Coalition Formation In Multi-agent Uav Systems

2005

Paul DeJong
University of Central Florida

Coalition Formation in Multi-Agent UAV Systems

by

Paul DeJong
B.A. Calvin College, 1999

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2005

ABSTRACT

Coalitions are collections of agents that join together to solve a common problem that either cannot be solved individually or can be solved more efficiently as a group. Each individual agent has capabilities that can benefit the group when working together as a coalition. Typically, individual capabilities are joined together in an additive way when forming a coalition. This work will introduce a new operator that is used when combining capabilities, and suggest that the behavior of the operator is contextual, depending on the nature of the capability itself. This work considers six different capabilities of Unmanned Air Vehicles (UAV) and determines the nature of the new operator in the context of each capability as coalitions (squadrons) of UAVs are formed. Coalitions are formed using three different search algorithms, both with and without heuristics: Depth-First, Depth-First Iterative Deepening, and Genetic Algorithm (GA). The effectiveness of each algorithm is evaluated. Multi agent-based UAV simulation software was developed and used to test the ideas presented. In addition to coalition formation, the software aims to address additional multi-agent issues such as agent identity, mutability, and communication as applied to UAV systems, in a realistic simulated environment. Social potential fields provide a means of modeling a clustering attractive force at the same time as a collision-avoiding repulsive force, and are used by the simulation to maintain aircraft position relative to other UAVs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Unmanned Aerial Vehicles (UAV) are a popular subject in both the media as well as current research, probably owing to the popularity of aviation and robotics, two fields from which UAVs are derived. Technologically advanced countries have begun to employ actual UAVs in military operations, beginning with Israel in the 1980's, and continuing with the use of the Predator and Global Hawk UAVs by the United States in Bosnia, Afganistan, and Iraq. These systems incorporate varying degrees of autonomy, but they all for the most part require close human supervision and control, often in the form of a human user controlling the aircraft with a joystick. Additionally, most systems are deployed with individual aircraft rather than squadrons, and one human user interacts or controls a single UAV at a time, presumably due to aircraft cost, short supply, and complexity of control.

As costs continue to fall and technological hurdles are overcome, squadrons of UAVs may begin to replace today's individual UAVs. The most obvious benefit would come from redundancy, as the roles of damaged UAVs could be replaced with equally capable backup UAVs, without costing any extra measurable amount of time. The user of squadrons would also allow division of UAV abilities into several types of aircraft. This would limit the mission impact of a downed aircraft on mission success, and allow for the production of smaller aircraft, since the number sensors, armaments, fuel, and other features could be

limited according to aircraft role. Smaller aircraft would be easier to store, harder to shoot down, and allow for the formation of squadrons.

Squadrons of UAVs would present additional difficulties to their operators due to the complexity of managing multiple aircraft. As the number of vehicles in the squadron increases, additional external human pilots would be necessary, and the potential of vehicle collisions would increase. These risks could be mitigated by an increased level of autonomy in the vehicles themselves. Moving important but tedious jobs such as vehicle control and positional proximity to other aircraft from human controllers and into the vehicles themselves would free up human resources to focus on the more important, higher-level tasks which make up mission planning. The ultimate goal of a military operation is the fulfillment of mission objectives, and if human controllers could work at this level rather than at the individual UAV level, many of the complexities of UAV operation could be avoided.

UAV systems can be simulated in many ways. Multi-agent systems were selected for this work since they support the self-interested nature of UAVs. While a mission may require several UAVs to work together, at the same time each UAV should be individually responsible for maintaining its own stable flight, navigation and safety. Multi-agent systems also provide useful models for communication between agents, laws of identity, and mutability or dynamic change, each of which can be applied to the UAV domain.

This work automates the formation of the UAV squadrons in order to complete mission requirements. Shehory et al. [SK95] have done extensive research on agent coalition formation. Coalitions are collections of agents which join together to solve a common problem that either cannot be solved individually or can be solved more efficiently as a group. When a task is present, each available agent is considered for inclusion in the coalition, according to some algorithm. After the task is assigned a coalition of agents

required for success, the task is carried out. Shehory suggests that individual capabilities are joined together in an additive way when forming a coalition. Two agents with an identical capability are therefore twice as beneficial to the coalition together than they are individually.

This work substitutes UAV missions for tasks and squadrons for coalitions. Rather than require a human to assign UAVs to squadrons, coalitions are formed according to one of several algorithms. Three algorithms were used when forming the coalitions: Depth-First, Depth-First Iterative-Deepening, and Genetic Algorithm. The Depth-First and Depth-First Iterative-Deepening algorithms were tested both with and without heuristics which have the potential to considerably reduce search space. Rather than treating all capabilities as additive, this work introduces a new operator which models the result of combining agent capabilities together. The behavior of the operator is context-sensitive, depending on the nature of the capability. Six different capabilities are considered and applied to both UAVs and missions. As one of the algorithms is executed, mission requirements are examined for their required capabilities. Potential coalitions are generated by the algorithm, and a cost function is applied which determines the feasibility of assigning this squadron to the mission. If the mission is selected, the squadron is formed by removing UAVs from the pool of available UAVs, and the process repeats until every required mission has either been assigned a squadron, or it is otherwise determined that the mission cannot be completed with the available resources.

A software simulation was developed in order to test these ideas. The system provides a realistic environment complete with missions and mission planning, formation flight, collision avoidance, and a communication system. An existing 3-D display environment intended for UAV systems developed by Luotsinen [Luo04] was adapted to provide the physical model and display.

Several experiments were performed to determine the relative efficiency of each coalition algorithm. The goal of this work is to determine the feasibility of dynamic squadron formation, evaluate the relative merits of several different approaches for forming squadrons, and establish a new context-sensitive operator which best models the effect of combining like capabilities when forming coalitions.

