Symbiotic Adaptive Multisimulation

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Bradley Mitchell

Certificate of Approval:

Alice E. Smith
Professor and Chair
Department of Industrial and Systems Engineering

Levent Yilmaz, Chair
Assistant Professor
Department of Computer Science and Software Engineering

Drew Hamilton
Associate Professor
Department of Computer Science and Software Engineering

George T. Flowers
Interim Dean
Graduate School

Symbiotic Adaptive Multisimulation

Bradley Mitchell

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
December 17, 2007

Symbiotic Adaptive Multisimulation

Bradley Mitchell

Signature of Author

Date of Graduation

Thesis Abstract

Symbiotic Adaptive Multisimulation

Bradley Mitchell

Master of Science, December 17, 2007
(B.S., Auburn University, 1999)
(B.A., Auburn University, 1995)

78 Typed Pages

Directed by Levent Yilmaz

Systems characterized by non-linear interactions among diverse agents often exhibit emergent behavior that may be very different from what the initial conditions of these systems would suggest. Traditional simulation techniques that rely on accurate knowledge of these conditions typically fail in these cases. The goal of Symbiotic Adaptive Multisimulation (SAMS) is to enable robust decision making in real-time for these problems. Rather than rely on a single authoritative model, SAMS explores an ensemble of plausible models, which are individually flawed but collectively provide more insight than would be possible otherwise.

The insights derived from the model ensemble are used to improve the performance of the system under study. Likewise, as the system develops, observations of emerging conditions can be used to improve exploration of the model ensemble. In essence, a useful coevolution between the physical system and SAMS occurs. In this thesis, an outline of the core techniques of SAMS is provided. In addition, a parallel simulation application for the study of autonomous Unmanned Aerial Vehicle (UAV) teams was developed. Experimental

results from this application are presented and their implications for further study are discussed.

Acknowledgments

To my advisor and the other members of my committee: Dr Yilmaz, Dr. Smith, and Dr. Hamilton, Thank you for your encouragement, advice, and patience.

To Dr. Dozier, Thank you for your generous help, time, and insight.

To my lovely wife, Preeti, Thank you for your kindness, wisdom, and steadfast support. I would not have made it without you.

To my parents, Thank you for instilling in me a love of science at an early age.

To my Jack Russell: "Satchi", Thanks for hanging in there with me during all those late nights of writing code.

Style manual or journal used Publication Manual of the American Psychological Association (together with the style known as "aums").

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Dynamic updating of simulation models is a key requirement for using simulation as a tool to improve systems in which information only becomes available once the system is in progress (Yilmaz & Ören, 2004). In these types of systems, the initial conditions provide little or no insight into how the system may develop over time. Emergent behavior that arises dynamically is a primary source of information in these systems. Exploiting this information to enable robust decision making in a timely manner requires the ability to observe the system in real-time and adapt useful characteristics for the system with as little computational effort as possible.

Symbiotic Simulation (S2) involves the use of simulation systems that are synchronized with the physical systems they model to enable mutually beneficial adaptation (Fujimoto, Lunceford, Page, & Uhrmacher, 2002). In S2, simulation outputs are examined and used to determine how the physical system may be optimized. Similarly, measurements from the physical system are used to validate the simulation. When uncertainty in the physical system is present, multiple "what-if" simulation experiments can be helpful in adjusting the physical system. However, since the number of "what-if" experiments that may be performed is limited by both computational and real-time constraints, the ability to conduct an efficient search of the model space is essential.

As evidence of current interest in S2 techniques, extensive research in this field is already being applied to the problem of coordinating Unmanned Aerial Vehicles (UAVs) by the Swedish Defence Research Agency (Lozano, Kamrani, & Moradi, 2006). This problem

is significant because the physical system is distributed across a number of agents with varying degrees of autonomy that must perform in a dynamic and uncertain environment. Much of the research in the field of cooperative UAV control has focused on optimizing team behavior for a narrow range of conditions. Given the dynamic and uncertain nature of this problem however, it may be better to emphasize robust systems capable of handling a diverse range of conditions. In order to meet this challenge using S2, any search of a model space should be exploratory in nature, covering a large range of possible circumstances while still providing an efficient system configuration.

Symbiotic Adaptive Multisimulation (SAMS) is intended as an S2 technique appropriate for physical systems characterized by distributed, dynamic, and uncertain conditions. It is heavily inspired by the fields of Multisimulation (Yilmaz, 2007), Exploratory Analysis (Davis, 2000), and Exploratory Modeling (Bankes, 2005) which all involve the use of an ensemble of plausible models to provide insight in the absence of a single authoritative model. The salient feature of SAMS is the use of a Genetic Algorithm (GA) to evolve the model ensemble in response to changes in the physical system. With this feature, it was theorized that an efficient search of an uncertain model space would be possible, thus permitting synchronization with the physical system.

The goal of this thesis is to formulate a description of SAMS as an S2 methodology and to examine its utility involving a distributed, dynamic, and uncertain system. Toward this end, a parallel application for simulation experiments has been developed along with a simple, proof-of-concept structural model of UAV behavior. This model is parameterized to allow creation of a virtually infinite number of distinct models. Experiments with this application with an emulator of the system under study have been conducted to assess

the potential effectiveness of SAMS. They appear to show that SAMS can provide an improvement in system performance when it is coupled with a dynamic system.

In Chapter 2, various related methodologies and supporting techniques are examined. In Chapter 3, a Symbiotic Adaptive Simulation is fully described as a simulation methodology. In Chapter 4, the SAMS Parallel Application itself is discussed. Finally, in Chapter 5, an agent-based model of autonomous UAV behavior, which was used in conjunction with the Parallel Application is described. The experiments with this system are also presented. The thesis concludes with Chapter 6.

Chapter 2

Literature Review

2.1 Symbiotic Simulation

Symbiotic Simulation was originally described at a Dagstuhl Grand Challenges workshop in 2002 (Fujimoto et al., 2002) although the idea of using simulation to enhance the real time performance of a system had previously existed in the form of on-line simulation. Since that time, two noteworthy efforts in S2 have examined its usage in the context of semiconductor manufacturing (Low et al., 2005) and UAV path planning (Lozano et al., 2006). In addition, there is also an effort underway to create a generic architecture for symbiotic simulation (Ayani, 2007).

In (Low et al., 2005), the use of S2 was motivated by the need for a more rapid simulation analysis technique. The semiconductor industry is characterized by short product life cycles and complex manufacturing processes that are highly sensitive to disruption. As such, the time required to perform conventional simulation analysis typically exceeds production constraints. The authors of this study implemented a proof-of-concept system in which online agents monitor the production line for potential delays and perform what-if simulation experiments to examine the consequences of adjusting various portions of the production line using an iterative search to compensate. Although the production life cycles examined in this paper occurred over several months, the high degree of fidelity required by the simulations meant that only a small number of experiments could be carried out.

In (Lozano et al., 2006), S2 is examined as a means of providing decision support for UAV path planning. Although the research in this paper is not yet complete at the time

of this writing, it appears that the authors are implementing a high fidelity model of UAV team behavior. The means for selecting "what if" experiments is a transition model in which experiments branch from the current system state according to a physics model and known enemy doctrine.

2.2 Uncertainty and Plausible Models

The S2 approaches described above use models that are believed to be accurate and valid descriptions of a physical system. In problem domains characterized by a large degree of uncertainty, it may not be possible to construct such models. For these cases, the preferred approach is to use an ensemble of models that need not be accurate and valid, but only plausible. This is a key feature of the fields of Exploratory Modeling, Exploratory Analysis, and Multisimulation. In (Bankes, 1998) plausible models are described as predicting "how the system would behave if the assumptions the model is based on were true". When model excursions are viewed collectively, the behavior of plausible models can be informative despite the flaws of each individual model. Because of the presence of uncertainty, there may be many plausible models that could represent a system (Bankes, 1993). Conversely, knowledge of the system limits the set of plausible models. Multisimulation, Exploratory Analysis and Exploratory Modeling all experiment with ensembles of models as experimentation with a single plausible model would be just as likely deceptive as informative.

Among a set of plausible models, variation occurs according to input uncertainty and structural uncertainty (Davis, 2000). Input uncertainty which deals with the possible values that model input factors may assume can be further subdivided into model uncertainty and

parameter uncertainty (Henderson, 2003). Model uncertainty is dependent on the choice of distribution function for a particular input whereas parameter uncertainty relates to the choice of parameters that govern the shape of those distributions. Classical experimental design techniques can be useful in dealing with input uncertainty (Barton, 2004) provided the factors to be examined are relatively few in number and their interactions are linear. Structural uncertainty can be much more challenging to deal with. This includes such things as the actual choice of variables used to represent a system. It may arise from difficulty in analyzing a system or even from differences of expert opinion.

Because of the variation that can occur across a set of plausible models, methods for consistently representing them have been of interest. In the fields of Exploratory Modeling and Exploratory Analysis, there has generally been a focus toward parameterizing both kinds of uncertainties. In the field of Multisimulation, there has been an emphasis toward the use of an emerging multimodel formalism. An examination of the kinds of models described by this formalism can be found in (Yilmaz, Ören, & Ghasem-Aghaee, 2006). In general, the multimodel formalism provides a specification of how models may vary in relation to each other. This specification is necessary to permit dynamic replacement and updating of models which is an area of specific interest in Multisimulation.

Multiresolution Modeling (MRM) is a type of multimodel that is likely to be useful in SAMS. MRM has been extensively studied in the fields of Exploratory Modeling, Exploratory Analysis, and Multisimulation. MRM is described in (Bigelow & Davis, 1999) as "constructing a model or family of models that describe consistently the same system or process at different levels of resolution". In that paper, the authors outline several motivations for MRM. With respect to SAMS, the two most important of these are economy

and dealing with chaos. In the first case, exploring a set of models in a high level way before examining certain phenomena in more detail can make efficient use of computational resources. Chaos on the other hand means that minute details may result in significantly different end states of a system, even when the initial states of two models appear similar. Davis and Bigelow point out that chaos is common in systems with non-linear dynamics. MRM goes against the usual technique of modeling a system from the bottom up. It suggests that models with varying amounts of detail can be informative in dealing with chaos and thus mutually calibrating. This notion is also discussed in (Davis & Bigelow, 2003). One issue raised there is that of aggregation and disaggregation of input factors. In order to maintain consistency between levels of resolution, there needs to be an appropriate mapping between factors in low resolution models to their constituents in high resolution models.

While multimodel techniques like MRM are helpful in dealing with the variation among a set of plausible models, the size of a typical ensemble can pose equally difficult challenges in terms of computational resources. Knowledge of the system under inquiry can reduce the size of the model ensemble, but even with extensive knowledge, the number of plausible models could still be infinite (Bankes, 1993). In light of this problem, effective techniques for sampling a model ensemble are of interest. Bankes asserts that the choice of technique is largely dependent on the purpose of the study and for models that maximize certain outputs of interest, a search strategy should be used.

Additional insight into the problem of sampling model ensembles can be found in (Davis, 2000). The author makes a distinction between controllable and uncontrollable input factors of a model. In general, controllable inputs are considered to be the factors that a decision maker can change within the physical system in response to observations

made within a simulation study. Uncontrollable factors represent environmental conditions within which a proposed system operates. This distinction is also mentioned on page 620 of (Law, 2007). For each type of factor, a different sampling technique is appropriate. Davis asserts that for uncontrollable inputs, Probabilistic Exploration should be used. With Probabilistic Exploration an attempt is made to identify a suitable distribution function for each uncontrollable input. The choice of function is often based on the subjective knowledge of the analyst. For the controllable input factors of a model, a Parametric Exploration strategy is suggested. In Exploratory Analysis, Parametric Exploration involves a combination of classical experimental design with software visualization tools. Using such tools, an analyst is capable of identifying significant interactions between factors in a matter of minutes. Davis refers to this strategy of using different techniques to explore controllable and uncontrollable inputs as Hybrid Exploration.

2.3 Evolutionary Algorithms

Effective sampling of a model space of potentially infinite size requires a robust and efficient search algorithm. Such an algorithm should be capable of broad exploration while uncertainty is pervasive within the model space. This requirement implies the need for some stochastic element to deal with the combinatorial explosion of potential configurations (Dreo, Petrowski, Siarry, & Taillard, 2006). Conversely, if uncertainty is decreasing, an appropriate search algorithm should also be capable of rapid exploitation of increasing information. This requirement rules out a simple random walk of the search space. In addition, since the search space itself involves a set of models and not a differentiable

function, any potential search technique must not rely on derivatives or other gradient techniques. This eliminates calculus-based methods such as hill-climbing (Goldberg, 1989).

The field of Evolutionary Computation (EC) provides several techniques for implementing search algorithms with effective exploration and exploitation properties. These algorithms model the processes of Darwinian evolution as a means of accomplishing a robust stochastic search. The term Evolutionary Computation itself was coined in 1993 as a class of methods originally including Evolutionary Programming, Genetic Algorithms, and Evolution Strategies (Fogel, 2006). Prior to that time, these three algorithms were studied largely in isolation and each was characterized by distinct differences from the others. The incorporation of these three techniques under the rubric of EC has encouraged a tremendous cross-fertilization of ideas. As a result, it has now become common to refer to any algorithm from the field of EC as an Evolutionary Algorithm (EA) (Dreo et al., 2006).

EAs employ multiple candidate solutions, collectively referred to as a population. This population is evolved over a number of generations, with the evolutionary process favoring solutions of increasing fitness. The exact means of accomplishing this evolution varies depending on the design of a particular EA. However, all techniques rely on some degree of selection pressure to determine which individuals survive and reproduce, thus guiding the evolutionary process (Dreo et al., 2006). Increasing selection pressure results in more rapid improvement to population fitness. Too much selection pressure, however, can result in convergence to a sub-optimal solution in multimodal problem spaces. Decreasing selection pressure can enable better exploration of the problem space and the eventual discovery of the global optimum. If there is too little selection pressure, genetic drift can occur. This

results in an increasingly homogeneous population of inferior candidate solutions and is caused by sampling error resulting from the stochastic search process (DeJong, 2006).

