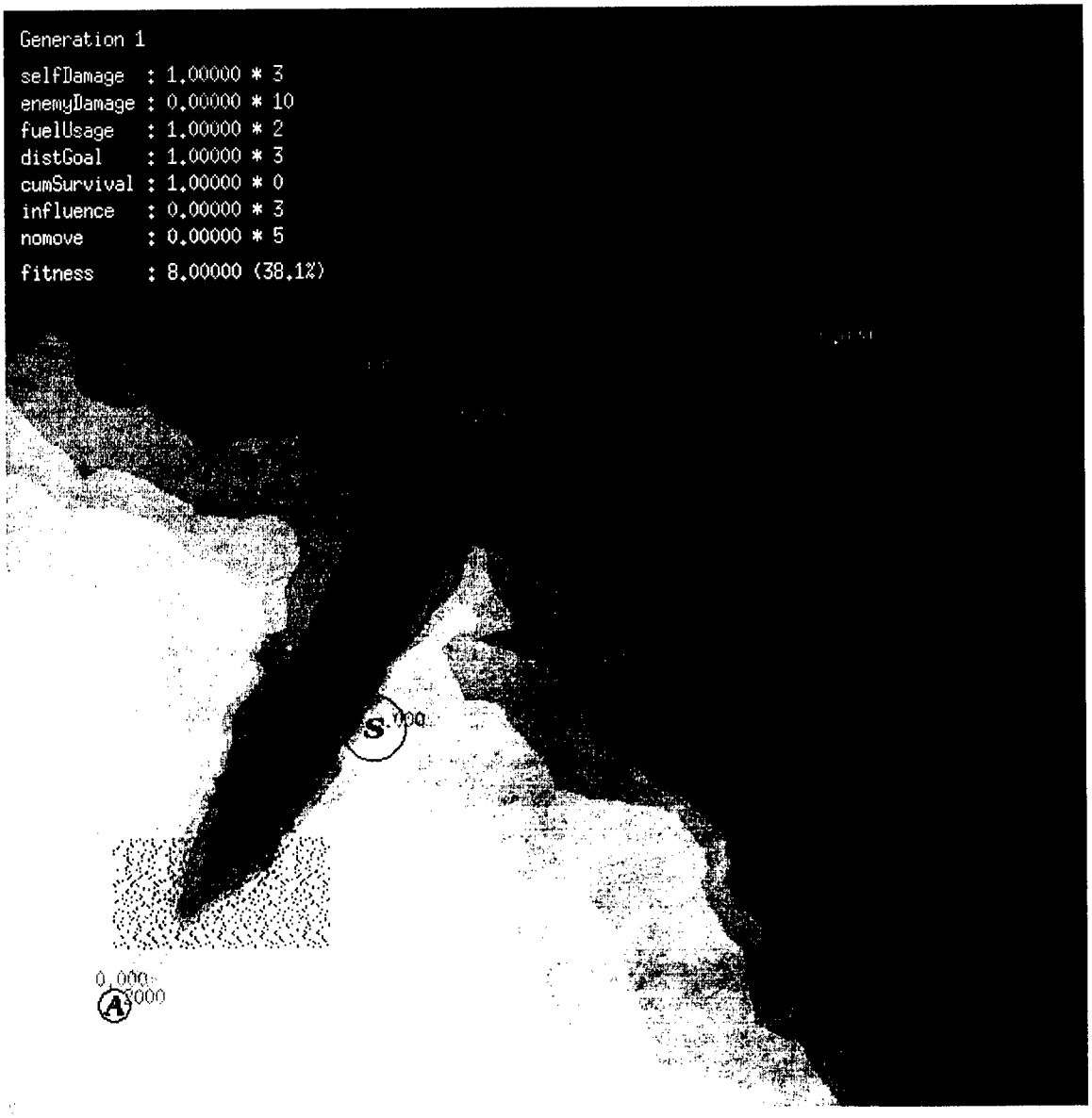


**Figure 7**. Initial scenario with one rectangular no-fly zone added.

In figure 8, the plot shows solutions generated after 9 iterations of the evolutionary process. Notice that the best solution, while hitting one of the targets, does fly over the no-fly zone, for which it is penalized. The component score for hitting a target, however, out-weighs the penalty for this solution. Variations in the no-fly zone penalty function can be made to adjust the degree of desired constraint (hard/soft).