CHAPTER 2

LITERATURE REVIEW

2.1 Multi-Agent Systems

Multi-agent systems are software systems composed of multiple agents, each of which operates with some degree of autonomy. Multi-agent systems have been a popular area of research, and have been applied to many applications. Multi-agent systems have also been used as a basis for software simulation by Champagne ([Cha03]) and others.

A multi-agent system is typically developed on top of an existing multi-agent framework, rather than developed from scratch. A multi-agent system framework provides some of the basic features required by every multi-agent application, such as communication, independent execution threads or processes, coordination strategies, management schemes, and so on. Several popular multi-agent systems exist. The D'Agents system was introduced at Dartmouth College and supports several languages (TCP, Java, etc) [DAG02]. Aglets is an extension of the Java programming language which supports agents, developed by IBM [AGL02]. JADE is a library written in Java which uses Agent Communication Language (ACL) messaging as the basis of communication [JAD05]. BOND is a multi-agent system built on top of JADE which incorporates a multiplane state machine model to represent the states of the executing agents [BON02].

2.2 Multi-Agent Communication

Several methods of communication have been proposed for multi-agent systems. One of the more commonly used communication systems has been FIPA ACL (Agent Communication Language) [FIP05]. ACL is based on the theory of communicative acts, a branch of psychology. ACL is made up of a number of "performatives," each of which identifies the general intent or context of a message, without defining the message itself; for example, "INFORM," "PROPOSE," "AGREE," and "REFUSE." Performatives do not themselves convey any higher-level meaning, but languages which do convey meaning are built on top of performatives, and are referred to as "content languages." FIPA has specified several content languages, and other organizations have specified their own content languages build on top of ACL.

Another communication language commonly used in multiagent systems is KQML [KQM93]. KQML defines a basic structure for message passing, and like ACL contains a set of performatives. The set of performatives in KQML is larger than in ACL, and unlike ACL, are not derived from communicative acts theory.

2.3 Agent Social Interaction

2.3.1 Coalitions and Negotiation

Agent coalitions are groups of agents which work together for the common good, and negotiation is the primary means of agents coordinating their efforts. Agent negotiation is the basis for social interaction between self-interested agents and provides a means

of coordinating agent activity. Completely cooperative agents have no self-interest and therefore have little need for negotiation, so agent negotiation is the domain of agents which are to some extent self-interested. In the field of UAVs (Unmanned Aerial Vehicles), agents may have a common goal of mission accomplishment, but are also motivated by lower-level self-interested goals such as avoiding collisions and maintaining stable flight.

Many approaches have been proposed for coordinating agent activity. FIPA describes eleven "Interaction Protocols" which detail the exchange of performatives between agents. Deugo, Weiss and Kendall explore more general methods for agent coordination than message passing, based on a set of reusable patterns [DWK01]. Shehory and others have described systems in which agents form coalitions based on group-rationality, the game-theory concept that payoff will be greatest if the individual agents execute tasks as a group [SSJ97]. General principals of agent coordination and a survey of coordination technologies which are often used are discussed by Papadopoulos in [Pap01].

Agent coalitions makes the completion of higher-level goals which are shared between agents possible. Without the ability to form coalitions, single agents may recognize a goal but be unable to complete it due to resource limitations. Even with coalitions, careful work must go into coordinating their efforts in order for the goal to be realized.

Coalitions are explored in more detail in chapter 3.

2.3.2 Contract Net

One of the more well-known mechanisms for agent coordination is the contract net protocol. This is a method of agent coordination initially proposed by Smith [Smi80], and for which a FIPA specification exists [FIP02]. The contract net mimics the real-world

activities of problem recognition, solution solicitation, and contract award. An agent ("initiator") which has a problem presents the problem to a number of participating agents. Agents analyze the problem and either propose a solution, or remove themselves from the process. Based on all proposed solutions, the initiator agent selects the most appropriate solution based on some specific criteria, and subsequently awards the contract to the agent which proposed the best solution. The results of the decision are delivered to all agents involved in the process.

The contract net protocol effectively reflects both self-interested agent behavior as well group rationality. Self-interested behavior is supported since each agent decides whether or not to participate in the negotiation. Group rationality is also supported in some cases, if the problem the negotiation attempts to solve will benefit the group. Contract net negotiation is a well-understood process since it reflects the real-world process of contract bidding and awards. A common example of the real-world contract net process is the awarding of contracts by the military to defense contractors.

In the contract net protocol, one agent acts as initiator and begins the process, and in the end makes the final contract award decision. Each agent therefore fulfills a unique role, one which probably could not be replicated by a randomly-chosen agent. The diversity of agents necessary for a contract net proposal results in a more complex system than one built with completely homogeneous agents because of the variety of agendas involved in the process.

A diagram of the FIPA contract net protocol is shown in figure 2.1.

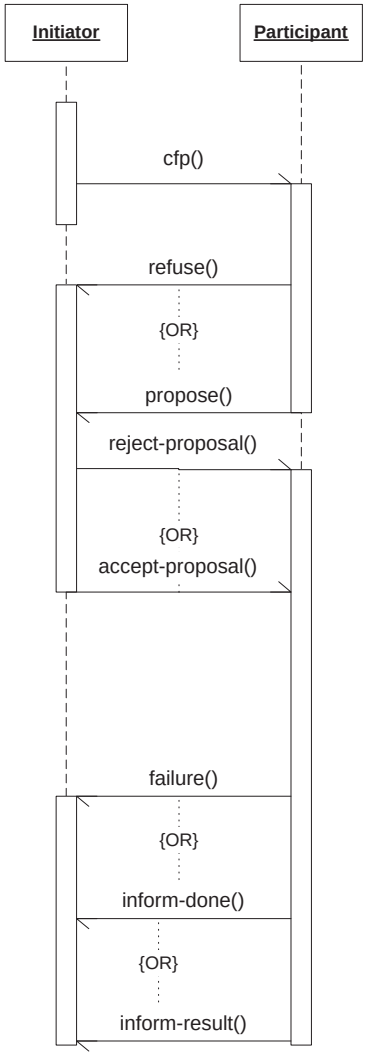

Figure 2.1: FIPA Contract Net Interaction Protocol

2.3.3 Commitments

One final means capable of governing multi-agent interactions are Commitments, proposed by Singh [XS01]. Commitments are used to model obligations that agents hold towards other entities to complete certain operations over time. For example, one agent may be

obligated to provide another agent with a certain amount of data which is not readily available to the first agent. The specifics of the commitment leave no room for doubt; either the first agent has provided the data to the second agent, or it has not, and both agents will agree on the issue.