In order to explore additional candidate solutions outside the initial population, diversity must be introduced. This may be done by means of either sexual recombination or asexual mutation. As the name suggests, recombination (also referred to as crossover) uses information gained from the solutions of multiple parent individuals to create one or more children with characteristics of each of the parents. It offers a powerful means of rapidly exploring the solution space, particularly in dynamic environments (Fogel, 2006). Mutation on the other hand can introduce fine grained variation that is usually impossible with recombination alone, but it has a tendency to disrupt improvements from recombination when used excessively (DeJong, 2006).

The encoding of candidate solutions, or individuals, is an important consideration in the design of an EA. The field of Genetic Algorithms, inspired directly by biological genetics makes a distinction between the genotype, which serves as blueprint for the creation of an individual and its phenotype, the resulting individual formed by the interaction of the genotype and the surrounding environment (Goldberg, 1989). In GAs, the data structure encoding the genotype is referred to as a chromosome, with specific characteristics referred to as genes. Early examples of GAs described by John Holland (Holland, 1975) and his students used a string of bits to encode the genotype of individuals but in recent years it has become more common to use real-valued encodings for problems in the continuous domain (Fogel, 2006).

2.4 Adaptive Agents

Learning agents can be considered to possess two major computational features, namely a performance element, and a learning element(Russell & Norvig, 2003). The performance element represents a system of rules that govern an agent's behavior at a single instant. These rules may be modified by the learning element over time as the agent receives information from its environment and other agents. In the context of Complex Adaptive Systems, learning agents have been referred to as Adaptive Agents (Holland, 1995).

The notion of Complex Adaptive Systems is relevant to this research as much of the work in Exploratory Modeling and other agent-based simulation approaches have been directly applied to problems in this area of study(Yilmaz, 2007). A useful definition of what constitutes a Complex Adaptive System can be found in (Singer, 1995). In this article, a Complex Adaptive System is defined as having a large number of diverse agents that produce an aggregate behavior through non-linear interactions. Furthermore, the diversity of agents is robust to change and may evolve to produce new specializations of agent behavior.

Self-organization is a kind of aggregate behavior that is often associated with Complex Adaptive Systems. Self-organization as a property of an engineered system has been described as being one in which individual agents or units respond to local stimuli to achieve through a division of labor the efficient performance of some task (Collier & Taylor, 2004). The collective efficiency of this task performance must be greater than what could be accomplished individually. Additionally, self-organization involves the creation of an equilibrium state that arises from the local interactions of agents (Namatame & Sasaki, 1998). This equilibrium may be achieved through either competition or cooperation.

Particle Swarm Optimization (PSO) is one possible technique for implementing the learning element of an adaptive agent. A broad introduction of PSO can be found in (Kennedy & Eberhart, 2001). PSO is a continuous numeric optimization technique in which a potential solution to a problem is characterized as a point in some n-dimensional space, with the number of dimensions being equal to the number of decision variables. As the name suggests, PSO uses a population of potential solutions. These solutions "fly" through the problem space over time. As each particle moves through the problem space, it records the best solution, *pbest* that it has found so far as well as the best solution, *gbest* discovered by the other members of the swarm. Particles tend to gravitate toward these two positions over time as they search for better solutions.

In (Bratton & Kennedy, 2007), a precise specification of the canonical PSO is given. In this specification, each dimension d of the velocity vector $\overrightarrow{v}$ of a particle i is calculated according to equation 2.1. In this equation, it can be seen that $\overrightarrow{v}$ is a function of the particle's current position x, personal best p_i, and global best p_g. A constriction factor χ is included to prevent explosive growth of the particle's velocity. In addition, randomness is included by ϵ_1 and ϵ_2 which are independent uniformly random variables that govern cognition and social influence respectively. The constants c_1 and c_2 affect the relative contributions from the cognition and social components. These components also appear in many sources listed as φ_1 and φ_2.

$$\overrightarrow{v}_{id} = \chi(v_{id} + c_1\epsilon_1(p_{id} - x_{id}) + c_2\epsilon_2(p_{gd} - x_{id})) \tag{2.1}$$

The social aspect of PSO presents intriguing possibilities for use in multi-agent systems. One example can be found in (Hereford, 2006) where the operation of a PSO was distributed across a number of simulated robots in a 2-dimensional search problem. Each simulated robot acted as a particle with decision variables being the x and y coordinates of the search space. Each robot would attempt to move toward the position specified by its particle. As a robot moved from one position to the next, it would attempt to detect a hidden target. Detection was modeled according to a sphere function and thus as robots moved closer to the target, the fitness of their particle solutions improved. These results were broadcast to other robots resulting in social coordination.

The social influence within a particle swarm propagates according to the communication topology of its members. An in depth discussion of this topic can be found in (Richards & Ventura, 2003). Richards and Ventura performed experiments with the two most common topologies (star and ring) and presented a dynamic topology as well that begins as a ring topology and adds communication links over time. Star topologies rapidly converge to a good, but not necessarily optimal solution. On the other hand, ring topologies tend to be more successful in discovering a global optimum (especially for nonlinear functions with numerous local optima) but at the cost of additional search time (Bratton & Kennedy, 2007). The authors hypothesized that increasing the number of communication links over time would allow the PSO to perform a good initial exploration of the problem followed by a rapid local search and their experiments provide some evidence to support this. Beyond these observations, a dynamic communication topology may have interesting consequences for agents interacting in an environment where signal attenuation and interference is pervasive.

Chapter 3

Conceptual Framework and Methodology

3.1 Complex Adaptive Systems

Symbiotic Adaptive Multisimulation is intended as a technique for symbiotic simulation of problem domains characterized by massive uncertainty in a dynamic environment. Such problems may possess one or more characteristics of a Complex Adaptive System. In particular, an appropriate problem for SAMS involves autonomous agents that produce an aggregate behavior through non-linear local interactions. This aggregate behavior is self-organizing to the extent that the collective performance of the physical system's task is enhanced through cooperation or competition among these agents. Additionally, these systems may have real time constraints that require the ability to vary model resolutions based on the computing resources available. Possible examples of problems appropriate to SAMS include emergency response, and cooperative UAV behavior.

Because of the nonlinear interactions among autonomous agents in systems for which SAMS is intended, SAMS makes use of an ensemble of plausible models rather than a single authoritative model of system behavior. The nonlinear interactions in these types of systems often makes it impossible to create an accurate and detailed model of their behavior. It was also thought that the use of an ensemble of plausible models in SAMS would provide a rapid exploratory capability of the kinds of circumstances that may arise in an uncertain system.

3.2 Hybrid Exploration

Symbiotic Adaptive Multisimulation supports decision making by conducting a search through a potentially infinite space of models. The search works by evolving a set of potential system configurations for a dynamic set of environmental conditions. In order to efficiently search a potentially infinite number of plausible models, SAMS uses a hybrid exploration technique. As shown in Figure 3.1, uncontrollable inputs and controllable inputs are handled with an input analysis module and an output analysis module respectively. Measurements of the physical system's behavior are used to hypothesize distributions for uncontrollable inputs. As more details of the physical system's environment become known, the fidelity of these distributions with the actual values of the physical system should improve. Controllable input factors representing the configuration of the physical system are evolved using a genetic algorithm. A set of controllable inputs that completely describes a potential system configuration can be thought of as an individual. The controllable factors therefore, are considered to be the decision variables of a given problem while the uncontrollable factors determine the shape of a dynamic fitness landscape. An evolutionary algorithm enables the adaptation of individuals through a process of natural selection, with better performing individuals being more likely to survive and ultimately be used as the configuration settings for agents within the physical system itself.

3.2.1 Physical System Measurement

In order to hypothesize distributions for uncontrollable inputs and estimate their parameters, a means of observing the physical system is required. As real-time symbiotic simulation is performed, observations of the physical system enable more accurate estimates of

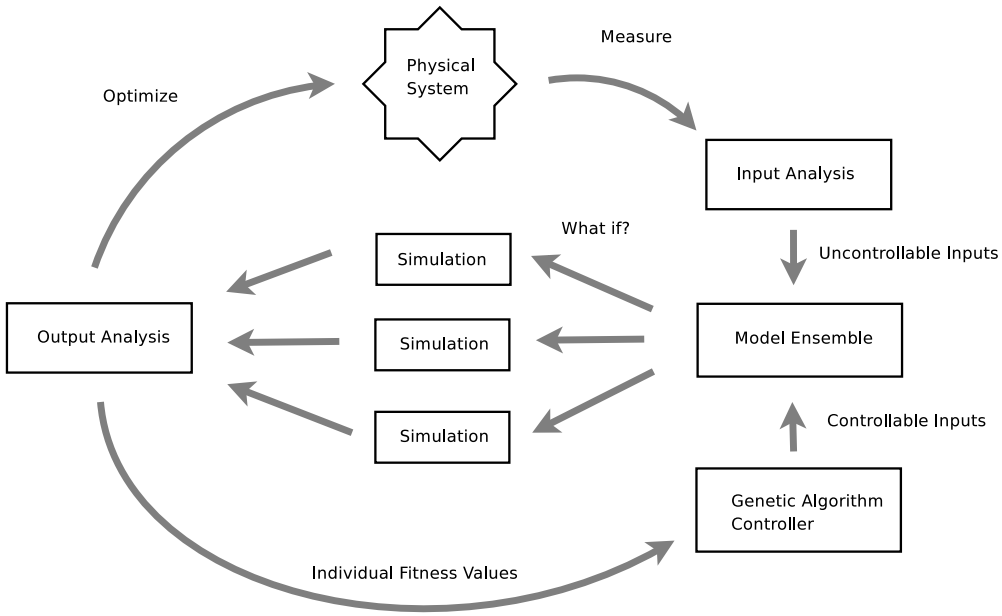


Figure 3.1: Symbiotic Adaptive Multisimulation

the input distribution parameters. With these improvements, the space of plausible models shrinks, allowing the system to simulate fewer models in greater detail. While exact methods for accomplishing this are beyond the scope of this thesis, the experiments presented in Chapter 5 appear to support the validity of these assumptions.

3.2.2 Partial Model Ensembles

Once a distribution has been hypothesized for each uncontrollable factor in the physical system, and the associated parameters for those distributions have been estimated, each factor is sampled multiple times. These samples are stored in a 2-dimensional array referred to as a Partial Model Ensemble (PME). Variates for each sampled distribution are associated on a per-sample basis and placed into the same row of an array. Each column of the array stores variates from a single input factor, and each row of the array is considered to be a

partial plausible model of the physical system. Later, these partial models are combined with potential system configurations to create fully specified plausible models for simulation.

One question regarding construction of the PME concerns the number of samples for each input factor that should be made. In the experiments performed for this research, PMEs with 30 rows were created. This number, while obtaining some benefit from the Central Limit Theorem was an arbitrary choice, and may be too small for most real-world applications. However, even with this limited number of samples, an apparent benefit from SAMS was realized. Details of these experiments are discussed in Section 5.5.

3.2.3 Genetic Search of Potential System Configurations

SAMS evolves potential system configurations for use within the physical system. As information is gained and improved PMEs are generated. Unlike the creation of a PME, however, a search of the space of potential system configurations requires the exploitation of a different type of system information, namely the performance characteristics of the configuration. Evolutionary Algorithms are very good at exploiting this type of information to improve a search as they are well suited for optimization problems (DeJong, 2006).

Among Evolutionary Algorithms, Genetic Algorithms have a feature useful to SAMS, which is the distinction between genotypic and phenotypic representation. A genetic encoding of the characteristics of a potential system configuration allows the same system to be interpreted in different ways and in varying levels of detail. This topic is discussed in greater detail in Section 3.2.5.

The structure of a typical GA is shown in Algorithm 1. An initial population of m individuals is randomly created in line 1 and the fitness values of the resulting individuals are

evaluated. The algorithm then enters a loop in which successive generations of individuals are evolved. An inner loop beginning on line 7 continues until a group of n children are created. In line 8, two individuals are chosen to be parents. Crossover combines the characteristics of both parents to create a new child. A mutation operator is then applied which may modify the child with characteristics not possessed by either parent. The fitness of the new child is then evaluated.

Algorithm 1 : A genetic algorithm.

```
 1: pop[m] ⇐ createInitialPopulation( )
 2: for i = 1 to m do
 3:         pop[i] ⇐ calculateFitness( pop[i] )
 4: end for
 5: repeat
 6:         children[n]
 7:         for i = 1 to n do
 8:                 parents[2] ⇐ selectParents( pop )
 9:                 children[i] ⇐ crossover( parents )
10:                 children[i] ⇐ mutatation( children[i] )
11:                 children[i] ⇐ calculateFitness( children[i] )
12:         end for
13:         pop[m] ⇐ selectSurvivors( pop, children )
14: until termination = true
```

Once the children have been created, the total population of individuals is then reduced back to m on line 8. Depending on the design of the GA, survivor selection may or may not involve direct competition between parents and children. The evolutionary process continues until some termination condition (such as a fixed number of fitness evaluations) is reached. Many variations of GAs are possible, including encodings and operators. Some discussion of the encoding used in the experiments for this thesis is presented in Section 3.2.5, while details of the selected genetic operators is postponed until Section 5.5.

In SAMS, potential system configurations are individuals created by the evolutionary process of a GA. The fitness of each individual is determined by the performance of its phenotype (the system itself) within multiple simulations. The fitness of the best individual at any generation is used to update the physical system. In order to make an assessment of fitness, each individual in the population must be simulated using a number of plausible models.

3.2.4 Combined Model Ensembles

In SAMS, a Combined Model Ensemble (CME) is a specification for conducting a series of simulation experiments involving a single individual from a population of potential system configurations. The goal of these experiments is to test an individual in multiple possible environments. Each simulation experiment examines the individual in the context of a set of uncontrollable factors representing a single possible environment. An objective fitness of the individual for a particular environment is obtained as an output from a simulation experiment. The resulting fitness values from all experiments with the individual in each respective environment are then averaged together to obtain an overall fitness for the individual which is used to determine its probability for reproduction and survival within the genetic search.