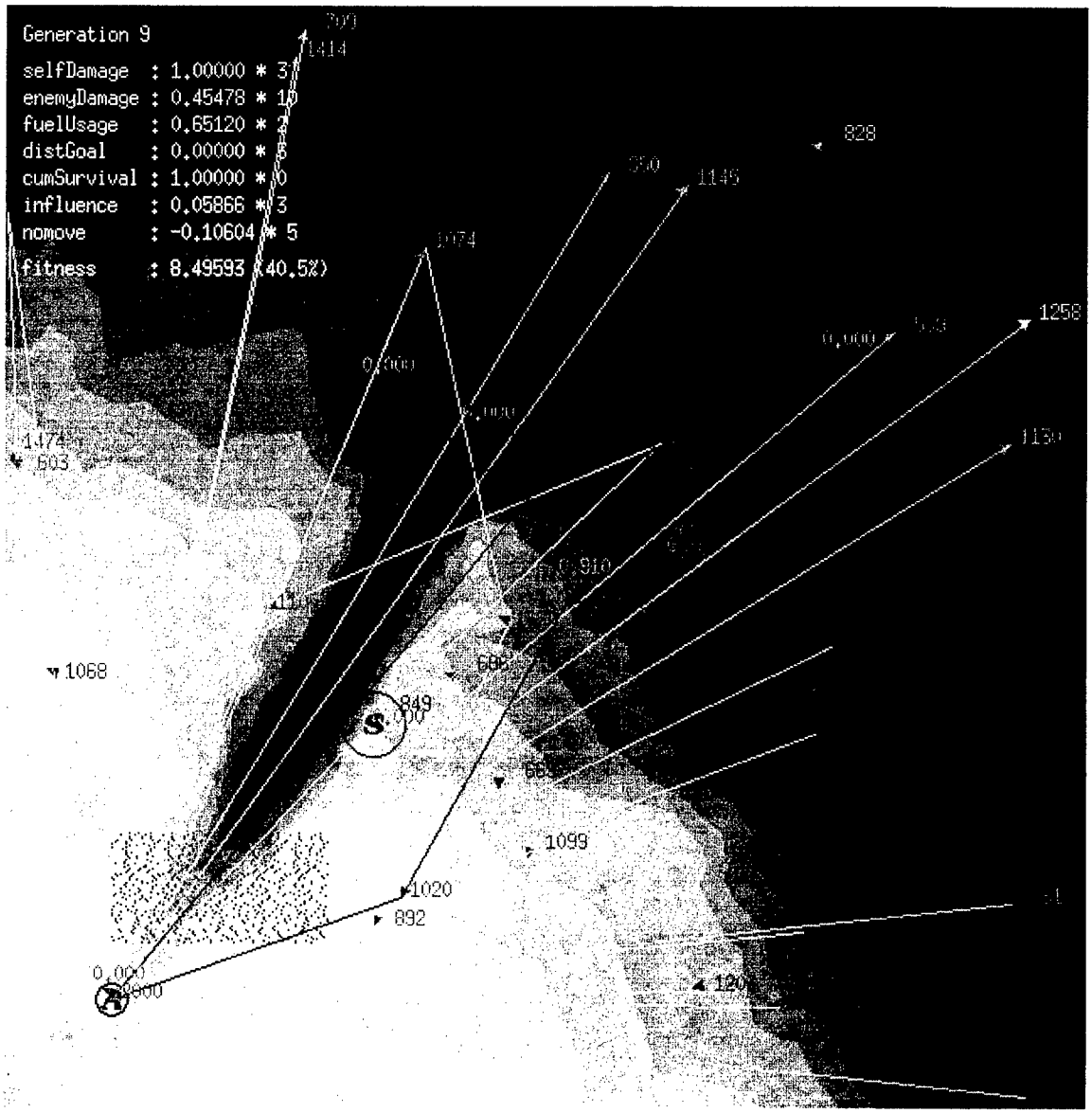


**Figure 8**. Modified scenario with one rectangular no-fly zone, after 9 evolutionary generations.

Figure 9 displays the results of the iterative process after 206 generations of evolution. By generation 206 the best evolved plan has successfully navigated around the no-fly zone, flies directly over both target sites, and avoids the SAM site threat zones. The second-best solution is also displayed (in light-grey). Again, the majority of this solution is coincident with the best solution, with minor differences noticeable in the upper left quadrant of the plot. In further iterations, the evolutionary program will continue to refine the solution (i.e., minimizing the overshoot in the upper left quadrant, thus minimizing fuel used and potential threat exposure). Note that this solution is not unique, as there are an infinite number of variations that could be evolved. Experiments with different starting random number seeds evolve multiple solutions with quite unique path plans.