Commitments are not a protocol themselves, but are instead used to define protocols. They do not hold agents to a rigid set of interactions, but instead define the basis of one agent's relationship to another without specifying how they must interact. Commitments can therefore be used between diverse agents, since only the commitment must be agreed upon, and not an entire recipe for interaction.

Commitments can be defined using the following formalism;

$$C(x, y, G, p) \tag{2.1}$$

where x is the debtor, y is the creditor, G is the context and p is the condition.

Nonmonotonic Commitment Machines can be used to assign states to agents which support commitments, and track those states as progress toward the commitment commences [CS03]. These protocols are less rigid than those which would arise from use of an FSM (Finite State Machine), since commitments do not dictate the content and timing of actual message passing. Paths followed through the commitment machine define protocols which are composed of agent interactions. Commitments are useful for modeling the continual work of an agent towards a goal, its obligations toward other agents, and obligations other agents have toward itself. Commitments recognize the self-interest and autonomy of the agent.

2.4 Agent Mutability

Agent mutability is a process which allows agents to undergo some sort of fundamental change in state. The challenges of engineering such systems have been described by Bölöni et al. [BKB04]. Agents could be compelled to change state for many reasons, either self-triggered or as result of some kind of external stimulus.

Two important kinds of agent mutability include "split" and "merge," both of which change not only the state of an agent, but assign the missing state to another agent in a zero-sum equation. "Split" refers to the division of state into two or more distinct agents. "Merge" refers to the integration of two agent states into a single agent. Splitting and merging of agents are not widespread topics of research in multiagent systems, likely because few parallels can be found in human society, but their implementation in multi-agent systems can solve certain problems more elegantly than is possible when state remains static and stationary in a single agent. Agent splitting and merging are discussed by Bölöni et al. in [BM04].

2.5 Agent Identity

Agents each hold their own identity by which they are known to other agents within a system. An association is made between the agent and the state it contains, since the two are inseparable under normal circumstances. The identity of the agent therefore typically does not change.

However, the proper identity for an agent that undergoes some kind of mutability event (e.g., split or merge) is more ambiguous. For example, when a split occurs, at least one

of the agents must acquire a new identity [BDT04]. Bölöni proposes a naming scheme which encodes such metadata as split number and identity candidacies in the name itself, so that identity can be maintained in a consistent manner during splits and merges [B03]. Such naming is especially useful in cases where an entity (either human or agent) who knows an agent by one identity can still find remnants of the agent's state following a split or merge. By tracing through the metadata contained in the agent names, the disposition of the original agent's state can be extracted, and the present owner of the state can be located. The notation used for agent naming is explained in the following section.

2.5.1 Notation

An extended name is triplet of identity set I, split number S, and candidacy set C.

$$N = \{I, S, C\} \tag{2.2}$$

Identity set I is an ordered set of identities

$$I = \{i_1, ..., i_n\} \tag{2.3}$$

Split number S is a positive integer representing the number of times the agent has been split.

$$S = \{i\} \tag{2.4}$$

Candidacy set C is a set of ordered lists, each ordered list consisting of an identity followed by positive integers. The candidacy represents other identities for which this agent qualifies.

$$C = \{(i_1, n_0, n_1, ...)_1 ... (i_n, m_0, m_1, ...)_o\} \tag{2.5}$$

An example of the naming system resulting from a split is shown in figure 4.6. The agent naming rules used during splits, merges, and other operation are fully explained in [B03].

2.6 YAES

YAES (Yet Another Extensible Simulator) is a framework for developing simulations [YAE05]. YAES is a set of Java libraries providing support for various algorithms, utilities, and graphical output. It makes heavy use of Java interfaces, requiring concrete implementation from the library user. YAES provides an environment in which simulations can be quickly developed, since the basic model for the simulation components are present in the YAES libraries. Some of YAES' current features include two-dimensional abstract maps, Newtonian physical model of two dimensional movement, basic search algorithms (depth-, breadth-, best-first, A*), basic path-planning algorithms, a basic genetic algorithm framework, basic visibility models for wireless sensor networks, basic energy consumption models for wireless sensor networks, and a collection of vehicle control behaviors.

YAES is comprised of a framework section and an application section. The framework contains the libraries necessary for application development, and the application section

contains a collection of applications which were developed using the YAES framework. The applications provide sample usage for many of the features in the framework.

2.7 Search Algorithms

This section will review several different search algorithms which were employed for coalition formation in this work. Search algorithms are used to find one or more states within a search space.

2.7.1 Depth-First (DF)

A Depth-First search algorithm is a tree-based algorithm, with nodes of the tree representing problem states. The primary advantage of this algorithm is its small memory requirements, since unlike breadth-first searches, it does not need to keep knowledge of all nodes of a given depth in memory. Its space complexity is linear in the search depth [Kor95]. A Depth-First search is complete, since it covers the entire search space, as well as optimal, since it will find the best solution possible if one exists.

A diagram of a Depth-First search tree is shown in figure 2.2. The numbers in the nodes represent search order.