The layout of a CME is shown in Table 3.1. The possible environments with which the individual is to be tested are determined by a PME of Uncontrollable Factors making up the left hand side of the table. Copies of the individual are paired with each row of the PME. A single row of the CME thus includes both a sampled set of values from the uncontrollable factor distributions and a copy of the individual which specifies the settings

of the controllable model factors. Taken together, these two sets of elements form a single plausible model described by the entire row of values in the CME.

Uncontrollable Factors				Controllable Factors			
X_{11}	X_{12}	$\cdots$	X_{1m}	G_1	G_2	$\cdots$	G_ℓ
X_{21}	X_{22}	$\cdots$	X_{2m}	G_1	G_2	$\cdots$	G_ℓ
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
X_{n1}	X_{n2}	$\cdots$	X_{nm}	G_1	G_2	$\cdots$	G_ℓ

Table 3.1: A Combined Model Ensemble of n models is composed of a Partial Model Ensemble of m sampled variables, X, and an individual of ℓ genes, G. Each row represents a single model for which one or more simulation replications should be performed. Note that the same individual is used in each model.

Note that for a given CME, the same individual is paired with each row of the PME. Furthermore, the same PME is combined with each individual in the population to create a set of unique CMEs, one for each individual. Since the same PME is used in each CME, each individual is evaluated using the same environmental conditions. In essence, the uncontrollable factor settings of the PME, which change each time a new PME is created, become a dynamic fitness landscape.

3.2.5 Multiresolution Modeling with a Binary Representation

As mentioned in Section 2.2, an implementation of SAMS is likely to benefit from some degree of Multiresolution Modeling. Real-time constraints imposed by the systems for which SAMS is intended require the efficient use of computing resources. Additionally, the complexity of these systems makes it likely that there will be a high degree of variance in model outputs despite similar, even identical input settings. By creating multiple models that represent the same phenomena at varying levels of detail, MRM helps to solve both of these problems (Bigelow & Davis, 1999).

In a given situation, if the primary concern is to improve computational efficiency, the initial simulations within SAMS may all be run with low resolution models. This would permit a large exploration of possible conditions during early execution. Later, as information from the physical system becomes known and is used to update the distributions of uncontrollable factors, the system may switch to higher resolution models to examine fewer cases in greater detail.

When widely varying simulation outputs are a concern it might also be desirable to run models at multiple resolution levels concurrently. This would allow mutual calibration between resolutions, thus improving the overall accuracy of the system. Further research is needed to elaborate on how mutual calibration of varying model resolutions might be accomplished within a SAMS context. However, it is clear that before this problem can be examined, a mapping between model resolutions must be maintained during a search of the model space.

In the study of EAs, genotypic representation of individuals by means of a binary string has become less common in recent years. Nevertheless, a binary string representation seems promising in the context of SAMS. The fitness landscape within a CAS is likely to be subject to a large degree of noise, making the increased precision of real-valued encodings redundant. Secondly, a binary string representation permits the same code implementing genetic operators to be used at all resolution levels. This ability to use the same generic operator regardless of the actual genetic specification is one advantage of a binary encoding (DeJong, 2006).

As an example of these ideas one can consider the case of an autonomous UAV Search and Attack Model (which will be discussed in more detail in Section 5). This is an agent-based model in which each UAV within a team is specified separately with two variables. The first is a Boolean value indicating whether or not each UAV is a "Leader", to which other UAVs look for guidance. The second variable is a floating point value representing the "Cooperation Threshold" of a given UAV. Cooperation Threshold determines how likely the given UAV is to work closely with its team mates or else strike out on its own. The Leader variable is represented with a single bit. The Cooperation Threshold variable requires multiple bits depending on the degree of precision desired. In this case it was anticipated that the model would be subject to significant noise and so three bits are chosen to represent 8 possible values for this variable. In terms of canonical GAs, each of these variables constitutes a gene with the entire binary string making up a chromosome. Since UAVs are specified individually, the length of this string depends on the number of UAVs in the team.

One can also consider a hypothetical low resolution model of the same phenomenon. In this model, UAVs are not specified individually. There are only two variables of interest within this model: "Decentralization" and "Cooperativeness". The first variable is an abstract measurement of how leadership capabilities are distributed through the UAV team. It is a floating point value on the interval [0, 1] and is obtained by dividing the number of bits in the Decentralization gene equal to 1 by the length of the gene. The second variable measures the overall degree to which the UAV team tends to work together. In the high resolution model, Cooperativeness is considered individually for each UAV in the team as the Cooperation Threshold gene. In the low resolution model, however, Cooperativeness

applies to the team as a whole. The Cooperativeness variable is, like Decentralization, a floating point value on the interval [0, 1] and is obtained in the same way.

A possible mapping between these two model resolutions is shown in Figure 3.2. Here, the low resolution specification is presented above with the corresponding high resolution specification on bottom. In the low resolution specification, the genes D and C represent Decentralization and Cooperativeness respectively. The high resolution specification in this example shows a chromosome representing a team of 2 UAVs. The genes $L1$ and $L2$ represent the Leader values for each UAV while $C1$ and $C2$ represent each UAV's Cooperation Threshold setting. In the high resolution model, the individual 11000111 indicates that one of the UAVs is a leader and that the Cooperation Threshold of each UAV is 0.57 and 1.0 respectively. In the low resolution model the same string indicates a UAV team with a Decentralization of 0.5 and Cooperativeness of 0.67. With this mapping, the total number of UAVs within the team does not matter from the point of view of the low resolution model. Any valid high resolution mapping is also a valid low resolution mapping.

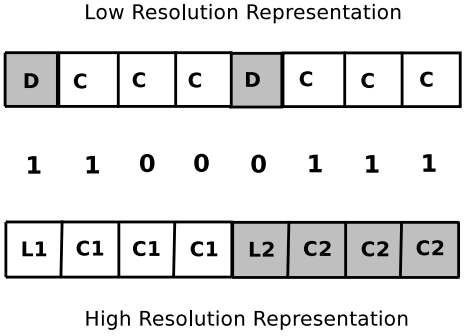


Figure 3.2: The low resolution mapping interprets the bit string as representing 2 genes, Decentralization (D) and Cooperativeness (C) for a UAV team as a whole. The high resolution mapping uses 4 genes to represent the Leader and Cooperation Thresholds for each UAV in a team of 2 UAVs.

3.3 Independent Component Architecture

A full SAMS implementation, designed to operate in mission critical environments, could be based on an independent component architecture in which the individual components of the system could execute in parallel and communicate via message passing (Braude, 2004). This could be especially useful for entities operating in the physical environment that would likely not have the hardware resources to process simulations. Rather, these entities would only need some means of measuring the physical system and sending those measurements to the components running the simulations (admittedly a non-trivial requirement itself).

Figure 3.3 shows the essential components of a SAMS system. Observations from the physical system are passed to an Input Exploration Component (IEC) responsible for conducting input analysis and selecting appropriate distributions for the uncontrollable model factors. Samples from these distributions are then used to create a PME, a copy of which is integrated with each individual to form one CME for each Agent-based Simulation Process (ASP).

The Genetic Algorithm Controller (GAC) is responsible for evolving the population of individuals which are used to form the CMEs. and a number of Agent Simulation Instances (ASPs) to simulate CMEs in parallel. Ideally, each ASP is mapped to one or more CPU cores. If the population used by the GAC is large, or if hardware resources are limited, multiple ASPs can run on a single core. If there is an excess of hardware, a SAMS implementation should be capable of offloading models within the CME to multiple CPU cores.

Outputs from the ASPs take the form of objective fitness values of the individuals that have been averaged across all of the replications specified by the CME. These are passed to the GAC which then assigns the fitness values to the simulated individuals before continuing execution of the GA. Ideally, it should be possible for a user to interact with the GAC in real time to examine the individuals generated and possibly seed new configurations to the population.

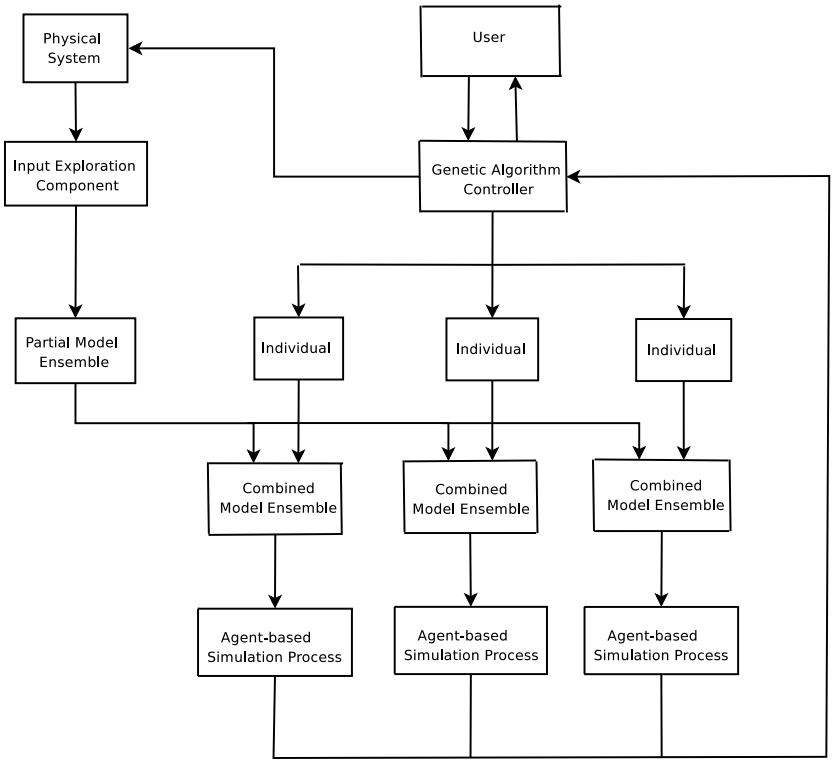


Figure 3.3: SAMS Components

3.4 System Execution

The sequence of operations that occur within the main execution loop of a SAMS implementation are shown in a UML sequence diagram in Figure 3.4. Active components and

processes, displayed as boxes across the top of the diagram include the Genetic Algorithm Controller, Agent-based Simulation Processes (of which several run simultaneously), an Input Exploration Component and the Physical System. An initial message is passed from the GAC to the IEC requesting a new Partial Model Ensemble. The IEC takes observations from the physical system and uses them to produce uncontrollable factor settings which are incorporated into the PME. The PME is then passed to the GAC which uses them to create a CME for each ASP.

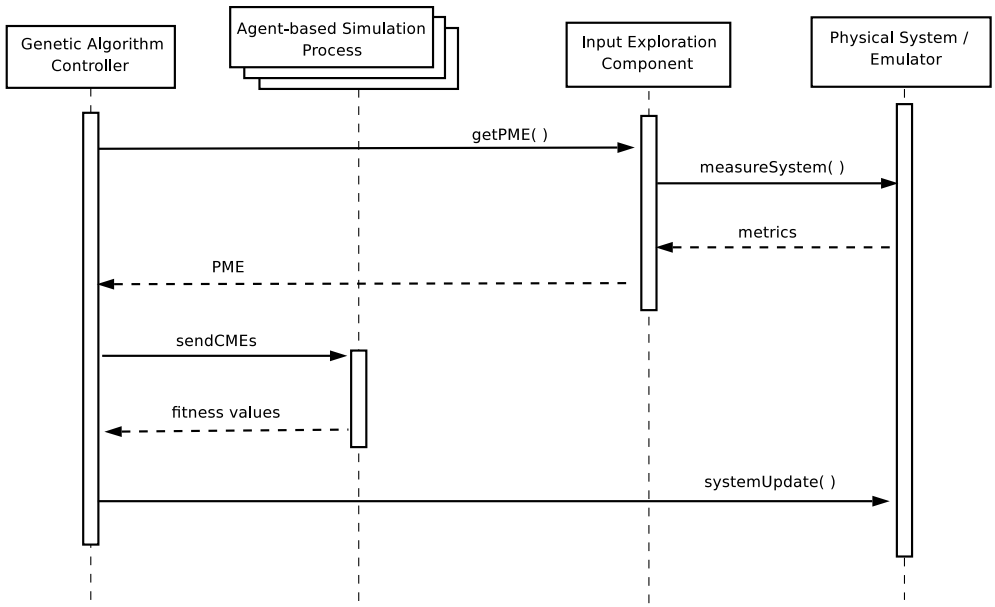


Figure 3.4: The main loop of SAMS.

When created, the CMEs are passed to the ASPs and simulated in parallel. Simulation results are examined on a per CME basis so that a fitness can be assigned to a given individual based on its performance against the environmental settings specified in the PME. After fitness values have been assigned to individuals, the GAC evolves the population. Individuals with higher fitness are given preference for producing offspring which are created

using genetic operators of crossover and mutation. Due to the need to maintain a naturally parallel structure for the algorithm, large numbers of children should be generated before selecting survivors.

In addition to evolving the existing population, the GAC also selects the most fit individual of each generation to update the physical system's configuration. This continual process of dynamically updating the physical system helps ensure that the physical system is responding correctly to observed changes in the environment. This process continues as long as symbiotic simulation is required for the physical system.

With each iteration of the loop, the physical system must again be observed, and these observations are used to determine the shape parameters of the random variable distributions from which uncontrollable factors are sampled. As the physical system develops, it is expected that the variance of observations from these distributions will lessen, permitting a more exploitative search for improved individuals. As this occurs, SAMS should also be able to switch to higher resolution models for the simulation step since fewer loop iterations should be needed after the initial uncertainty in the system has decreased. In addition, it is anticipated that many of the factors which are initially controllable will tend to become uncontrollable as conditions in the physical system converge to their final state. Because of this, constraints on controllable inputs will tend to become increasingly strict over time and some controllable inputs will also transition into the uncontrollable input set.

Chapter 4

Design and Implementation

A SAMS Parallel Application was developed in order to examine the effectiveness of the SAMS methodology. The application is capable of spawning multiple ASPs with which to simulate plausible models, while the GAC and system emulator execute in their own CPU process. The application was written in C, providing exceptional performance, and offers potential for scalability as inter process communication only occurs during the transmission of CMEs and fitness values for individuals.