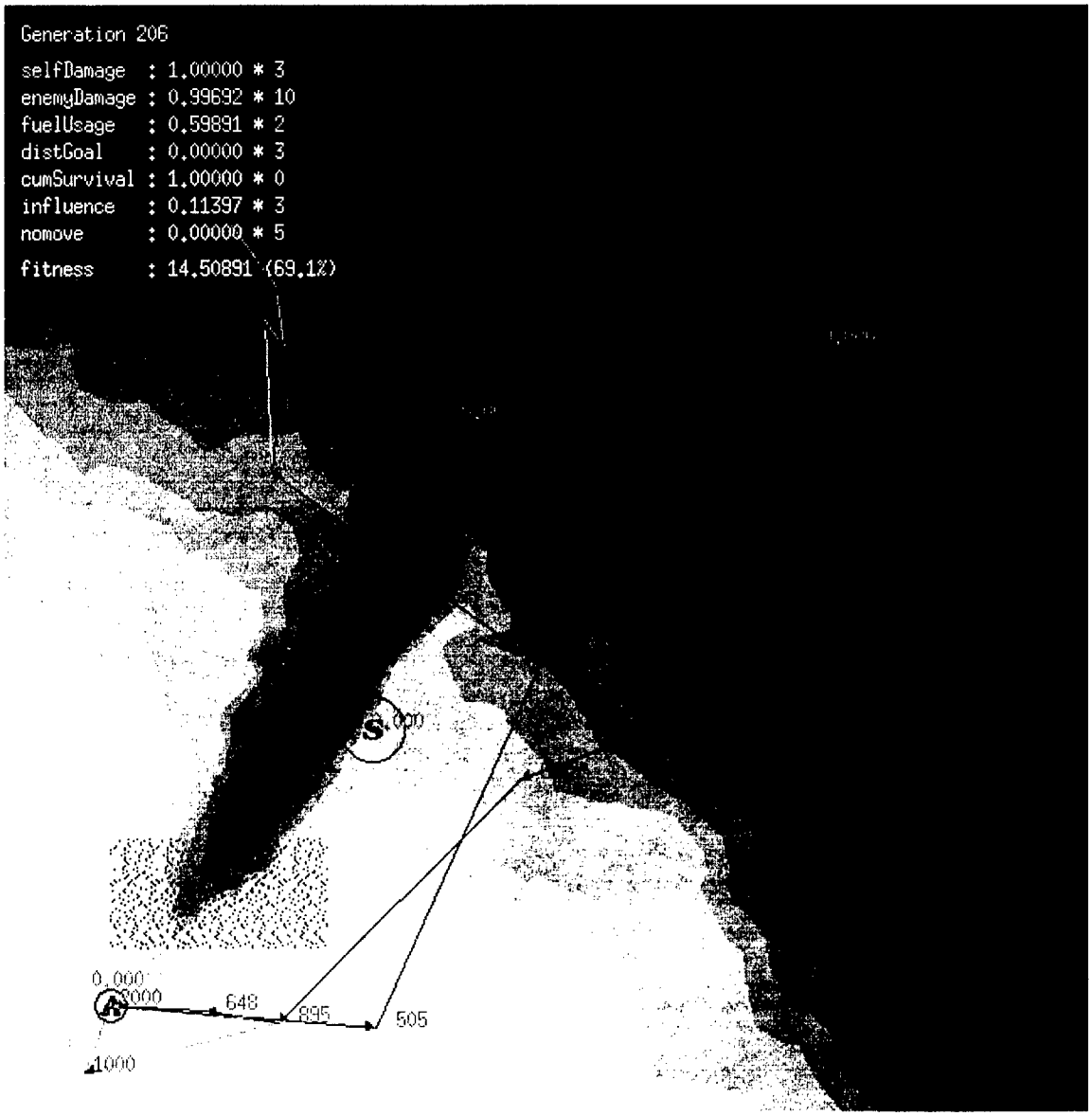


**Figure 9**. Modified scenario with one rectangular no-fly zone, after 206 evolutionary generations.

## 8.0 Conclusions and Recommendations

The combination of evolutionary programming with an appropriate objective function (such as the Valuated State Space Approach) has been proven in several instantiations to generate intelligently interactive behavior for autonomous entities. This combination has demonstrated the ability to learn/adapt plans with respect to defined mission goals in a highly dynamic environment. An assigned mission can be used to measure the worth of the current and prospective situations so that the evolutionary program discovers increasingly appropriate plans as it searches through the solution space. When a sufficient score is reached (or the available computation has been expended), the best evolved behavior is implemented. This capability has now been demonstrated for autonomous vehicles operating individually or in combinations, as well as in the presence of a similarly driven adversary. The mission of each side is arbitrary and may be changed dynamically. The constraints that each side reacts to are dependent upon the developing situation.

Optimal or near-optimal plans can be evolved in light of fixed or dynamic barrier regions (i.e., no-fly zones). Incorporation of terrain/airspace traversibility greatly increases prediction accuracy. The variable time nature of this approach makes combining high-level planning and low-level navigation possible. Incorporating probabilistic kill and survival functions further extends this capability, allowing for realistic simulations of two or more opposing teams on a battlefield.