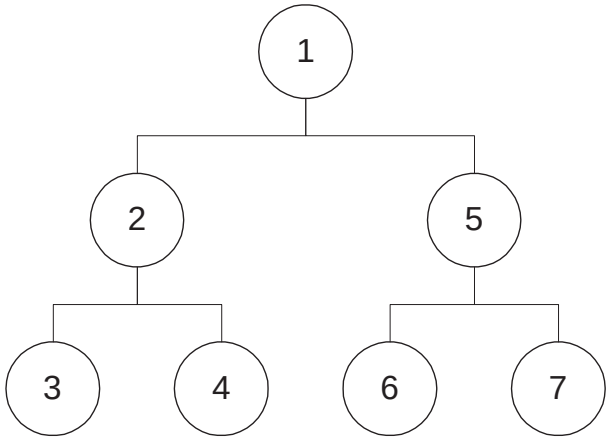

Figure 2.2: Depth-First Search

2.7.2 Depth-First Iterative Deepening (DFID)

The Depth-First Iterative Deepening algorithm is a modification of the Depth-First search in which a maximum search depth is set [ST85]. After the algorithm completes execution to the maximum depth, the maximum depth is incremented and the algorithm is restarted. This makes the algorithm less efficient than Depth-First, since for max depth d, all states $< d$ have already been explored and are being explored again. However, in practice this is not a significant issue, since so many more states are generated at deeper depths, and the algorithm can be initialized with starting depth $d > 1$. Depth-First Iterative Deepening algorithms are useful when a solution is expected at a shallow depth, since the algorithm will be unlikely to "miss" the solution by iterating too deeply, as long as the selected starting depth is picked wisely. This type of algorithm also avoids the problem created by infinite search space, since a traditional Depth-First search will never get beyond the first branch as it infinitely deepens.

A Depth-First Iterative Deepening search is complete, since it covers the entire search space, as well as optimal, since it will find the best solution possible if one exists.

A diagram of a Depth-First Iterative Deepening search tree is shown in figure 2.3. The algorithm sets a maximum depth and performs a search using this subtree. The depth is incremented and the process repeats. The depth will continue to be incremented until the bottom of the search tree is reached, or a solution providing a terminating condition is found.

2.7.3 Genetic Algorithm (GA)

Genetic algorithms were introduced by John Holland at the University of Michigan in the 1970s [Hol75]. Genetic algorithms take an initial population, or chromosome, and iteratively perform crossover (combination of parent chromosomes) and possibly mutation (random modification of the chromosome). The suitability of the solution, or "fitness," is evaluated, meaning the goal is either reached or the process must continue with another iteration, or "generation." Genetic algorithms are stochastic searches which do not guarantee an optimal solution. However, they can be quite efficient when performing large, complex searches.

Figure 2.4 shows an example of a genetic algorithm. A bitmap format is determined for the chromosomes, each of which represent a possible solution to the problem. During the first step, the chromosome population is determined. During the first iteration of the algorithm, this is called the "initial population." On subsequent iterations, this step is used to keep the number of chromosomes constant after the crossover step increases the pool size. Second, the fitness of each chromosome is evaluated using a problem-

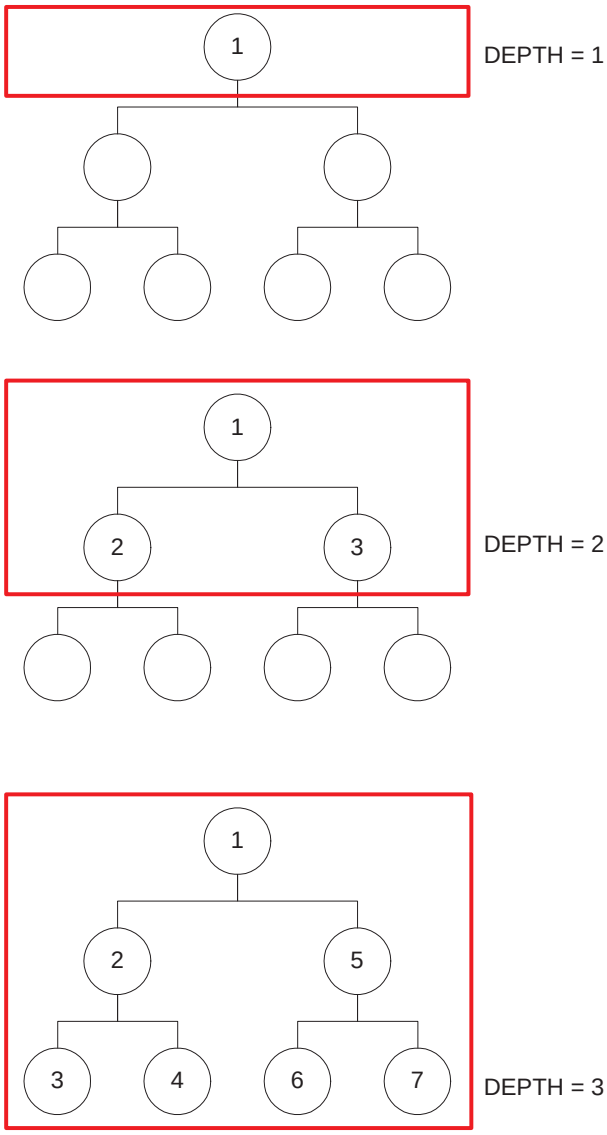

Figure 2.3: Depth-First Iterative Deepening Search

specific fitness function. Third, the two most fit chromosomes as determined by the fitness function are selected for crossover. A crossover point is determined in each chromosome, and the first part of one chromosome is concatenated with the second part of the other chromosome, producing a new chromosome. Fourth, mutation is randomly applied to