4.1 Hardware Design and Implementation

Each run of the parallel application requires several thousand model replications, making experimentation on a single CPU impractical. As an alternative, experiments with the application were performed on an SGI Altix 350 Shared Memory Supercomputer operated by the Alabama Supercomputer Authority (ASA)(Authority, 2007). This system has a total of 144 CPU Cores, although the job queuing system typically allocates only a fraction of these to the user. While shared memory is not a requirement of the application, the Altix is capable of running applications that use MPI (Authority, 2005), and was therefore chosen over the Cray XD1 Supercomputer, also available at the ASA.

4.2 Software Design and Implementation

The SAMS Parallel Application including the agent-based simulation with which it operates was developed using functional decomposition and is integrated into a single executable. The application was developed and compiled using Suse Linux GCC 3.3.3 on the Altix. It was intended to be portable to other platforms and only has dependencies with MPI and the Standard C libraries. MPI allows the user to run the application with a variable number of processes. In the current version, a separate ASP is allocated to all but one of the processes which is reserved for the GAC.

In addition to the parallel application, a stand-alone version of the UAV simulation was developed for testing. This version includes a rudimentary 2D graphical display that makes use of OpenGL and the Simple DirectMedia Layer for rendering in the X-Window system. The purpose of this display was to allow a simple visual examination of UAV movement behavior within the simulation.

The SAMS Parallel Application is a work in progress and does not yet include all of the features of a true SAMS system. For the research presented in this thesis, the system included a fully functional GAC and ASPs. However, the ability to measure a physical system or emulator has not yet been implemented. Thus, as will be described in Section 5.5, certain assumptions were made during experimentation.

4.2.1 Random Variate Generation

Rather than make use of the Standard C Library rand() function, a random number generator was implemented according to a specification provided in (Press, Teukolsky, Vetterling, & Flannery, 2007). Implementing a custom generator allows for the incorporation

of a more reliable generator than that provided by the Standard Library. The generator described by (Press et al., 2007) and incorporated into the parallel application uses 64 bit integers and has a period of approximately 3.18×10^{57} which is ample for the purposes of this system. The generator features uniform distributions for double precision floats as well as integers. In addition, the application makes extensive use of normal deviates and the Box-Muller method of normal deviate generation, also described in (Press et al., 2007), was implemented.

4.2.2 System Emulator

Since experimentation with physical UAVs was not possible for this study, a system emulator was incorporated into the Parallel Application. The emulator is an additional simulation which resides in the same CPU process as the GAC. Unlike the simulations in the ASPs, the input factors of the emulator are not modified by the creation of a CME. The emulator is assumed to have the true settings of the physical system. These settings are unknown to the IEC, GAC, and ASPs. However, the IEC does begin with an initial distribution for each uncontrollable factor which includes the true value for the respective setting in the emulator.

4.2.3 Model Execution

In this section, a sample of a stand-alone model's execution within the application is presented along with screen shots from the graphical display of an early build. In Figure 4.1, the first group of UAVs (represented by triangles) can be seen launching from the base located in the center of the map. Dotted lines behind the UAVs indicate their path of movement. Targets are represented by 'x' marks.

As the situation develops, UAVs come into contact with the targets and begin massing their fires on specific targets. Figure 4.2 shows that even with a rudimentary set of movement rules and adaptive capabilities, some surprising examples of coordinated behavior emerge.

Finally, as the scenario winds down with only one target remaining, the UAVs spread out to resume reconnaissance duties. In this model run, Maximum Target Visibility and Communication Range were set to fairly small values. As shown in Figure 4.3, only 1 UAV appears to be aware of the last target. Meanwhile, a large group of UAVs above the base flock together, advancing along a wide front in search of any remaining targets. The UAVs have no global knowledge of the number of remaining targets. These aggregate behavioral changes emerge from local interactions and changing conditions in the learning element of each UAV.

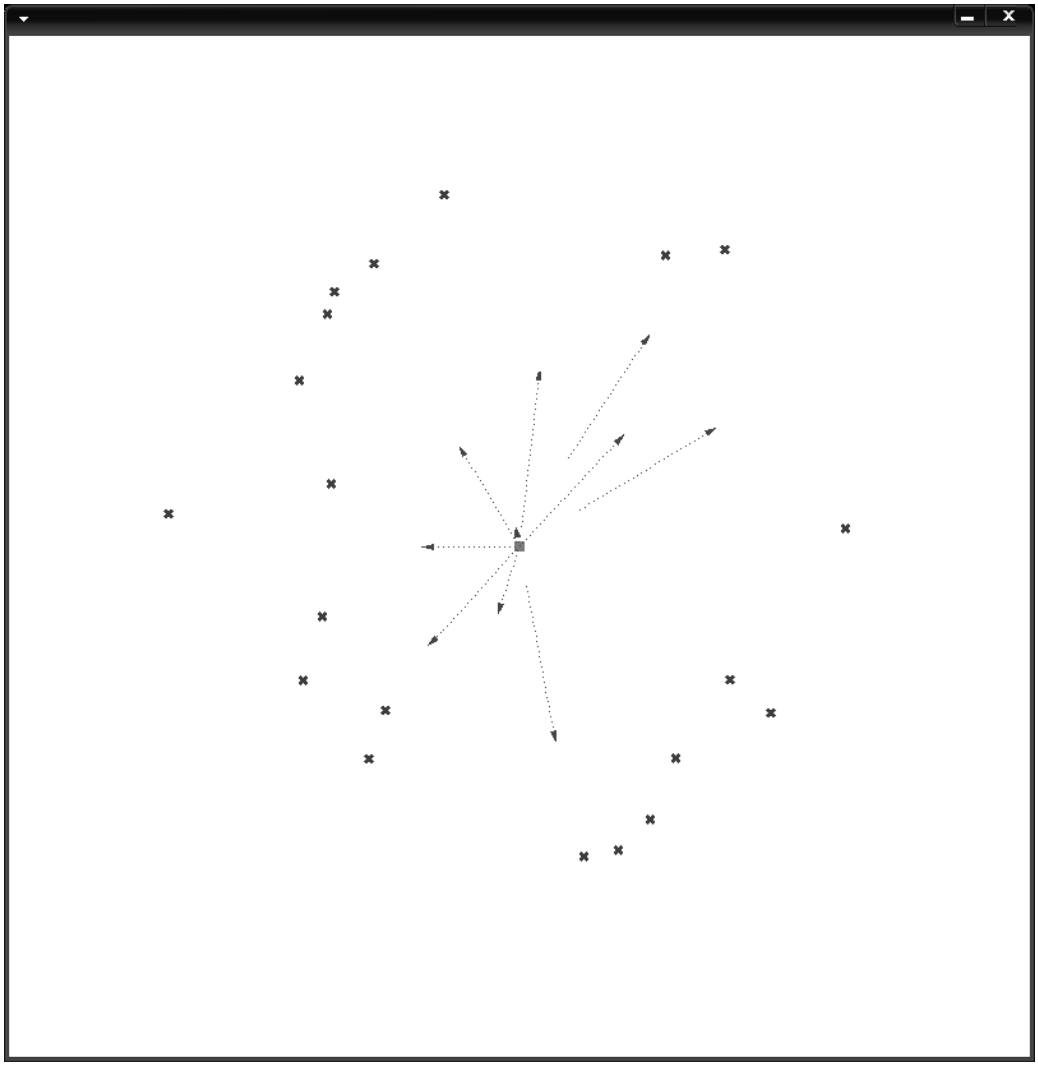


Figure 4.1: Initial launch of UAVs from the base.

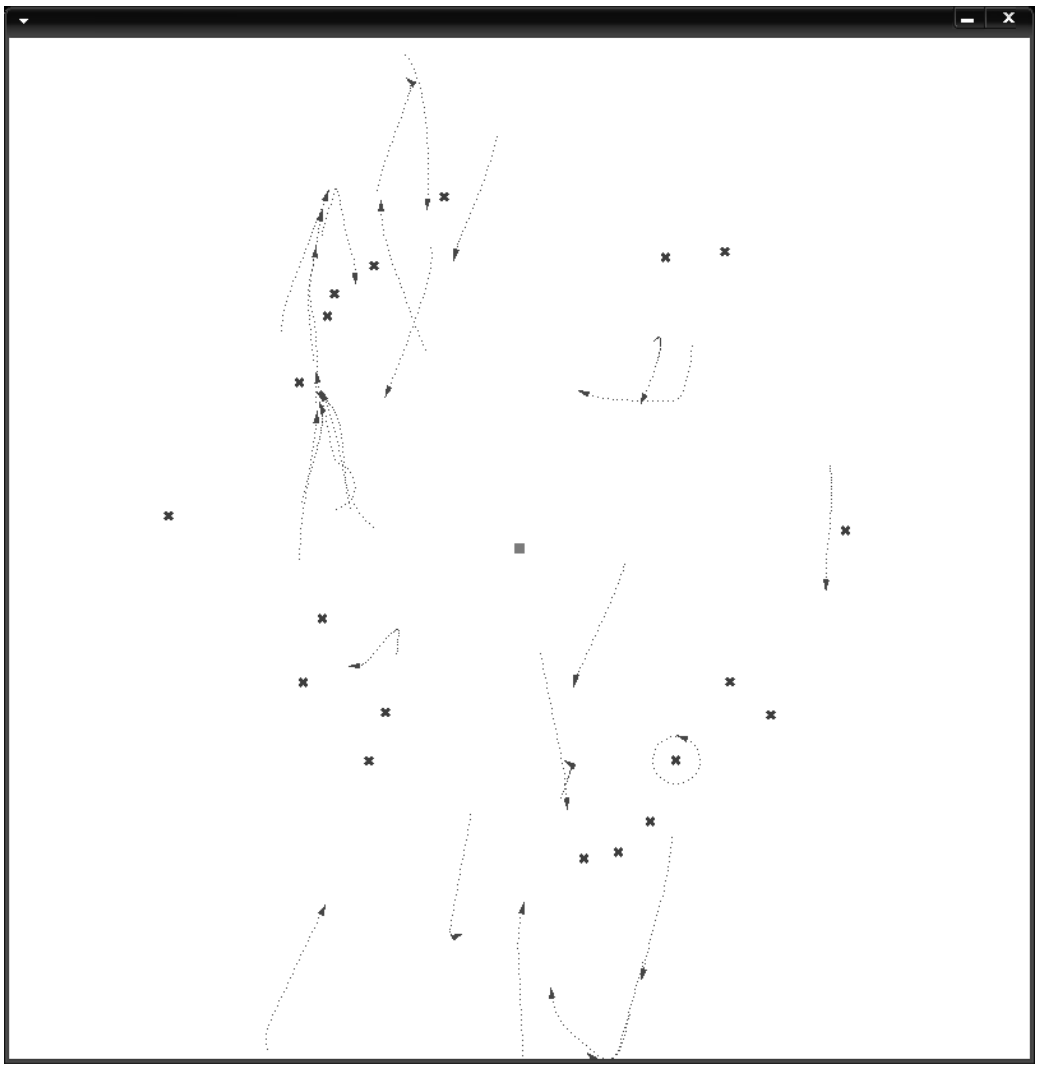


Figure 4.2: UAVs begin cooperative behavior to mass their fires on individual targets.

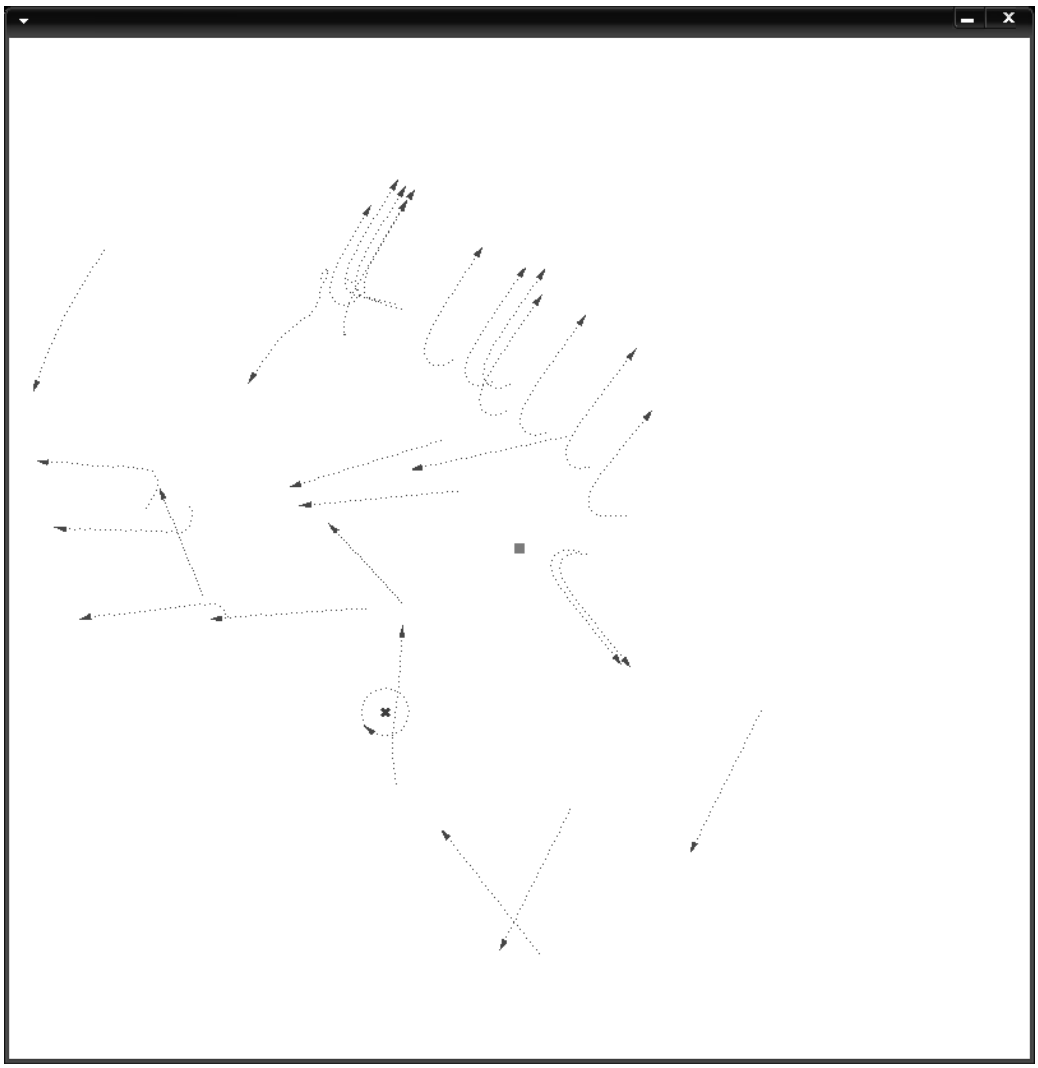


Figure 4.3: With only one target remaining, UAVs begin spreading out to conduct mopping up operations.