By varying the population dynamics, number of generations, and timing between updates, scenarios with simulated forces of different intelligence can be effectively created. Further extensions of this approach will increase the usefulness of this software as a tactical aid and/or training tool. These extensions may include enhancement of predicted traversibility, extension of the task frame constructs, in addition to further refinement of the scoring function.

An important and informative lesson learned is due to the stochastic nature of both the simulation engine as well as the evolutionary program. Since the solutions and simulated vehicles utilize randomness, the same scenario may produce multiple yet equally valid solutions depending upon the random seed. Evidence of this is shown in re-running tests with different random number seeds where multiple outcomes are possible from the same scenario. For example, re-running a sample path planning test (eg., a barrier positioned symmetrically between a vehicle and its goal point) with different random number seeds produced different solutions (clockwise and counterclockwise) *each of which demonstrated optimal planning*. This is important as it indicates there may be more than one optima point in the solution space at a specific place in a dynamic environment.

Finally, it is important to note that this algorithmic approach and software can be used for non-military as well as military purposes. For example, in the civilian domain, routing and planning for fighting fires is of critical importance. This optimization approach can be modified to plan optimal paths to fight forest fires, and re-optimize based upon the often rapidly changing scope of the fire. Other application areas include traffic routing, automated navigation in ocean environments, and in the vast commercial computer game sector.

## 8.1 EP/Simulator Problem-Specific Recommendations

In the course of developing similar evolutionary software systems we have learned many important lessons regarding simulation of behavior and military doctrine. Continued development and extension of the software and algorithms will provide an effective tool for the military simulation community. While there are many possible avenues to turn the software into a fully realistic Air Force C4ISR tactical aid/training tool, we have detailed a course of action that will accomplish a set of realizable goals for the next phase of development.

Specific recommendations below are based upon experience and knowledge learned within this pilot project.

Due to the broad areas covered under the umbrella C4ISR, prior to any software modifications, one or more specific, well-defined Air Force C4ISR problem areas should be identified for implementation. These may include development of an air-strike planning tool similar to the prototype described above, training software for air-to-air combat maneuvers, high-level combat outcome tools which utilize interactive simulation capabilities, or any host of other applications.

Next, assuming JANUS, ModSAF, OneSAF, or other reviewed software engine is formally selected, the scope of the software revisions should be broken into discrete, mutually independent tasks. This will allow testing individual components such as communication improvements without impacting all other development efforts. All software should be implemented only after a thorough design review process wherein interfaces, algorithms, and performance requirements are formally specified. Although afterthoughts will always be part of the software process, this will minimize re-coding efforts.

## 8.2 Software Extensions

The current software system implemented together with JANUS makes an ideal starting point for anticipated Air Force C4ISR combat simulation tasking. In addition to the methods and extensions mentioned in section 6, other reasonable extensions to the software should include

1) adding to the current set of aircraft models and potentially, future aircraft models to test performance characteristics through simulation.
2) modifications to utilize full multi-threaded capabilities.
3) altering the fitness function to incorporate all known parameters of concern.
4) increasing the robustness of the communication mechanisms to further decrease failure and minimize effects on other simulation operators.
5) adding additional threat types and refining parameters for each based upon known characteristics.
6) modifying the point-to-point model to incorporate required flight dynamics.

As with any problem, utilizing available information specific to the problem domain can greatly increase the performance (if judiciously applied) at the expense of generalization. In this case, the low-level navigation components could benefit from such information. Augmenting the mutation operators, implementing path smoothing algorithms, and using other navigation specific information could produce more optimal paths. Again, an object-oriented approach would make this both economically realizable and easily extensible.

A full set of applicable entity types should be implemented. The current build of the software utilizes only a very limited set of entity types (e.g., tanks), therefore is seriously limited in the scope of scenarios which can be accurately simulated. A number of additional entity types, including other ground vehicles, fixed wing aircraft, rotary wing aircraft, and perhaps even dismounted infantry should be implemented. These modifications should allow for any arbitrary combination of these entities and entity types in order to provide for a useful planning/training tool.

Future software development should also incorporate the capability for autonomous control of singleton entities, platoons, companies, battalions and any combinations thereof. The software should be upgraded to evolve behavioral plans of action at each desired level in the hierarchy, from low-level up. Part of this upgrade involves incorporating the inherent ability to prevent or resolve conflicts as the chain of command dictates. For example, evolved behavioral commands filtered down to the platoon level must be consistent with and not conflict with commands evolved for entities at the battalion level. Accomplishing this in an efficient manner requires an object-oriented design. This approach would make incremental improvements more timely and economically feasible. Implementing the autonomous controller in object-oriented software would make extensions to incorporate different hierarchical levels relatively seamless. This is perhaps where the largest long-term benefits could be realized.