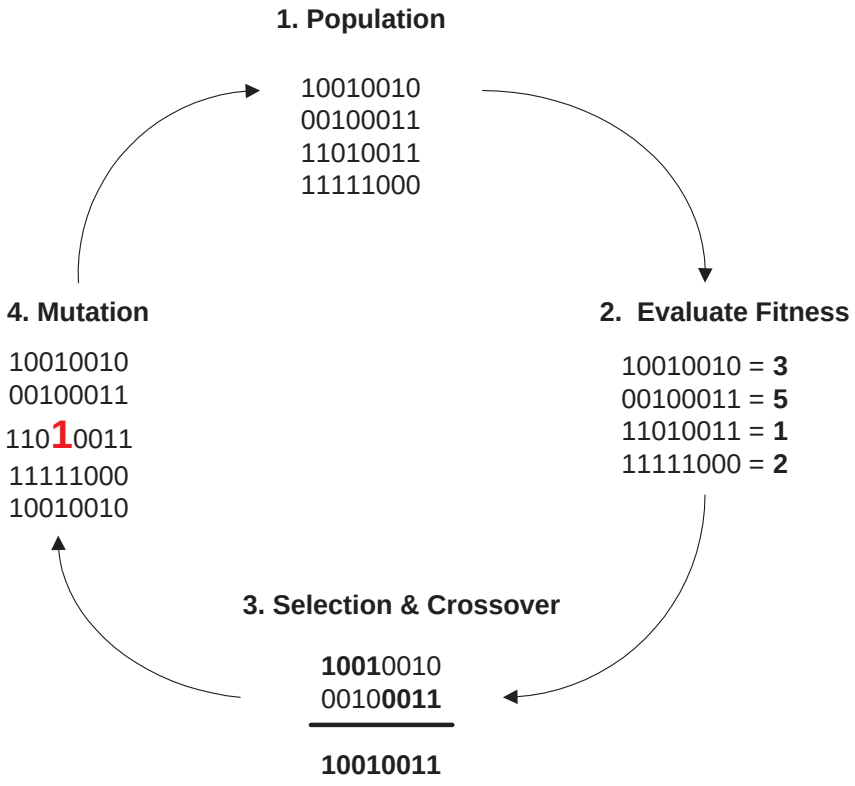

Figure 2.4: Genetic Algorithm

one of the chromosomes. The result of the operations produce a new generation, back to the first step in the process. The process continues until a certain number of generations are produced, and the best (most fit) solution is selected. Many variations of genetic algorithms exist, but this is a fairly typical representation.

2.7.4 Heuristics

Heuristics are knowledge about the specific problem space which are applied to a search algorithm to reduce search space. Heuristics can take many forms. Heuristics are well-

suited to Depth First-type searches, since a heuristic may be able to "cut off" entire branches of the search tree. If the state represented by a node does not provide a solution, and it is known that any other state based on this node will also not provide a solution, then the branch can be removed from the search space.

2.8 Social Potential Fields

Social potential fields are a model for robot group movement based on models of particle interaction in the field of Physics [RW99]. Social potential fields model entities which are both attracted and repelled by other entities as a function of distance. One force attracts entities toward each other, and increases with proximity to other agents. Another force is only present in large amounts when the distance between the entities is small, causing a repulsion force to dominate the attractive force. The net effect is a natural clustering of entities which are kept from tight groupings and collisions by the repulsive force.

Social potential field magnitude can be computed according to the following formula:

$$-\frac{c_1}{r^{\sigma_1}} + \frac{c_2}{r^{\sigma_2}}, \quad c_1, c_2 \geq 0,\ \sigma_1 > \sigma_2 > 0 \tag{2.6}$$

where c_1 and c_2 are independent variables, r is the distance between the two entities, and σ_1 and σ_2 are independent variable exponents to r. The clustering and repulsive behavior is easy to modify by adjusting all values besides r, until the desired behavior is found. The adjustment is somewhat arbitrary and requires trial and error, since the formula does not take into consideration any units of measurement or other factors that tie the formula to the scale used by the entities.

The force exerted on a single entity is calculated by applying the equation to a group of entities with a summation, computing a force between itself and all other entities. The summation is the net force acting on the entity. A positive force indicates attraction (entities are far enough apart to be attracted), and a negative force repulsion (entities are so close that repulsive force is dominating attraction). Social potential fields are a useful way to guarantee collision avoidance, since the force grows infinitely large as distance approaches zero, and enforce absolute positions or relative distances between entities.

Social potential fields have been used in several agent simulations. Gelenbe et al. explored the use of social potential fields along with other navigational strategies and contrasted the applicability of each approach [GHK04]. Several experiments were conducted based on assigning different permutations of navigational strategies to leader and non-leader agents.

Beckhaus used social potential fields to enable navigation through a simulated environment [Bec02]. However, unlike Gelenbe et al., only a single entity, a moving camera, moved though the simulated world. The rest of the force-emitting entities were stationary. This allowed the camera to be "pulled" and "pushed" through the world in an automatic fashion, rather than be operated by remote control.

CHAPTER 3

COALITIONS

3.1 Coalition Formation

The forming of agent coalitions has been discussed by Shehory et al. [SSJ97]. A coalition is a group of agents which seeks to collectively perform tasks in an efficient way. Where a single agent may be unable to complete a task, the collective abilities of a coalition may meet the task requirements. Although the utility of coalitions is clear, efficiently forming coalitions can be difficult. Shehory suggests that the coalitions can be formed by using a modified set-partitioning problem algorithm [SK95], dividing the the available set of all agents into subsets. Shehory introduces the additional criteria that smaller coalitions are preferable to larger coalitions.

Applying the set-partitioning problem to the UAV arena, we are faced with a mission to complete which is comprised of multiple tasks. We have available to us several different kinds of UAVs, each with its own capabilities. We seek the smallest distribution of UAVs into squadrons (coalitions), each assigned to one task, which will enable mission success. The problem is described more formally in the following section.