Chapter 5

Case Study: A UAV Search and Attack Model

In order to perform an assessment of the potential of the SAMS methodology, a model based on the features of Complex Adaptive Systems was integrated into the parallel application. The field of autonomous UAV cooperation provides a natural environment from which to create such a model. Toward this end, an agent-based model of an autonomous UAV team in a Search and Attack mission was developed. The UAVs in this model interact through local communication only. There is no global system of coordination or prior intelligence of targets. The simulation software for this model was written in the C programming language which provides excellent run time efficiency, ease of use, portability, and compatibility with MPI for parallel programming.

A rule-based approach for UAV movement was implemented for this model, inspired by the work described in (Price, 2006). This set of movement rules, which is based on general knowledge of the problem domain, results in a robust performance element for UAVs capable of both independent and cooperative actions. These rules are implementations of the steering behaviors described in (Reynolds, 1999). Similar implementations of these steering behaviors have been featured in a number of autonomous UAV studies including (Price, 2006) and (Crowther, 2004).

5.1 Design of the Model

A Search and Attack scenario was chosen for the UAV model. As the name suggests, the objective of the UAV team is to effectively sweep an area in order to locate targets,

and eliminate them from play once they are discovered. This scenario was selected because it requires the UAV team to establish a balance between independent and cooperative behavior. UAVs must spread out to efficiently search the area, but must also be capable of massing their attacks in order to effectively prosecute targets. Numbers, communication, and coordination all play a role.

The Search and Attack scenario takes place in a 2-dimensional space of equal dimensions. UAVs begin play from a base located at the center of the map. Targets are distributed randomly across the map and their positions are initially unknown to the UAVs. Once they are launched, the UAVs must find and destroy all of the targets as quickly as possible. Figure 5.1 depicts the opening simulation steps in which UAVs are in the process of launching from the base (represented by a gray hexagon) and sweeping the map. Targets (represented by gray triangles) within the sensor envelope of a UAV have a chance of being discovered while those outside remain hidden.

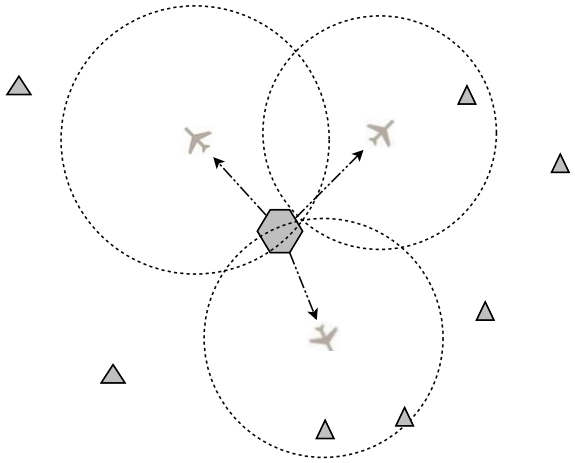


Figure 5.1: The Search and Attack scenario shortly after simulation start. UAVs depart from the centrally located airbase (represented by a hexagon) with uniformly random initial movement vectors. The sensor envelopes of individual UAVs are represented by dotted circles.

Progression of the model is simulated through time-stepped execution, dividing simulation time into a number of equal size increments. During a simulation step, each UAV may act. The results of these actions are updated synchronously and may affect actions performed by other UAVs in the same time step. In distributed simulation, this scheme can complicate the processing of events on different CPUs (Fujimoto, 2000). However, since this model is designed for simulation using a single CPU process, processing of concurrent events does not require separate time steps.

5.1.1 Input Factors

The UAV model has numerous variables that can potentially be parameterized. However, in order to more easily assess the model (and the effectiveness of SAMS as a whole) the number of parameters has been kept small. It was theorized that propagation of target information within the UAV team as well as the command structure of the team itself would play decisive roles in determining success or failure in a given simulation run. These aspects of the model were parameterized as shown in in Table 5.1. The Leader and Cooperation Threshold parameters are controllable factors evolved by the GAC. Maximum Communication Range limits the extent of local communication among UAVs and is intended to loosely model interference from ground clutter or active radio jamming. Maximum Target Visibility models both natural target concealment provided by terrain as well as active camouflaging techniques. Toughness determines the amount of damage a target can sustain from UAVs before being eliminated. These last three factors are initially unknown to the SAMS application. Estimates of their actual values are produced by the IEC.

Input Factor	Controllable?	Range
Leader	Yes	$0, 1$
Cooperation Threshold	Yes	$0 \leq x \leq 1$
Maximum Communication Range	No	$0 \leq x \leq 100$
Maximum Target Visibility	No	$0 < x \leq 100$
Toughness	No	$1 \leq x \leq 10$

Table 5.1: Inputs include both uncontrollable and controllable factors. Uncontrollable factors represent enemy characteristics and environmental conditions while controllable factors represent aspects of the UAV team that may be modified.

5.1.2 Agent Specifications

Each UAV, U, within the model possesses a number of member variables and constants. These are listed in Table 5.2 and represent both the physical state of a UAV as well as its internal cognitive state during a given time step. The physical state includes the UAV's movement characteristics. The cognitive state of a UAV is based on an associated particle in a distributed PSO (discussed in Section 5.3). Each variable or constant has a data type, some of which are vectors. The initial values for some of these variables apply only to simulations which occur when the SAMS application is initiated. Subsequent simulations which are created in ASPs may have different values depending on observations made within the emulator. These variables are indicated with a '†'. Some variables are also directly evolved by the GAC and thus their initial settings are based on the values encoded within an individual's genotype. Many of these variables are described in more detail in the following sections and may be referred to with their respective symbol listed in in Table 5.2.

The other entities within the model, the targets and base, while not strictly agents, are described in this section as well. Targets are immobile and do not actively resist the UAVs. They are specified with two variables listed in Table 5.3. The initial setting for a target's position is a polar coordinate with a normally distributed distance and uniformly

Name	Symbol	Type	Initial Value	Constant?
mass	m	double	2	yes
position	$\mathbf{r}$	double[2]	$(0,0)^{\dagger}$	no
velocity	$\mathbf{v}$	double[2]	$(0,0)^{\dagger}$	no
maximum force	$\lvert\mathbf{F}\rvert_{max}$	double	1.0	yes
maximum speed	$\lvert\mathbf{v}\rvert_{max}$	double	0.5	yes
fuel	f	double	$1.0^{\dagger}$	no
leader flag	l	Boolean	*by genotype*	yes
cooperation threshold	ct	double	*by genotype*	yes
particle position	$\overrightarrow{x}$	double[2]	$U(0,1)^{\dagger}$, $U(10,50)^{\dagger}$	no
particle velocity	$\overrightarrow{v}$	double[2]	$U(-0.25,0.25)^{\dagger}$, $U(-10,10)^{\dagger}$	no
personal best	$\overrightarrow{p}$	double[2]	$\overrightarrow{x}^{\dagger}$	no
local best	$\overrightarrow{pl}$	double[2]	$\overrightarrow{x}^{\dagger}$	no

Table 5.2: The member variables of a UAV affect both its movement physics in 2-D simulation space and the movement of its particle within the space defined by its decision variables. Physics vectors are distinguished by boldface symbols while particle vectors are given with an '$\rightarrow$'. A '†' indicates an initial setting that is dependent on the emulator state.

distributed angle. This setting is discussed in more detail in Section 5.1.5. As previously described in Table 5.1, Toughness is an uncontrollable factor estimated by the IEC and therefore its initial setting is based on observations from the emulator.

Name	Symbol	Type	Initial Value	Constant?
position	r	double[2]	$(N(\mu,\sigma), U(0, 2\pi radians))$	yes
toughness	$tough$	double	$[1,10]^{\dagger}$	no

Table 5.3: The member variables of a target determine its location in the 2D simulation space and whether or not it is currently 'alive'. A '†' indicates an initial setting that is dependent on the emulator state.

The member variables of the base are shown in Table 5.4. The base provides two service queues: fuel and launch, each with one service resource. In the ASP simulations produced by the first generation of CMEs, the fuel queue is empty and the launch queue is full. This reflects the starting condition of the model in which UAVs are considered to be fueled but not yet launched. Later ASP simulations may have different initial settings for these queues

to reflect the servicing of UAVs in the emulator. The launch and fuel queues are discussed in more detail in Section 5.1.8.

Name	Symbol	Type	Initial Value	Constant?
position	r	double[2]	$(50, 50)$	yes
fuel queue	S_{fuel}	list	$empty^{\dagger}$	no
launch queue	S_{launch}	list	$full^{\dagger}$	no

Table 5.4: In addition to its location, the airbase consists of a single-server fuel queue and a single-server launch queue. All UAVs are considered fueled but not yet launched when a simulation begins, hence the initial settings of those queues. A '†' indicates an initial setting that is dependent on the emulator state.

5.1.3 UAV Movement

Each UAV is modeled as a point with mass, m, maximum velocity, $|\mathbf{v}|_{max}$, and maximum thrust, $|\mathbf{F}|_{max}$. Thrust may be applied in any direction in an effort to steer the UAV toward a desired location, but the UAV's mass decreases maneuverability due to inertia. Note that since a UAV is modeled as a point, collisions with other UAVs do not occur. The original physics modeling techniques on which this system is based are described in (Reynolds, 1999). The physics algorithms and most of the movement rules used in the UAV model are 2D implementations of those techniques. Algorithm 2 describes a simple means of truncating a vector so that it does not exceed a given magnitude. This is useful for calculating velocity.

Algorithm 2 : truncate(**a**, b) - Truncates a vector, **a**, by a maximum magnitude, b.

1: **if** $|\mathbf{a}| > b$ **then**
2: $\quad\quad \mathbf{a} \Leftarrow \frac{b\mathbf{a}}{|\mathbf{a}|}$
3: **end if**

For any given time step, the next position of a UAV is calculated by Algorithm 3. It considers inertia caused by changing velocity but does not consider aerodynamics or the effect of gravity. UAVs cannot crash regardless of their actual velocity.

Algorithm 3 : nextPosition(U, $\mathbf{d}$) - Calculate the next position, $\mathbf{r}(t+1)$, of a UAV, U, given a steering direction, $\mathbf{d}$.

1: truncate($\mathbf{d}$, $|\mathbf{F}|_{max}$)
2: $\mathbf{d} \Leftarrow \frac{\mathbf{d}}{m}$
3: $U.\mathbf{v} \Leftarrow U.\mathbf{v} + \mathbf{d}$
4: truncate($U.\mathbf{v}$, $|\mathbf{v}|_{max}$)
5: $U.\mathbf{r} \Leftarrow U.\mathbf{r} + U.\mathbf{v}$

A basic rule for UAV movement is *seek*, which causes a UAV to move toward some specified position on the map. This rule is illustrated in Algorithm 4 and is based on the steering behavior of the same in (Reynolds, 1999). Like Algorithm 3, seek takes a steering direction vector, $\mathbf{d}$, as an argument. In seek, $\mathbf{d}$ is assumed to already have some setting other than (0, 0) and thus seek only modifies the value of $\mathbf{d}$ instead of over writing it. Other movement rules use the same calling scheme so that composite rules of complex behavior can be made from seek as well as some other basic movement rules.

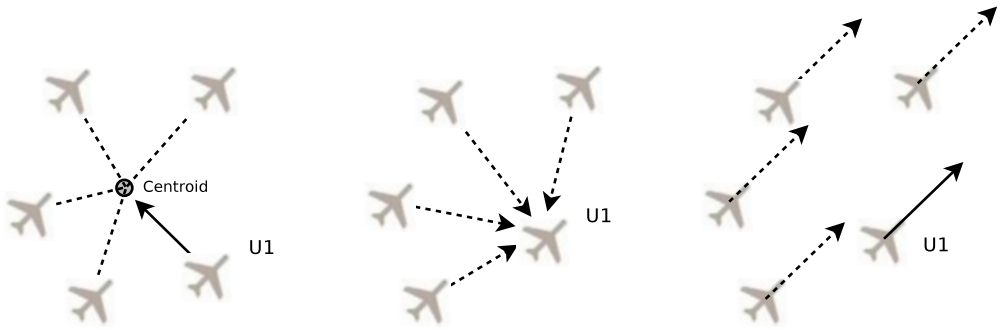
Algorithm 4 : seek(U, $\mathbf{s}$, $\mathbf{d}$) - Modify the steering direction, $\mathbf{d}$ given a UAV, U, and a position toward which the UAV should move $\mathbf{s}$.

1: $\mathbf{s} \Leftarrow \mathbf{s} - U.\mathbf{r}$
2: $\mathbf{s} \Leftarrow \frac{\mathbf{s}U.\mathbf{v}}{|\mathbf{s}|}$
3: $\mathbf{d} \Leftarrow \mathbf{d} + \mathbf{s} - U.\mathbf{v}$

In addition to seek, the model has the following basic movement rules: *offset seek*, *arrival*, *orbit*, *cohesion*, *separation*, and *alignment*. Offset seek allows a UAV to approach a position from a slight offset, similar to the way in which an aircraft might maneuver into a strafing run against a ground target. This is a modification of the *offset pursuit* steering behavior described in (Reynolds, 1999). The arrival rule is identical to seek except that

it causes the UAV to decelerate as it approaches a position. This is useful for simulating a UAV's landing approach to the base for refueling. It is an implementation of a steering behavior of the same name in (Reynolds, 1999). The orbit rule is a modification of the *path following* behavior found in (Reynolds, 1999) that allows a UAV to orbit a position from a desired distance.