### 8.3 Hardware Requirements

While the current hardware is capable of running a range of scenarios, it is limited by memory and computational speed. Currently, our testing evolving solutions for up to 24 ground entities simultaneously indicates sufficient computational capacity exists using 600-800Mhz CPU speeds. Additional computational speed would be required for faster updates (currently ground-based entities update solutions every 30 seconds), or for evolving simultaneous solutions for a significantly larger number of entities.

Additional RAM memory and an additional computational power are also necessary if larger scenarios are envisioned. Requiring allocation of 256Mb RAM will reduce page-faults and significantly increase processing speed.

### 8.4 Testing

When sufficient software capability exists to simulate and evolve actions for any number of entities at the desired hierarchical level(s), more extensive testing should be performed. Evolved solutions should be compared against known strategies and results noted appropriately. Attention to existing military doctrine will be noted and if optimal

behavioral plans result in deviations from this doctrine, the results will be thoroughly documented.

It is certainly possible and probable that this software will devise behavioral plans of action which exceed the fitness of existing plans derived through military heuristics or other sources. This is especially true in dynamic environments and in simulations with multiple opposing teams. Therefore a comparison should be made between the optimized plans of action with those plans obtained from human, heuristic, or other military sources and detail the differences, if any. A set of scenarios should be played wherein a human competes against the evolutionary program. In addition, tests should also be made to compare performance between the EP program and any other available heuristic planning resource. This will provide much needed information to definitively demonstrate the effectiveness of this optimization technique.

Evaluating scenarios where evolutionary programming controls all teams in the scenario will prove highly instructive. It is possible to implement various degrees of intelligence in many ways. One simple mechanism is to let the user select the number of iterations for each team in the scenario. Setting the number of iterations (and/or population size) for a team to a large value would imply higher intelligence (a larger portion of the parameter space is searched), hence generation of better solutions. Setting this team parameter to a small number of iterations would imply lesser optimization, and lower intelligence of the behavioral planner. Other potential mechanisms include limiting or augmenting prediction capabilities, and implementation of modified versions of the scoring function. This last technique could utilize a limited amount of knowledge in the scoring function (i.e., joint VSS) or possibly evolve the predicted behaviors for opposing teams as part of the solution for selected teams.

## References


Atmar, W., 1992, On the rules and nature of simulated evolutionary programming, *Proceedings of the First Annual Conference on Evolutionary Programming*, San Diego, CA: Evolutionary Programming Society, pp. 17-26.

Fogel, D.B. (1995) *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*, Piscataway, NJ: IEEE Press.

Fogel, L.J., 1964, On The Organization of Intellect, Ph.D. dissertation, University of California, Los Angeles.

Fogel, L.J., Owens, A.J., and Walsh, M.J. (1966) *Artificial Intelligence through Simulated Evolution*, New York, NY: John Wiley.

Fogel, L.J. (1995) "The Valuated State Space Approach and Evolutionary Computation for Problem Solving," *Computational Intelligence: A Dynamic System Perspective*, M. Palaniswami, Y. Attikiouzel, R.J. Marks, D. Fogel, and T. Fukuda, (eds.), IEEE Press, NY, pp. 129-136.

Goldman, S.R. (1996) "Knowledge Acquisition and Delivery: Constructing Intelligent Software Command Entities," *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*, D.E. Mullally (ed.), Orlando, FL, pp. 31-36.

Hocaoglu, C., and Sanderson, A. (1996) "Planning Multi-Paths using Speciation in Genetic Algorithms," *Proc. 1996 IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, May, pp. 378-383.

McDonnell, J.R., and Page, W.C., (1990) "Mobile Robot Path Planning Using Evolutionary Programming", *Proc. of the 24th Asilomar Conference on Signals, Systems, and Computers*, Vol. 2, Maple Press, San Jose, CA, pp. 1025-1029.

Porto, V.W., (1998) Evolving Integrated Low-Level Behaviors into Intelligently Interactive Simulated Forces, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, San Diego, CA: Springer Verlag, Berlin.

Porto, V.W., and Fogel, L.J. (1996) "Simulating an Intelligently Interactive Adversary", Final Report for STRICOM, Contract number N61339-95-C-0088, Natural Selection, Inc.

Porto, V.W., and Fogel, L.J. (1997) "Evolution of Intelligently Interactive Behaviors for Simulated Forces", *Proceedings of the Sixth International Conference on Evolutionary Programming*, P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (eds.), Springer Verlag, pp. 419-429.

Page, W.C., McDonnell, J.R., and Anderson, B. (1992) "An Evolutionary Programming Approach to Multi-Dimensional Path Planning," *Proc. of the First Annual Conference on*

*Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, San Diego, CA, pp. 63-70.

Rajput, S., and Karr, C. (1996) "A New Mechanism for Cooperative Behavior in ModSAF," *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*, D.E. Mullally (ed.), Orlando, FL, pp. 189-199.

Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S., and Schwamb, K. (1995) "Intelligent Agents for Interactive Simulation Environments," *AI Magazine*, Vol. 16(1), pp. 15-40.

Xiao, J., Michalewicz, Z., Zhang, L. and Trojanowski, K. (1997) "Adaptive Evolutionary Planner/Navigator for Mobile Robots," *IEEE Trans. Evolutionary Computation*, Vol. 1, No. 1, April 1997, pp. 18-28.

Zobrist, A. L., (1969) "A Model of Visual Organization for the Game of GO," AFIPS Conf. Proc., 34, pp.103-112.

## Appendix A: Overview of Evolutionary Programming

Evolutionary programming (EP) is one of a class of paradigms for simulating evolution which uses the concept of Darwinian evolution to iteratively generate increasingly appropriate solutions (the behavior of organisms) in light of a static or dynamically changing environment. This is in sharp contrast to earlier artificial intelligence research which largely centered on the search for simple heuristics (generally useful rules). Instead of developing a (potentially) complex set of rules derived from human experts, evolutionary programming evolves a set of solutions which exhibit optimal behavior with regard to the environment and specified payoff function. In a most general framework, evolutionary programming may be considered an optimization technique wherein the algorithm iteratively optimizes behaviors, parameters, or other constructs. As in all optimization algorithms, it is important to note that the point of optimality is completely independent of the search algorithm, and is solely determined by the adaptive topography (i.e., response surface) (Atmar, 1992).

In its standard form, the basic evolutionary program uses the four main components of all evolutionary computation algorithms (EC): initialization, variation, evaluation (scoring), and selection. At the basis of this, as well as other EC algorithms, is the presumption that, at least in a statistical sense, learning is encoded phylogenically versus ontologically in each member of the population. 'Learning' is a byproduct of the evolutionary process as successful individuals are retained through stochastic trial and error. Variation (e.g., mutation) provides the means for moving solutions around on the search space, preventing entrapment in local minima. The evaluation function directly measures fitness, or equivalently the behavioral error, of each member in the population with regard to the environment. Finally, the selection process probabilistically culls suboptimal solutions from the population, providing an efficient method for searching the topography.

The basic evolutionary programming algorithm starts with a population of trial solutions (e.g. plans of action) which are initialized by random, heuristic, or other appropriate means. The size of the population, $\mu$, may range over a broadly distributed set, but is in general larger than one. Each of these trial solutions is evaluated with regard to the specified fitness function. After the creation of a population of initial solutions, each of the parent members is altered through application of a mutation process; in strict evolutionary programming, recombination is not utilized. Each 'parent' member *i* generates $\lambda_i$ progeny which are replicated with a stochastic error mechanism (mutation). Mutations are chosen with respect to a probability distribution, typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution or it may be fixed *a priori*. These offspring solutions are then evaluated over the existing environment in the same manner as their parents.

After the fitness or behavioral error is assessed for all offspring solutions, the selection process is performed by one of several general techniques including: 1) the best $\mu$ solutions are retained to become the parents for the next generation (elitist), or, 2) $\mu$ of the best solutions are statistically retained (tournament), or 3) proportional-based selection. In most applications, the size of the population remains constant, but there is

no restriction in the general case. The process is halted when either the solution reaches a predetermined quality, a specified number of iterations has been achieved, or some other criteria (e.g., sufficient convergence) stops the algorithm. Figure A1 diagrams this process pictorially.

t:=0;
initialize $P(0) := \{a'_1(0), a'_2(0), ..., a'_\mu(0)\}$
evaluate $P(0) : \{\Phi(a'_1(0)), \Phi(a'_2(0)), ..., \Phi(a'_\mu(0))\}$
iterate
{
mutate: $P'(t) := m_{\Theta_m}(P(t))$
evaluate: $P'(t) : \{\Phi(a'_1(t)), \Phi(a'_2(t)), ..., \Phi(a'_\lambda(t))\}$
select: $P(t+1) := s_{\Theta_s}(P'(t) \cup Q)$
t := t + 1;
}

where

$a'$ is an individual member in the population
$\mu \geq 1$ is the size of the parent population
$\lambda \geq 1$ is the size of the offspring population
$P(t) := \{a'_1(t), a'_2(t), ..., a'_\mu(t)\}$ is the population at time t
$\Phi: I \rightarrow \Re$ is the fitness mapping
$m_{\Theta_m}$ is the mutation operator with controlling parameters $\Theta_m$
$s_{\Theta_s}$ is the selection operator $\ni s_{\Theta_s}\left(I^\lambda \cup I^{\mu+\lambda}\right) \rightarrow I^\mu$
$Q \in \{\varnothing, P(t)\}$ is a set of individuals additionally accounted for in the selection step,
i.e. parent solutions.