3.1.1 Notation

The set of all UAV agents N in the system is notated as follows.

$$N = \{A_1, A_2, ..., A_n\} \tag{3.1}$$

Each of the UAVs in the agent set is equipped with a vector of capabilities B. The actual capabilities assigned to UAVs in the simulation software are described in section 3.2.

$$B_i = \langle b_1^i, ..., b_r^i \rangle \tag{3.2}$$

The set of UAVs N is divided into a set of UAV coalitions C.

$$C = \{A_{C_1}, A_{C_2}, ..., A_{C_m}\} \tag{3.3}$$

The UAVs are responsible for carrying out a set of missions which are described as a set of tasks T. These tasks will be divided between the coalitions according to some algorithm. Possible algorithms are described in section 3.3.

$$T = \{t_1, t_2, ..., t_m\} \tag{3.4}$$

In classical coalitions, all capabilities are additive; i.e., two UAVs possessing the same amount of the same capability are twice as beneficial to the coalition as one. However, we propose that the benefit or cost to the coalition depends on the nature of the capability. If one UAV has a capability of stealth, allowing it to pass undetected, then adding another

UAV with the same capability to the coalition will not double the chance of the squadron to pass undetected. Formally, we assume a vector of capability operators O:

$$O = \langle \oplus_1, ..., \oplus_r \rangle$$

where

$$\oplus_i \in \{+, max, min, S\} \tag{3.5}$$

Table 3.1: Coalition Operators

Operator	Description
+	addition
max	maximum
min	minimum
S	stealth

We seek a distribution of UAVs from N into coalitions C which will allow the successful completion of all tasks T. In order for a task to be completed successfully, the capability required for the task at hand from each coalition member UAV is operated on using the new operator together with all other coalition members. The result must equal or exceed the task. If all tasks T are completed successfully, the mission is successful. Formally,

$$\begin{cases} b_1^{C_1} \oplus_1 b_1^{C_2} \oplus_1 ... \oplus_1 b_1^{C_m} > t_1 \\ ... \\ b_n^{C_1} \oplus_n b_n^{C_2} \oplus_n ... \oplus_n b_n^{C_m} > t_n \end{cases}$$

where

$$A_{C_1} \in N, C \subseteq N \tag{3.6}$$

When comparing potential coalitions for assignment to a task, the smallest coalition C which satisfies T is preferred, as this will conserve resources.

The coalition problem is not completely unique. If all tasks T require the additive (+) operator, the problem is reducible to a set packing problem. The set packing problem is a specific set covering problem which requires all elements of a graph to appear in at most one subset of the available elements. A set partitioning problem is similar, but requires every element without exception to be allocated to a subset [GN69]. Optimal solutions to these problems have been researched in other work, for example by Paschos in [Pas97].

Shehory in his research uses the set partitioning problem as the basis of his coalition formation [SK95]. However, the set packing model better enables the additional criteria of minimizing coalition size when applied to the UAV domain. It is actually beneficial to leave UAVs unused in the available UAV pool when missions requirements can be satisfied by smaller coalitions. Set packing is at worst a 2^n complexity problem, meaning each possible combination of UAVs in N must be considered. By allowing context-sensitive capability operators, the complexity of the problem is increased further. The following section considers possible approaches for forming UAV coalitions in an environment where UAVs hold multiple diverse capabilities and multiple missions must be completed.

3.2 Capabilities

A set of capabilities were devised which would use one of the operators described in table 3.1 when combining UAVs in the process of forming coalitions. UAVs are assigned one or more of the capabilities, and missions in the system require one or more of each capability in order to have a chance (not a guarantee) of successful completion.

Table 3.2: UAV Capabilities

Capability	Operator	Description
Firepower	+	additive
Visibility	max	maximum
Speed	min	minimum
Stealth	S	stealth
Number of Bombs	+	additive

- Firepower is the ability to shoot down enemy aircraft. It is possessed by fighter UAVs.
- Visibility is the ability to spot enemy ground positions from a distance.
- Speed is the ability to move quickly between positions.
- Stealth is the ability to pass by enemy aircraft and ground units undetected.
- Number of Bombs is the count of munitions capable of destroying enemy ground installations.

These capabilities are intended to be representative of features aboard real-world UAVs.

3.3 Coalition Formation Algorithms

Several methods of coalition formation were considered, ranging from the simple but inefficient to the quite complicated. The coalitions must be formed in real-time, since UAV capabilities are dynamic. For example, firepower is likely to be depleted as missions

progress, but the amount of depletion cannot be determined ahead of time. Dynamic capabilities will require the dissemination of real-time capability data from each UAV to the system which runs the coalition algorithm. The coalition formation must occur within bounded time to prevent the capability data from being invalidated, and due to hostile environment and limited fuel constraints.

3.3.1 Brute Force

The simplest method for solving the problem is trying all 2^n combinations of UAVs and considering each as a potential coalition for a given task t. The smallest coalition whose capability meets or exceeds the requirement for task t is selected. This method is guaranteed optimal since it will find the smallest possible coalition, but the search time is exponential. The brute force approach may be quite satisfactory for a small set N of UAVs, but may exceed the bounded time requirements for large N. It is the most simple algorithm to implement.

The brute force algorithms selected were Depth-First (DF) and Depth-First Iterative Deepening (DFID). A Depth-First search is a able to cover the entire problem space without exhaustive memory usage, since the search does not need to keep track of multiple nodes. One of the major complaints again Depth-First search, that it will not terminate on an infinite tree [Kor95], does not apply, since the coalition formation will be bounded by the number of available UAVs.

The DFID algorithm was also evaluated as a brute-force method. Although the search tree is not infinite, a DFID algorithm is still useful, since it will consider all solutions at a certain depth (i.e., coalitions of a certain size) before moving on to larger coalitions.