The cohesion, separation, and alignment rules are useful for coordinating group behavior among UAVs. Cohesion causes a UAV to move toward the average position, or centroid, of a group of other UAVs. In contrast, separation causes a UAV to move away from other UAVs. Separation is scaled so that the separating force becomes stronger as a UAV gets closer to another UAV. Finally, alignment causes a UAV to align its velocity vector with the velocities of other UAVs. When taken together as a linear combination with weight constants for each rule, the advanced movement rule, *flocking*, occurs. These rules are shown in Figure 5.2. Like the other movement rules listed above, flocking and its constituent rules are based on steering behaviors presented in (Reynolds, 1999).



(a) Cohesion (b) Separation (c) Alignment

Figure 5.2: Flocking is a linear combination of three rules that affect the movement of a given UAV (U1) relative to other members of the flock.

Another advanced movement rule implemented in the model is *leader following*. Like flocking, this rule affects group coordination among UAVs. The flocking rule, however, is completely decentralized, relying on no commands or guidance from a leader. The movement and direction of a flock are therefore emergent, arising from the interaction of movements made by individual flock mates. In leader following, UAVs attempt to steer toward a position immediately behind another UAV that is designated as a leader. The separation rule is also incorporated in leader following to prevent UAVs from clustering too tightly. This is shown in Figure 5.3. The leader is completely unaware of the other UAVs following it and behaves exactly as if it were acting independently. The implementation of leader following in this model is based on the description provided in (Reynolds, 1999).

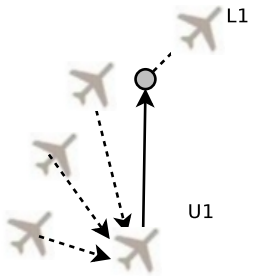


Figure 5.3: In leader following, a cooperating UAV (U1) attempts to move toward a position behind the nearest leader (L1). This movement is somewhat inhibited by separation from other cooperating UAVs.

5.1.4 UAV Communication

UAVs coordinate their behavior through local communication which is limited by the Maximum Communication Range input factor. UAVs receive all messages that are transmitted as long as the distance from the sender is less than this range. All UAVs from

whom a given UAV receives messages are considered to be within its neighborhood, which affects its cooperative behavior as discussed in Section 5.2. A message transmitted from a given UAV includes a list of targets detected by that UAV as well as the best position of the UAV's particle, $\overrightarrow{p}$. UAVs do not retransmit messages that they have received, thus resulting in a single hop wireless network as shown in Figure 5.4.

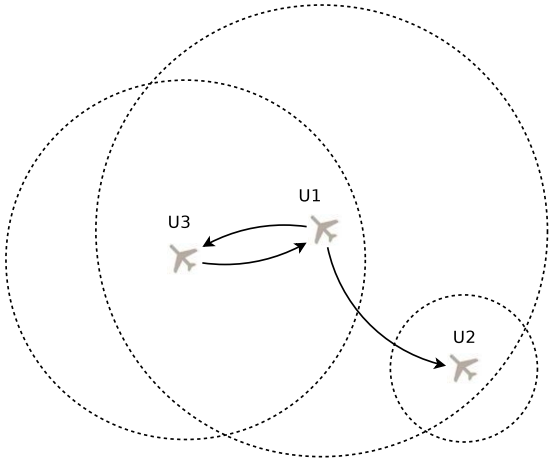


Figure 5.4: Communication range, indicated by dotted lines, affects the ability of UAVs to send messages, indicated by solid lines. In a single hop network, U2 can receive data from U1, but cannot receive data from U3. In a multihop network, U1 can act as a transceiver, passing a message from U3 to U2.

5.1.5 Target Distribution

Targets are placed on the map according to a randomly distributed polar coordinate relative to the base in the map's center. The angle is uniformly distributed from 0 to 2π radians. The distance is normally distributed with the model setting, Mean Target Distance, determining the μ parameter and Target Standard Deviation determining σ. As shown in Figure 5.5, this usually results in targets positioned in a ring encircling the base.

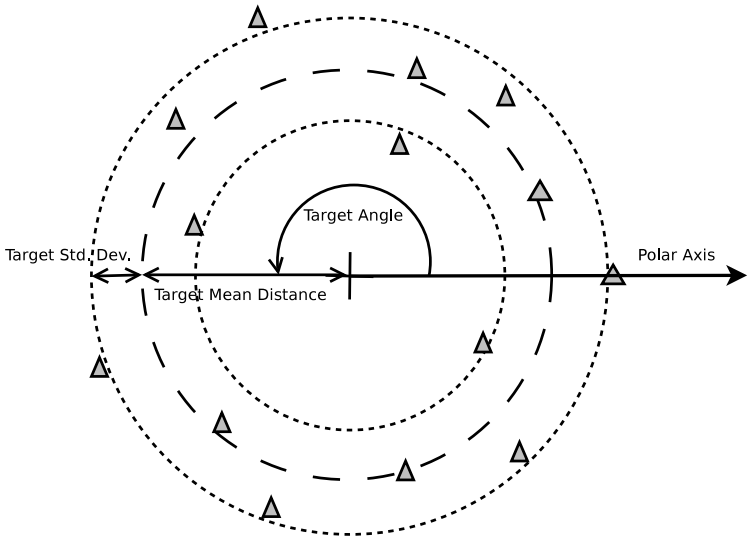


Figure 5.5: Targets are positioned at a polar coordinate relative to a distribution point represented by the crosshair in the center of the diagram. The target angle is distributed uniformly on the interval $[0, 2\pi)$ in radians. The radial coordinate, r, is normally distributed according to the model settings of Mean Target Distance, and Target Standard Deviation.

5.1.6 Target Detection

During each simulation time step, A UAV may discover a target either by making a successful detection attempt, or by receiving the target's location data via a message from another UAV. Detection attempts are possible within a maximum sensor range of a given UAV, defined by the Maximum Target Visibility model setting, which is also an uncontrollable input factor. The probability of a given UAV detecting a given target decreases linearly as distance between the UAV and the target increases. Equation 5.1, below, illustrates this rule.

$$p = 1 - \frac{TargetDistance}{MaximumTargetVisibility} \tag{5.1}$$

Target detection must be reattempted during each simulation time step so it is possible that previously detected targets may revert to an undetected status. A UAV that detects a target will include the target's position data in its communication message during that same time tick. At the end of the time step, however, all targets are cleared from a UAV's memory so it is not possible for a UAV to receive target information from another UAV in one time step and then retransmit that target information in the next time step.

5.1.7 Combat

In order to attack a target in a given time step, the UAV must have either successfully detected the target in the same step, or received the target's position data from another UAV that made a successful detection attempt. UAVs always attack the nearest known target once during each time step regardless of the direction of the target relative to the UAV. However, only known targets whose distance to the UAV does not exceed the Weapon Range model setting may be attacked. Attacks always hit and inflict a normally distributed amount of damage with parameters determined by model settings of Average Weapon Effect and Weapon Effect Standard Deviation. Once a target's Toughness variable has been reduced to 0 or less from UAV attacks, it is considered eliminated and removed from play. Targets do not repopulate the map once removed. In addition, targets do not attack UAVs.

In some studies of autonomous UAV teams, such as (Lua, Altenburg, & Nygard, 2003), the notion of "multi-point attack" has been of interest. This is essentially the idea of flanking an opponent. This concept has been implemented in the UAV model to reflect an

additional benefit of cooperative behavior. When at least 2 UAVs attack the same target in the same time step, and the smallest angle between them relative to the target is at least $frac23\pi$ radians, a multi-point attack bonus is applied. These conditions are illustrated in Figure 5.6. When applied, this bonus doubles the damage inflicted by every UAV attacking the target in that time step.

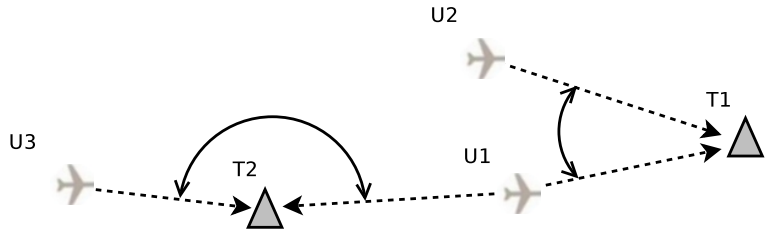


Figure 5.6: A multipoint attack bonus occurs when the shortest angle between any 2 attacking UAVs relative to the target is at least $\frac{2}{3}\pi$ radians. In this case, a bonus occurs if U1 attacks T2 in conjunction with U3, but not if U1 attacks T1 instead.

5.1.8 Supply

A simple model of resupply has been implemented in the UAV model to penalize inefficient UAV behavior. While UAVs are considered to have unlimited ammunition, their fuel supply is finite. This supply is initially set to 1. A certain amount of fuel is consumed each time step that a UAV is in flight according to Equation 5.2, where m is the mass of the UAV, $\mathbf{v}_t$ and $\mathbf{v}_{t-1}$ are the UAV's velocity vectors during the current and previous time steps, and C is a constant equal to the Fuel Consumption model setting.

$$fuel_t = fuel_{t-1} - (m * (|\mathbf{v}_t - \mathbf{v}_{t-1}|) * C) \tag{5.2}$$

Once the fuel supply of a UAV is exhausted, it must return to the base for resupply. (UAVs are assumed to have a spare capacity for the return trip.) Once at the base, it is placed in the base's fuel queue. The service time for the fuel queue is a number of time steps uniformly distributed by the model settings Minimum Fuel Time and Maximum Fuel Time. Once serviced, the UAV is placed in the base's launch queue, also with a uniform distribution and associated model settings for the distribution parameters. Both queues are single server, and while a UAV is at the base it may not send or receive messages from UAVs in flight.

5.2 Design of the UAV Performance Element

The UAV performance element affects only the movement of a UAV as the actions a UAV performs during a time step such as message sending and attacking are purely deterministic. The performance element incorporates the movement rules described in Section 5.1.3 into a decision tree. As shown in Figure 5.7, the various movement rules occupy the leaf nodes within the tree. The decision process begins at the top of the tree. If a UAV's fuel is exhausted, the arrival rule is selected with the base as the desired movement location. Otherwise, the UAV must choose whether or not to act cooperatively or independently during that time step. This decision is based on a Cooperation Threshold, ct, and a current Cooperation value, stored in $\overrightarrow{x}$. The value of ct is fixed for the duration of a simulation run, as it is determined by the genotype of the simulated individual. The Cooperation value is modified by the Learning Element during each time step, discussed in Section 5.3. If Cooperation is not greater than ct, the UAV will choose cooperative behavior. If a leader is within the UAV's neighborhood (defined in Section 5.1.4), leader following will occur.

Otherwise, the UAV will flock with other cooperating UAVs if any are present in its neighborhood. When independent behavior is chosen and known targets are present, the UAV will perform the offset seek rule with the nearest target as its destination. If no known targets are present, the UAV will perform the orbit rule using a Base Distance variable stored in $\overrightarrow{x}$.

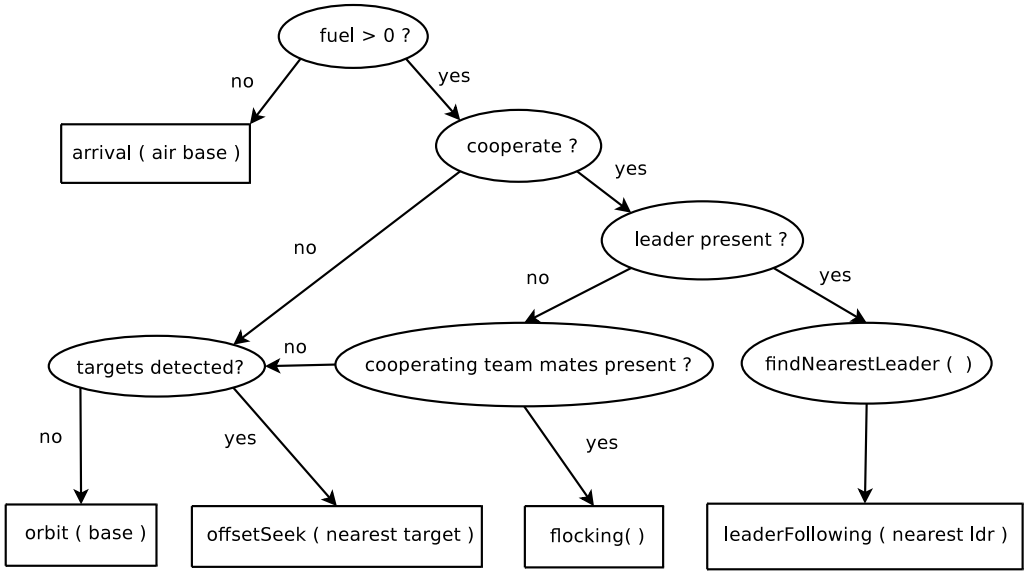


Figure 5.7: UAV movement is a based on a decision tree in which leaf nodes represent steering behaviors and interior nodes are rules implementing considerations such as fuel, other UAVs in the current UAV's neighborhood, known targets, and the PSO decision variable settings of the UAV itself.

The behavior described in the UAV decision tree applies both to leader and non-leader UAVs. However, a leader cannot act as a leader and perform cooperative behavior simultaneously. Therefore, a UAV with the leader flag is not recognized by other UAVs as a leader during any time step in which it chooses to cooperate.