**Figure A1: The evolutionary programming paradigm**

Evolutionary programming differs philosophically from other evolutionary computational techniques such as genetic algorithms in a crucial manner. Evolutionary programming is a top-down versus bottom-up approach to optimization. It is important to note that (according to neo-Darwinism) selection operates only on the phenotypic expressions of a genotype; the underlying coding of the phenotype is only affected indirectly. The realization that a sum of optimal parts rarely leads to an optimal overall solution is key to this philosophical difference. Genetic algorithms rely on the identification, combination, and survival of "good" building blocks (schemata) iteratively combining to form larger "better" building blocks. In a genetic algorithm, the coding structure (genotype) is of primary importance as it contains the set of optimal building blocks discovered through successive iterations. The building block hypothesis is an implicit assumption that the fitness is a separable function of the parts of the genome. This successively iterated local

optimization process is different from evolutionary programming which is an entirely global approach to optimization. Solutions (or organisms) in an evolutionary programming algorithm are judged solely on their fitness with respect to the given environment. No attempt is made to partition credit to individual components of the solutions. In evolutionary programming (and in evolution strategies), the variation operator allows for simultaneous modification of all variables at the same time. Fitness, described in terms of the behavior of each population member, is evaluated directly, and is the sole basis for survival of an individual in the population. Thus, a crossover operation designed to recombine building blocks is not utilized in the general forms of evolutionary programming.

## Appendix B: The Valuated State Space Approach

*Making Missions Well Defined: The Valuated State Space*

A mission can be defined in quantitative terms using the Valuated State Space Approach [L. Fogel, 1995]. This approach can be used to express any mission in terms of the relative importance of each of the significantly different outcomes, and therefore can be used to measure the overall worth of the current and prospective situations. To illustrate, suppose three dimensions are of concern: x, y, and z. These dimensions have relative importance of, say, six, nine, and two, respectively (with reference to a 10-scale, where 10 indicates paramount importance, 5 indicates moderate importance, and 0 indicates no importance whatsoever). As depicted in Table 1, the lines following the parameter designators indicate the class intervals of significantly different degrees of achievement. Specific thresholds identify the limit of each class interval (although these are not shown here as specific measures). Values on a ratio or magnitude scale are attributed to each of the class intervals.

In general, the number of class intervals reflects the relative importance of the parameter being described; that is, the more important the parameter, the greater the specificity of its measure. Usually, a parameter of little importance may be adequately specified in a binary sense, as is the case with parameter z. By convention, the class intervals are arranged in order of decreasing worth in Table 1.

**Table 1**. A Valuated State Space

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | x | 3 | 1√ | 0 | | |
| 9 | y | 10 | 9 | 2√ | 1 | 0 |
| 2 | z | 10 | 0√ | | | |

$\sum = 17$

The number of states in the space is the product of the number of class intervals on each of the dimensions (parameters), in this case, 30. Achieving the most valuable class interval on each of the parameters corresponds with the state of the highest overall worth (a measure of 1.0, or 10 on a 10 scale, 100 on a percent scale). Achieving no success on a parameter corresponds with an overall worth of zero. (In situations where a negative worth is ordinarily associated with some class intervals, the scale can be linearly transformed so that the worst possible degree of achievement has zero value.) Any intermediate state has some value, depending upon the normalizing function. For example, taking the weighted arithmetic mean as that function, the state indicated by the check marks in Table 1 has a value as indicated:

$$w_0 = \left(\tfrac{1}{3}\right)\left(\tfrac{6}{17}\right) + \left(\tfrac{2}{10}\right)\left(\tfrac{9}{17}\right) + \left(\tfrac{0}{1}\right)\left(\tfrac{2}{17}\right) = 0.22 \text{ or } 22\% .$$

Suppose four dimensions are of concern, say, (1) Performance Reliability, (2) Maintainability, (3) Ease of Operation, and (4) Resistance to Enemy Countermeasures. These dimensions are given relative importance weights of 10, 8, 3, and 6, respectively. The class intervals of significantly different degrees of achievement are shown under each of these parameters in Table 2, these spanning the range from the least desirable to the most desirable achievement. The check marks indicate the degrees of achievement that define the state being measured.