The DFID algorithm may therefore find an optimal solution sooner than the regular DF search.

3.3.2 Brute Force + Pruning

The brute force + pruning method applies a heuristic to coalition formation as a way to reduce search space and therefore minimize search time. The heuristic is based on the context in which the $\oplus$ operator is being applied; i.e., $\oplus_i \in \{+, max, min, S\}$. The heuristic detects states where the algorithm can be short-circuited, either ceasing further consideration of the current branch in the search tree, or halting the search algorithm altogether. For the $min()$ and stealth (S) operations, consideration of the current branch ceases and the algorithm continues. For the $max()$ and additive ($+$) operations, the heuristic cannot be applied.

Coalition size is also used as a heuristic under some circumstances. Coalition size take precedence over the UAV capabilities when forming coalitions. Therefore, if all coalitions containing up to a certain number of UAVs have been considered, and at least one solution has been found, then the best solution can be selected and execution can cease without considering the rest of the coalitions in the search space. The DFID algorithm provides an environment where this heuristic can be applied, while the DF algorithm does not.

For example, if the task t requires a certain speed s, and the current coalition being considered has $min(s) < t$), then we need not consider any coalition supersets containing the present coalition since we have already met the requirement. If a coalition of size n has produced a solution, and all coalitions of size $<= n$ have been considered, then the best coalition can be selected and the search is terminated.

Table 3.3: Heuristic Type

Operator	Heuristic Behavior
min	branch short-circuit
S	branch short-circuit
coalition size	termination

Kraus acknowledges the need for heuristics when forming coalitions if either the number of tasks (i.e., missions) or agents is large, due to the exponential order of brute-force algorithms [KST03].

3.3.3 Stochastic Search

A stochastic search is one in which some form of randomness is present in the search algorithm. The prior algorithms presented for forming coalitions were deterministic; multiple runnings of the same algorithm will produce identical results. A stochastic search may produce different results on each run, and the path to the solution is unlikely to be the same.

A genetic algorithm was selected as the stochastic search algorithm, since it is in wide usage and has the potential to reduce search size. Instead of nodes in a search tree, genetic algorithms use chromosomes to represent possible solutions. Chromosome fitness in a genetic algorithm is evaluated using a fitness function.

The fitness function used in the UAV simulation is somewhat complicated. It is based on the difference between the coalition's capabilities and the task requirement, similar to the

cost function described in section 3.4. Fifty (50) points are given for each mission-required capability met by the coalition, with points subtracted for exceeding the capabilities and therefore starving future missions from capable UAVs. The cost of all failed UAV capabilities are subtracted (this term is represented by the addition operator (+) because the component costs will be negative). Finally, additional points are awarded for forming smaller coalitions, as this conserves resources and helps prevent starvation. Formally,

$$f = \sum_{i_{success}} 50 - min(cc, 40) + \sum_{i_{fail}} cc + max(30, 5 * u) \tag{3.7}$$

where cc is component-cost, $i_{success}$ is the number of capabilities successfully met, i_{fail} is the number of capabilities not met, and u is the number of available UAVs left out of the coalition.

The algorithm was applied with mutation at a rate of 0.125. Mutation inverted a single random UAV value (if the random UAV was present in the potential coalition it was removed, and vice versa). Crossover rate was set to 0.125. The number of chromosomes was set to the pool size of available UAVs, typically between one and twenty-five (1-25).

Genetic algorithms have been used by others to form coalitions. Sen et al. used order-based genetic algorithms as a means to find optimal coalitions [SD00]. Hyodo et al. used genetic algorithms to find optimal coalitions for the purpose of group buying in an online commerce environment [HMI03]. This work differs from those two by the use of the context-sensitive coalition operator ($\oplus$) introduced in this chapter.

3.4 Coalition Valuation

In order for coalitions to be evaluated, a means must be established by which they can be compared. A coalition cost was established for each coalition generated by the coalition algorithms. The cost function generates values only if the coalition is feasible; that is, if the coalition possesses the correct combination of capabilities to meet mission requirements, without falling short of the required capability level for a single capability. The cost function takes the difference between each capability level required by the mission and the capability level possessed by the coalition, and sums these values. Formally,

$$cost = 2 * size + \sum_{i} min(0, t_{i_{req}} - t_{i_{actual}}) \tag{3.8}$$

Lower cost values are preferable, since these are possessed by coalitions which best fit the requirements of the mission without wasting resources. Because scenarios containing multiple missions are a possibility, coalitions with high cost can present starvation situations, since resources which were unnecessarily assigned to early missions may be necessary to complete later missions.

In this system, coalition costs are determined by a single UAV acting as an agent negotiation "Initiator." More complicated scenarios are possible where a variety of cost functions exist in the same system, and the selection of a cost function depends on the agent performing the calculation. Such systems have been explored by Scully et al. [SML04].

CHAPTER 4

SOFTWARE

This chapter discusses the software which was produced in order to provide a realistic environment for UAV coalition formation. It also examines the existing applications and frameworks which were extended and built upon to form the basis of the produced software.

The UAV simulation software described in this chapter provided a framework for testing UAV coalitions, and were included in this work for that purpose. While coalitions can be discussed without any need for software, the simulation provides a more realistic environment for exploring issues related to coalition formation.

4.1 Environment

4.1.1 UAV Simulator

The UAV Simulator used was developed by Luotsinen [Luo04]. It provides a 3-D graphical environment built on the OpenGL library. The simulation provides an ASCII TCP/IP command set of around twenty messages by which clients synchronously provide instructions and receive data. The simulator keeps track of terrain, buildings, targets, all UAVs,

and their respective positional and attitudinal states. Units of measurement sent to and provided by the simulation are based on the OpenGL coordinate system. The simulation provides a description of the simulated terrain to requesting clients by means on the "Range Finder" messages, which returns a grid of height offsets based on the requesting UAV's current altitude. By periodic use of this message, clients are able to construct internal representations of the simulated world, and plan and act based on those representations.