5.3 Design of the UAV Learning Element

The learning element of a UAV has two features. The first feature is a distributed particle swarm that adjusts two decision variables: Cooperation and Base Distance. Each UAV has a single particle associated with it. The position, $\overrightarrow{x}$, and velocity, $\overrightarrow{v}$, of this particle are updated periodically, according to a specified sampling interval. During this update, if other UAVs are within communication range, the UAV will exchange personal best values, $\overrightarrow{p}$, and potentially obtain a new local best value, $\overrightarrow{pl}$. The fitness of a particle's position varies from 0 to 1 and is determined by maximizing the objective function 5.3. In this function, a is the number of attacks performed by the UAV, k is the number of the UAV's kills, and d is the number of target detections propagated by the UAV to itself and its team mates. These variables are counted during the current sampling interval only. They do not accumulate from one interval to the next.

$$max(x) = 1 - \frac{1}{a^{(k+1)} + 2} - \frac{1}{d + 2} \tag{5.3}$$

The second feature of the learning element is a string of 4 bits which models a rudimentary cognitive bias, or "personality" and is generated by the GAC of the parallel application as a part of the UAV team's specification. The first bit, sets the leader flag value, l. This value determines whether or not other UAVs consider the UAV to be a "leader" as described in Section 5.2. The other three bits determine the Cooperation Threshold, ct, which determines how easily the UAV decides to cooperate with other UAVs. Each value of these

three bits represents one of eight values uniformly ranging from 0 to 1. Thus, when the Cooperation decision variable stored in $\overrightarrow{x}$ is less than ct, it will choose to cooperate with other UAVs.

5.4 Model Test Results

In order to assess the suitability of the UAV model for use in the parallel application, a number of experiments were performed to gain insight into the interactions between various controllable and uncontrollable factors. The results of these experiments are listed in Table 5.5. In keeping with the theory that a useful model should force autonomous UAVs to adopt a balance between independent and cooperative behavior, the intent was to examine the interactions between the number of leaders in a team, the Cooperation Thresholds of its members, and Communication Range. Toughness was also included in the experiments as this affects the length of a mission and therefore the amount of time the team has to adapt. Given this intent, a 2^4 factorial experimental design was imposed. Low and high settings for Toughness and Communication Radius were $\{1, 10\}$ and $\{6, 60\}$ respectively. Settings for the number of leaders were $\{0, 4\}$. It was thought that leaders would need lower Cooperation Thresholds than their non-leader counterparts, since only non-cooperating leaders are recognized by other UAVs. Therefore, leaders and non-leaders were given different Cooperation Thresholds for each setting, low or high. Leader settings were given as $\{0.05, 0.1\}$ while non-leaders used settings of $\{0.5, 0.9\}$. The objective for a mission was to minimize the number of time steps needed to eliminate all targets in play. The observations for this metric are listed under the "Average Result" column. The number of time steps required to eliminate all targets was selected as the performance objective because

it subsumes the goals of reconnaissance, combat, and economy of resources. Simulations running longer than 5000 time steps were thrown out and repeated. For each experiment, 30 replicates were obtained in an attempt to improve the odds that the average Mission Complete scores would be normally distributed.

Treatment	Uncontrollable Factors		Controllable Factors		Mission Complete	
No.	Tough.	Comm. Range	Leaders	Coop.	Average Result	Standard Dev.
1	*Low*	*Low*	*Low*	*Low*	462.60	74.28
2	*Low*	*Low*	*Low*	*High*	468.57	89.52
3	*Low*	*Low*	*High*	*Low*	433.20	50.32
4	*Low*	*Low*	*High*	*High*	446.93	79.44
5	*Low*	*High*	*Low*	*Low*	468.30	132.48
6	*Low*	*High*	*Low*	*High*	643.33	225.85
7	*Low*	*High*	*High*	*Low*	473.47	109.47
8	*Low*	*High*	*High*	*High*	556.87	119.6
9	*High*	*Low*	*Low*	*Low*	1826.63	139.56
10	*High*	*Low*	*Low*	*High*	2350.9	293.15
11	*High*	*Low*	*High*	*Low*	1777.24	129.68
12	*High*	*Low*	*High*	*High*	1995.90	219.96
13	*High*	*High*	*Low*	*Low*	1769.18	246.63
14	*High*	*High*	*Low*	*High*	1926.7	453.9
15	*High*	*High*	*High*	*Low*	1885.40	526.5
16	*High*	*High*	*High*	*High*	2475.71	814.61

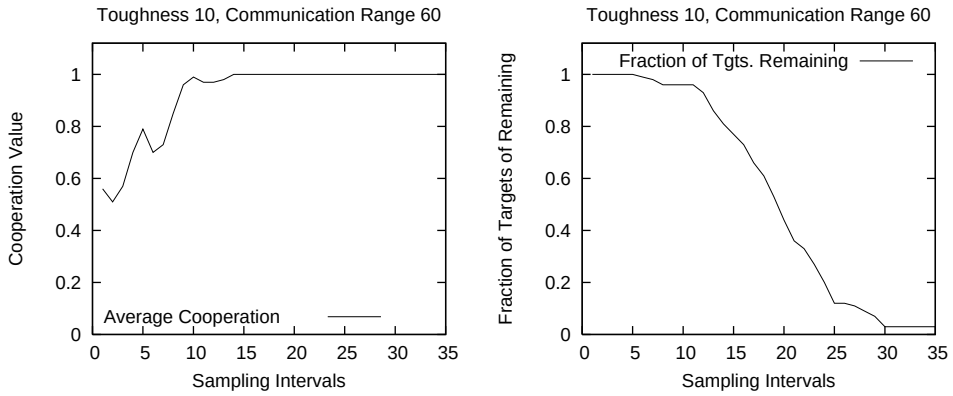
Table 5.5: The initial tests of the UAV model involved 4 model factors examined with high and low values for each factor. Toughness and Communication Range settings were $\{1, 10\}$ and $\{6, 60\}$ respectively. Settings for the number of leaders in a team were $\{0, 4\}$. Cooperation Threshold settings differed depending on whether a UAV was a leader or non-leader. These settings were $\{0.05, 0.1\}$ for leaders and $\{0.5, 0.9\}$ for non-leaders.

As can be seen in the table, certain UAV team configurations appeared to perform better in certain settings. Teams with no leaders and low cooperation values tended to perform better when Communication Range was increased. Conversely, a slight advantage appeared to fall toward teams with leaders when Communication Range was decreased. For

all replicates, Target Max Visibility and Weapon Range were set to 5. Average Weapon Effect and Weapon Effect Standard Deviation were set to 0.05 and 0.03 respectively.

One interesting phenomenon observed in some of the experiments was the behavior of the UAVs' average Cooperation values due to the influence of the distributed PSO. Figure 5.8 illustrates a single replication from one set of experiments, which was not included in the original experimental design, but appears instructive. This case involved a high Communication Range setting of 60 and one leader UAV. As mentioned in 5.3, the distributed PSO evaluates the PSO objective function and updates its decision variables (including Cooperation and Base Distance) once per 50 time steps. This period is referred to as the sampling interval, shown across the bottom of the plot. Thus, at sampling interval 10, 500 time steps have elapsed in the simulation. This plot tracks the change in the average of all Cooperation values within the UAV team across a single model replication. In this case, by about the 10th sampling interval, the distributed PSO had evolved the team away from cooperative behavior. This coincided with a dramatic drop in the target population. Although possibly the effect of random chance, it may be that this change allowed the team to spread out sufficiently to locate and destroy the targets. This case is potentially instructive because it indicates that Communication Range may have a significant interaction with the number of leaders within a team and the Cooperation Thresholds of its members.

The behavior shown previously in Figure 5.8 contrasts with what occurs when Communication Range is set to a low value. Figure 5.9 shows that when Communication Range is set to 6, cooperation is more likely to occur. In this case, the average value of the PSO Cooperation value stayed relatively close to 0.5. Unfortunately, the standard deviation of



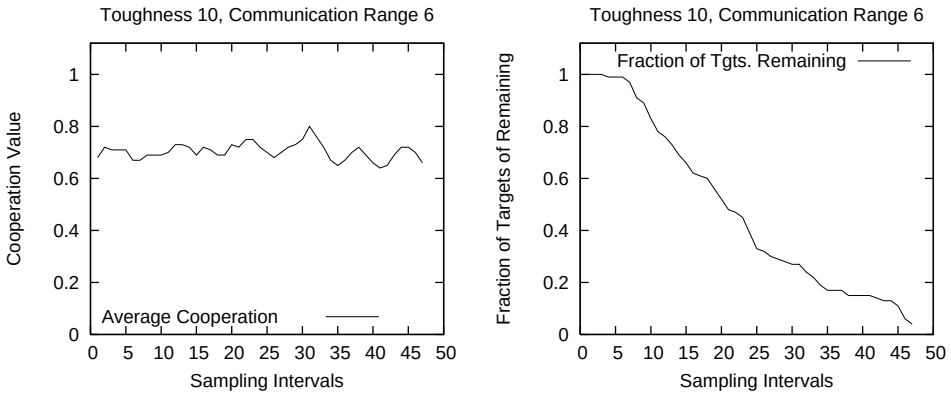
(a) Change in Average Cooperation (b) Change in Target Population

Figure 5.8: During a single model replication, UAVs choose not to cooperate when long communication ranges are in effect and only one leader is present. In this case, the Cooperation Threshold was set to 0.9 for all UAVs except for the leader which had a Cooperation Threshold of 0.1. The increased PSO Cooperation value in 5.8(a) coincides with a sudden drop in Target Population as shown in 5.8(b).

Cooperation values at each sampling interval were not recorded, so it is impossible to know how uniform this decision variable was during this model replication.

5.5 Experiments with the Parallel SAMS Application

The goal of computational experimentation for this thesis was to provide an initial understanding of the potential of SAMS as an S2 methodology. Toward this end, it was hoped that successful experimentation would help to answer a central question: Can the use of symbiotic simulation through ensembles of plausible models improve a physical system's performance?



(a) Change in Average Cooperation (b) Change in Target Population

Figure 5.9: During a single model replication, UAVs are more likely to cooperate when low Communication Ranges settings are in effect. This example involved a Cooperation Threshold of 0.9 for all non-leader UAVs and 1 leader with a Cooperation Threshold of 0.1.

5.5.1 Emulator Objective and Individual Fitness

As described in Section 4.2.2, experimentation with the Parallel SAMS Application involved the use of a system emulator, rather than an actual physical system. This emulator is itself a simulation with the same structural model of autonomous UAV behavior used by the ASPs. It only differs in terms of the values used for its input factors and in that its controllable factor settings are not fixed for the duration of its execution.

Like the model replications examined in 5.4, the performance objective of the Emulator was to minimize the number of time steps required to eliminate all targets. Therefore, in order to synchronize the performance of the simulations executed in the ASPs with that of the emulator, the fitness of an individual was defined to be the average number of time steps required to eliminate all targets in a replication across all replications specified by that individual's CME.

The emulator is an ordinary UAV model simulation which receives controllable factor updates from SAMS. Therefore, it was possible to examine the performance of the Parallel SAMS Application by running an emulator with identical uncontrollable factor settings to those used in the experiments from Table 5.5. Since the emulator and stand-alone models use the same model structure, their performance can be compared as long their uncontrollable factors and fixed model settings are identical.

The theory was that the emulator's system would have an advantage over the system in a stand alone model. The controllable factors of Leader flags and Cooperation Thresholds can be modified by SAMS to the benefit of the emulator, whereas the system in the stand alone model has controllable factor settings that are determined in advance and fixed for the duration of its execution. In order to compare the performance of the emulator with SAMS against a stand-alone model, all fixed model settings such as Weapon Range, Average Weapon Effect, Weapon Effect Standard Deviation, Average Target Distance, and Target Standard Deviation were set to the identical settings used in the experiments described in Section 5.4 as these settings were held constant across all treatments in that group. Furthermore, a subset of the treatments corresponding to one pair of settings of the uncontrollable factors was selected.

5.5.2 GA Design

One intent for SAMS is for the Genetic Algorithm Controller to produce a large degree of exploratory behavior initially, followed by increased exploitative search behavior as the physical system changes. With this in mind, proportional selection was chosen as a parent

selection operator for these experiments. The variation operators used for these experiments included one-point crossover, which obtains a single cut point that determines how the chromosomes will be divided and recombined to produce children as shown in Figure 5.10. Note that one or both of the children shown may actually be created. For these experiments however, two children were created for each crossover operation. The other variation operator, mutation rate, was set to 0.05 so that, on average, 4 bits from each chromosome were flipped. Each individual represented 20 UAVs and thus was 80 bits long. In addition, a non-overlapping survival model was selected so that all children survive while parents automatically die. This type of GA, which balances a relatively low selection pressure with low rates of variation from one-point crossover and low mutation, was thought to be suitable for encouraging exploration of controllable factors early in the physical system's development. As will be shown, the results were mostly favorable.

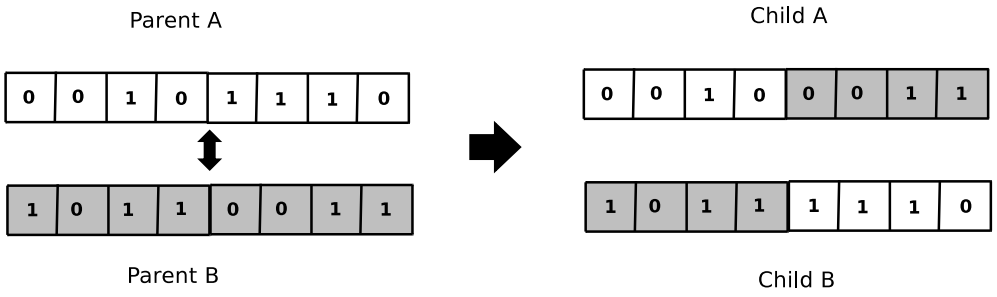


Figure 5.10: In one-point crossover, a single cut point is randomly selected along the length of each parent's chromosome. All genes to the left of the cut point are passed on to the child directly to the parent's right, while everything on the left is exchanged and given to the opposite child.

5.5.3 Emulator Results

The major limitation of these experiments was that the IEC was not yet implemented. Therefore, certain assumptions were made regarding emulator / physical system measurement and PME generation. The work of the IEC was simulated within the experiments. At run-time, the simulated IEC was initialized with uniform random distributions for three uncontrollable factors: Toughness, Communication Range, and Target Max Visibility. Note that Target Max Visibility was held constant across all treatments in Section 5.4 but it is considered one of the uncontrollable factors for these experiments as mentioned in Section 5.1.1. The true values used for these factors within the emulator were Toughness 10, Communication Range 6, and Maximum Target Visibility 5. These settings were identical to those used in Treatments 9, 10, 11, and 12 in 5.4.