**Table 2.** A Sample Valuated State Space

10 Performance Reliability:

| | | |
|---|---|---|
| | 10 | $\geq$ 99% |
| | 9 | $\geq$ 95% but < 99% |
| | 8 | $\geq$ 90% but < 95% |
| √ | 6 | $\geq$ 80% but < 90% |
| | 4 | $\geq$ 60% but < 80% |
| | 1 | $\geq$ 40% but < 60% |
| | 0 | < 40% |

8 Maintainability:

| | | |
|---|---|---|
| | 10 | Easily maintained at field level by operational personnel |
| | 7 | Maintainable at field level by unit military technicians |
| √ | 5 | Maintained at field level with assistance of specialized military technicians |
| | 3 | Skilled civilian field technicians required for all but minor procedures |
| | 1 | Must be returned to factory or depot for all but minor, routine procedures |
| | 0 | Non-repairable |

3 Ease of Operation:

| | | |
|---|---|---|
| | 10 | Routinely operated by regular troops |
| | 8 | Operated by regular troops directed by specially trained supervisor |
| √ | 4 | Operated by specially trained crew |
| | 1 | Operated only by highly technical and highly skilled personnel |
| | 0 | Inoperable |

6 Resistance to Enemy Countermeasures:

| | | |
|---|---|---|
| | 10 | Not susceptible to countermeasure |
| | 8 | Has self-contained capability to defeat enemy countermeasures |
| √ | 5 | Requires special support forces to suppress enemy countermeasures |
| | 3 | Highly degraded by sophisticated countermeasures |
| | 0 | Easily defeated by simple countermeasures |

Taking the weighted arithmetic mean, the overall worth of this situation is

$$w_0 = \left(\tfrac{6}{10}\right)\left(\tfrac{10}{27}\right) + \left(\tfrac{5}{10}\right)\left(\tfrac{8}{27}\right) + \left(\tfrac{4}{10}\right)\left(\tfrac{3}{27}\right) + \left(\tfrac{5}{10}\right)\left(\tfrac{6}{27}\right) = 0.5259 \text{ or } 52.59\%$$

By improving Performance Reliability from the 4th to the 5th class interval (i.e., from between 80% and 90% to between 90% and 95%), increases the overall worth to;

$$w_0 = \left(\tfrac{8}{10}\right)\left(\tfrac{10}{27}\right) + \left(\tfrac{5}{10}\right)\left(\tfrac{8}{27}\right) + \left(\tfrac{4}{10}\right)\left(\tfrac{3}{27}\right) + \left(\tfrac{5}{10}\right)\left(\tfrac{6}{27}\right) = 0.5999 \text{ or } 59.99\%$$

Note that statistical independence (or lack of correlation) of the subparameters is neither necessary nor sufficient for the appropriateness of the weighted arithmetic mean. What *is* necessary and sufficient is that the overall utility (worth) be *preferentially independent* of the subparameters.

In many situations, any level of achievement has some overall worth. In others, all the parameters are critical. Failure in any single regard nullifies any other contribution. In this case, it is appropriate to use the weighted geometric mean, the overall worth being the product of the normalized degrees of achievement, each raised to the power of the normalized importance of that parameter. Other methods for combining the contributions are appropriate for different situations. In any case, a multi-attribute utility function specifies the overall worth of any particular situation; that is, the Valuated State Space and its normalizing function yields a single overall measure for the worth of each significantly different situation.

The examples provided are single-level Valuated State Spaces. In practice, the main subparameters (i.e., Performance Reliability, Maintainability, etc.) would be explicated in terms of lower-level subparameters. Only the lowest levels can be calculated from below. It is also worthwhile to recall that the weighted arithmetic mean is always greater than or equal to the weighted geometric mean. The former simple calculation provides an optimistic view of the real value.

In reality war is never a one-player game. Almost always, decisions are best made in the light of the other player's perceived, known, or assumed intent, capabilities, and motivation. It is, therefore, suitable to construct a similar representation of the purpose of each of the other players, then examine the joint state space that defines the game. This portrays a finite number of possible situations, those situations that are significantly different from any single or multiple players' points of view. There is a joint payoff in each cell/state for each of the players, this being a function of their marginal worth. Every sequence of moves and countermoves corresponds with a trajectory across state in the joint state space, there being some overall worth for that series of transitions. Note that the payoff function need not be symmetric, nor must the game itself be zero-sum. The joint state space reflects the motivation (degree of resolve or commitment) of the individual players as well as their mutual attitudes.

Natural Selection, Inc. has had sufficient experience to ensure the practicality of this approach. After being briefed on the underlying logic, influential individuals within the organization are interviewed to reveal their views of the purpose to be achieved. Certain rules-of-thumb assist this process. For example, the more important parameters are measured in terms of a larger number of class intervals. The resulting viewpoints are compared. Any significant differences are resolved so that a final draft can be approved

by those responsible. This statement of purpose then indicates what to measure, with what specificity, how to fuse the data into the overall degree of achievement and, in addition, provides a listing of the remaining problems by priority.

☆U.S. GOVERNMENT PRINTING OFFICE: 2002-710-038-10228

## *MISSION*
## *OF*
## *AFRL/INFORMATION DIRECTORATE (IF)*

*The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.*