The parameterized distributions within the simulated IEC for the uncontrollable factors were arbitrarily selected to be uniform distributions of length 30 about each factor's true setting (truncated by zero as a minimum parameter). This included $U(0, 25)$, for Toughness, $U(0, 20)$ for Maximum Target Visibility, and $U(0, 21)$ for Communication Range. To simulate the effect of increasing information available from the IEC, the length of each distribution was divided by half during each iteration of the GAC's main loop, remaining centered about each true setting. For example, at the second main loop iteration (generation 2 in the GAC and time step 200 in the emulator) the distribution for Communication Range was set to $U(0, 13.5)$. Then, at the third iteration it was set to $U(2.25, 9.75)$ and so on.

At each iteration, a PME of 30 rows and 3 columns (one for each uncontrollable factor) was created. When combined with an individual, each CME therefore had 30 rows and

43 columns (3 columns for the uncontrollable factors and 40 columns representing 2 genes for each of the 20 UAVs in an individual). Each row, representing a single parameterized model, was run for one replication resulting in 30 replications for each CME. A population of 20 individuals mapped to 20 ASPs was used. This resulted in 600 model runs during each iteration of the main loop. The emulator replications included on average, 16 generations. Thus, on average, approximately 9600 model replications were performed by the ASPs for each replication of the emulator. More replications for each row of a CME and a larger population would have been desirable but were not used due to project time constraints.

For the experiment, 30 replications of the SAMS Parallel Application on the Altix Supercomputer were run. As can be seen in Table 5.6, the performance of the emulator enhanced by SAMS within the Parallel Application resulted in a noticeable (though not necessarily statistically significant) improvement in the number of time steps for mission completion compared to the best performing UAV team configuration among the stand-alone model tests. The best stand-alone model configuration was Treatment 11 in this category. Treatment 11 included 4 leaders and Cooperation Thresholds for the leaders and non-leaders of 0.05 and 0.5 respectively.

Experiment	Average Time Steps	Standard Dev.
Emulator with SAMS	1597.5	164.34
Stand-alone model with best average fitness	1777.24	129.68

Table 5.6: The results of the Parallel Application compared to the best performing Stand-alone Model from Section 5.4. The Parallel Application provides an apparent improvement in performance on average, but with higher variance.

Although the average reduced number of time steps required for mission completion obtained by SAMS in this experiment are encouraging, the higher standard deviation in results compared to the stand-alone model are a source of concern, since it was theorized that

SAMS would evolve robust configurations that produce less variance in fitness values. One possible explanation is that the design of the GA used in this experiment does not provide enough selection pressure to consistently steer the system toward an effective configuration. Evidence of this seems to be shown in Figure 5.11. This plot tracks the current fitness of the most fit individual obtained from the GAC over the course of a single replication of the emulator. Each '+' on the fitness line represents an update of the emulator with a new best individual. The plot shows an initial worsening in the fitness of the current best individual. This was common to most of the replications. This may be the result of an initial spurt of exploration which does not complete early enough to allow significant exploitation to occur before the emulator runs its course.

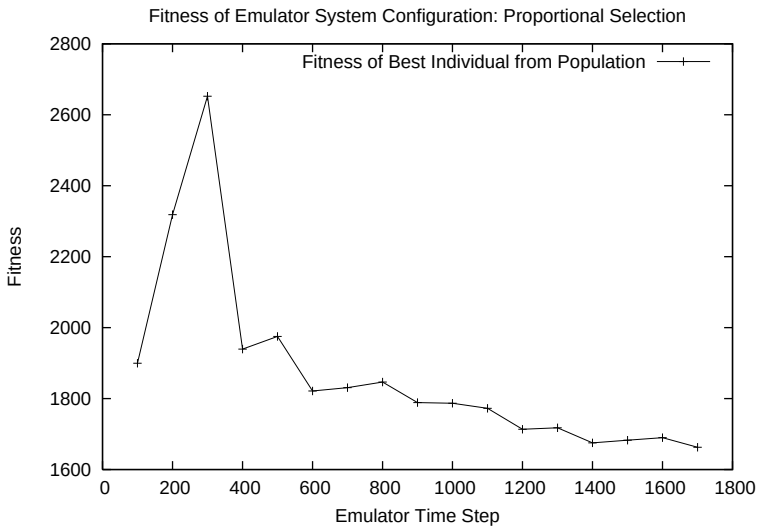


Figure 5.11: During a single replication of the SAMS Parallel Application, the current emulator system configuration is set by the current most fit individual in the GAC's population. With Proportional Selection, an initial worsening of fitness is typically followed by a slow improvement. Every 100 time steps, the emulator's system configuration is updated with the most fit individual.

Another possible explanation for the increased standard deviation in SAMS may be that the number of replications performed for each CME were simply insufficient to properly assess the fitness of the CME's respective individual. This might also explain the initial decrease in fitness in Figure 5.11 as the initial iterations of the main loop involve increased uncertainty in the uncontrollable factors.

While these experiments seem encouraging, they do not by themselves show that the SAMS approach is beneficial for a physical system or emulator. More research and experimentation are needed to explore this question in detail. In particular, experiments with larger numbers of model replications need to be performed. Secondly, the possibilities for modification of Genetic Algorithm used by SAMS have barely been scratched. Much more research needs to be done to identify what design features of a GA (or EA) contribute most to improved performance. Finally, more refinement to the UAV model should be undertaken. The field of Complex Adaptive Systems is rich with research questions and the medium of autonomous UAV teams is an interesting venue for investigation.

Chapter 6

Conclusion

This research has examined the need for dynamic model updating for Symbiotic Simulation of systems involving interacting agents with complex, non-linear behavior. These systems cannot be effectively studied with traditional simulation techniques that rely on valid, authoritative models of the physical system. Instead, techniques such as Multisimulation and Exploratory Analysis, which experiment with an ensemble of plausible models have been developed to deal with these problems.

The Symbiotic Adaptive Multisimulation approach described here used a Hybrid Exploration approach to study an ensemble of plausible models. When parameterized to account for input uncertainty in controllable and uncontrollable factors, SAMS is able to dynamically update a system emulator resulting in an apparent performance benefit. This benefit is realized with the help of a Genetic Algorithm that evolves potential system configurations over the lifetime of the system emulator and which can be used to update the emulator. These updates consist of adaptive strategies that are passed on to the agents operating within the emulator.

This initial study of SAMS as a simulation methodology has shown encouraging results, but has also left many problems unsolved. In particular, further experimentation with larger numbers of model replications is required. Also, experimentation with additional GA designs should be performed to understand the features that make an Evolutionary Algorithm suitable for SAMS. The understanding gained could provide further improvements to the methodology in terms of speed and robustness.

Other significant problems that remain are the incorporation of appropriate Multiresolution Modeling, input analysis for estimating uncontrollable factor distributions, and handling of structural uncertainty. Given these challenges, Symbiotic Adaptive Multisimulation appears to be a rich opportunity for further study.

Bibliography

Authority, A. S. (2005). *Hpc user manual.* Retrieved on Oct 15, 2007 from `http://www.asc.edu/html/man.pdf`.

Authority, A. S. (2007). *Asa home page.* Retrieved on June 30, 2007 from `http://www.asc.edu/index.shtml`.

Ayani, R. (2007). *A generic symbiotic simulation framework (g2sf).* Retrieved on August 19, 2007 from `http://web.it.kth.se/~rassul/research/Symbiotic-Simulation-Framework.html`.

Bankes, S. (1993). Exploratory modeling for policy analysis. *Operations Research*, *41*(3), 435–449.

Bankes, S. (1998). Policy analysis for complex and uncertain systems through computational experiments. In *Aerospace conference, 1998. proceedings.* (pp. 435–449). Snowmass at Aspen, CO: IEEE.

Bankes, S. (2005). *Exploratory modeling.* Retrieved on April 29, 2007 from `http://www.evolvinglogic.com/news/el_encyclopedia.html`.

Barton, R. R. (2004). Designing simulation experiments. In *Proceedings of the 2004 winter simulation conference* (pp. 73–79). Washington, DC: Winter Simulation Conference.

Bigelow, J. H., & Davis, P. K. (1999). *Experiments in multiresolution modeling (mrm).* Santa Monica, CA: RAND.

Bratton, D., & Kennedy, J. (2007). Defining a standard for particle swarm optimization. In *Proceedings of the 2007 ieee swarm intelligence symposium* (pp. 120–127). Honolulu, HI: IEEE.

Braude, E. J. (2004). *Software design: From programming to architecture.* John Wiley & Sons.

Collier, T. C., & Taylor, C. (2004). Self-organization in sensor networks. *Journal of Parallel and Distributed Computing*, *64*(7).

Crowther, B. (2004). Flocking of autonomous unmanned air vehicles. *Aeronautical Journal*, *107*(1068), 111-124.

Davis, P. K. (2000). Exploratory analysis enabled by multiresolution, multiperspective modeling. In *Proceedings of the 2000 winter simulation conference* (pp. 293–302). Orlando, FL: Winter Simulation Conference.

Davis, P. K., & Bigelow, J. H. (2003). *Motivated metamodels: Synthesis of cause-effect reasoning and statistical metamodeling.* Santa Monica, CA: RAND.

DeJong, K. A. (2006). *Evolutionary computation: A unified approach.* Cambridge, MA: MIT Press.

Dreo, J., Petrowski, A., Siarry, P., & Taillard, E. (2006). *Metaheuristics for hard optimization: Methods and case studies.* Berlin, Germany: Springer-Verlag.

Fogel, D. B. (2006). *Evolutionary computation: Toward a new philosophy of machine intelligence* (third edition ed.). Hoboken, NJ: John Wiley & Sons, Inc.

Fujimoto, R. M. (2000). *Parallel and distributed simulation systems.* New York, NY: John Wiley & Sons, Inc.

Fujimoto, R. M., Lunceford, D., Page, E., & Uhrmacher, A. M. (Eds.). (2002). *Grand challenges for modeling and simulation: Dagstuhl report.*

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley Publishing Company, Inc.

Henderson, S. G. (2003). Input model uncertainty: Why do we care and what should we do about it? In *Proceedings of the 2003 winter simulation conference* (p. 90-100). New Orleans, LA: Winter Simulation Conference.

Hereford, J. M. (2006). A distributed particle swarm optimization algorithm for swarm robotic applications. In *Ieee congress on evolutionary computation, 2006* (pp. 1678–1685). Vancouver, Canada: IEEE.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: The University of Michigan Press.

Holland, J. H. (1995). Can there be a unified theory of complex adaptive systems? In *The mind, the brain, and complex adaptive systems* (p. 45-50). Boulder, CO: Westview Press.

Kennedy, J., & Eberhart, R. C. (2001). *Swarm intelligence.* San Francisco, CA: Morgan Kaufmann Publishers.

Law, A. M. (2007). *Simulation modeling & analysis* (fourth edition ed.). Boston, MA: McGraw-Hill.

Low, M. Y. H., Lye, K. W., Lendermann, P., Turner, S. J., Chim, R. T. W., & Leo, S. H. (2005). An agent-based approach for managing symbiotic simulation of semiconductor assembly and test operation. In *Aamas '05: Proceedings of the fourth international joint conference on autonomous agents and multiagent systems* (pp. 85–92). New York, NY, USA: ACM Press.

Lozano, M. G., Kamrani, F., & Moradi, F. (2006). *Symbiotic simulation (s2) based decision support* (Tech. Rep.). Swedish Defence Research Agency (FOI). Retrieved on September 4, 2007 from `www2.foi.se/rapp/foir1935.pdf`.

Lua, C. A., Altenburg, K., & Nygard, K. E. (2003). Synchronized multi-point attack by autonomous reactive vehicles with simple local communication. In *Proceedings of the 2003 ieee swarm intelligence symposium. sis '03* (pp. 95–102). Indianapolis, IN: IEEE.

Namatame, A., & Sasaki, T. (1998). Self-organization of complex adaptive systems as a society of rational agents. *Artificial Life and Robotics*, *2*(4), 189-195.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical recipes: The art of scientific computing* (third edition ed.). New York, NY: Cambridge University Press.

Price, I. C. (2006). *Evolving self-organized behavior for homogeneous and heterogeneous uav or ucav swarms.* Unpublished master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH. Available from Storming Media, `http://www.stormingmedia.us/58/5876/A587644.html`.

Reynolds, C.(1999). *Steering behaviors for autonomous characters.* (Retrieved on October 2, 2007 from `http://www.red3d.com/cwr/papers/1999/gdc99steer.html`)

Richards, M., & Ventura, D. (2003, September). Dynamic sociometry in particle swarm optimization. *Proceedings of the Joint Conference on Information Sciences*, 1557–1560. (Retrieved on September 4, 2007 from `http://reason.cs.uiuc.edu/mdrichar/my_papers/richards.jcis03.pdf`)

Russell, S., & Norvig, P.(2003). *Artificial intelligence: A modern approach.* Upper Saddle River, NJ: Prentice Hall.

Singer, J. L. (1995). Mental processes and brain architecture: Confronting the complex adaptive systems of human thought (an overview). In *The mind, the brain, and complex adaptive systems* (p. 1-9). Boulder, CO: Westview Press.

Yilmaz, L. (2007). Toward next-generation, simulation-based computational tools for conflict and peace studies. *Soc. Sci. Comput. Rev.*, *25*(1), 48–60.

Yilmaz, L., & Ören, T. I. (2004). Exploring agent-supported simulation brokering on the semantic web: Foundations for a dynamic composability approach. In *Proceedings of the 2004 winter simulation conference* (p. 776-733). Washington, DC: Winter Simulation Conference. (Retrieved on September 4, 2007 from `http://www.site.uottawa.ca/~oren/pubs/2004/08-WSC-dynCompos.pdf`)

Yilmaz, L., Ören, T. I., & Ghasem-Aghaee, N.(2006). Simulation-based problem-solving environments for conflict studies. *Simulation & Gaming Journal*, *37*(4), 534–556.