The simulation recognizes four types of UAVs, each with different characteristics and attributes: Normal, Scout, Bomber, and Fighter. All but the Fighter are currently implemented in the simulation. Bomber is the sole UAV type which is capable of destroying building and SAM site targets, which have been previously spotted by UAVs of other types. The simulation is also capable of computing damage to UAVs caused by collisions with obstacles and SAM missiles, and reporting this damage back to the client.

The simulator was originally written to be used with client software also written by Luotsinen, but the TCP/IP ASCII command set is robust enough to support interfacing with external clients. The client software plus the Luotsinen simulator together provided the testbed for coalition and UAV-related experimentation.

4.1.2 YAES

YAES (Yet Another Extensible Simulator) is a framework for developing simulations [YAE05]. YAES was described in more detail in section 2.6.

YAES provided several simulation tools used in the client software, some of which were developed for YAES in the process of writing the client software:

- Directory: Part of the yaes.framework.agent library, the Directory class keeps all agents (UAVs) in a data structure. The class provides methods for adding and removing agents, as well as delivering messages from one agent to another.

- ICommunicatingAgent: This interface provides the methods all YAES agents must implement: receiveMessage() and getName(). AbstractCommunicatingAgent is the YAES implementation of ICommunicatingAgent, which was used as the superclass of all agents in the UAV simulation, including the UAVs as well as a Headquarters agent.

- World: This class provides the context for the YAES simulation. It is used to tie in the application with the rest of the YAES libraries.

- IMessage and GenericMessage: This interface and class provide a message-passing abstraction. The message is filled with data and passed between agents using the Directory.

- Genetic algorithm framework: The YAES genetic algorithm framework provides a collection of interfaces and a control class for implementing basic genetic algorithms. Crossover type and rate, mutation rate, number of chromosomes, and specific fitness functions can all be specified.

- Matlab plotting utilities: The SimulationPlot and MatlabBarGraph classes were used to collect statistics during coalition experimentation and generate visual representations in Matlab format.

4.1.3 IDE

The Eclipse IDE is an Open Source editor spearheaded by IBM but developed by the Open Source community. It supports many programming languages through its many add-on modules, but comes with built-in support for Java. Among its features are configuration management support (e.g., CVS), a generic plug-in framework, and strong symbolic debugging support. All client software was devleoped and debugged using the Eclipse IDE.

4.2 Client Software Architecture

The client software was built on top of the YAES simulation framework using the Java programming language, and interacts with the Luotsinen UAV Simulator through ASCII-based TCP/IP communication. The client software provided the experimental testbed for this work, which explored agent coalition formation, squadron mutability, social potential fields, agent identify, and mission management.

Figure 4.1 shows the general architecture of the system. The two primary components of the system are the simulator and the client software. The simulator was built using the OpenGL 3-D library, which provides the visual display of the simulation. The simulator also provides the physics engine, by constraining the movements of the UAVs within pre-set limits.

The client portion of the software is built on top of the YAES simulation framework. YAES provided useful abstractions for building various aspects of the simulation, such as agent-based communication, algorithms, and graphical output.

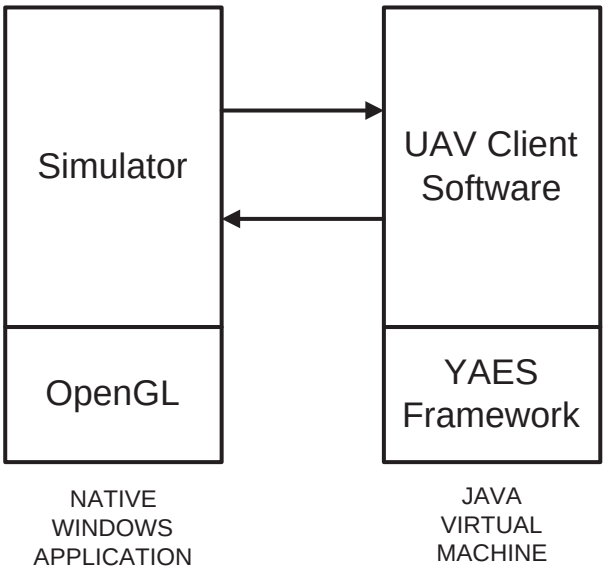

Figure 4.1: System Architecture

The simulator and client portions of the system communicate over TCP/IP using an ASCII-based simulator command set consisting of approximately twenty commands (described in Appendix B). Simulator commands are synchronous, providing an immediate response to the caller. No hard timing exists in the system. Instead, the UAV performs the following control loop actions in sequence, which together govern UAV behavior:

- Process incoming Messages (section 4.2.1)
- Obtain current position from the UAV (section 4.2.2)
- Determine the desired movement or other action (section 4.2.3)
- Send this action to the simulator as a simulator command (section 4.2.4)

4.2.1 Process Incoming Messages

During this phase of the control loop, the UAV examines all incoming messages sent to this aircraft. Message passing is the basis for all communication in the system, and was implemented using the GenericMessage class the YAES framework. Messages are essentially composed of a data structure which holds a sender name, receiver name, and other message-dependent objects. Messages arrive from other UAVs or the Headquarters agent, and can inform the UAV of an action which needs to be taken, indicate the position of other aircraft, request information from the UAV, etc. The full message command set is described in Appendix A.

The UAV has a handler method which processes each possible incoming message. Some message-handling behavior is common to all UAVs, while other behavior is UAV type-specific. Many of the messages trigger a response message which the UAV will send at this time; e.g., an acknowledgment to an instruction. All messages in the UAV message queue are processed before the UAV continues on to the next phase of the control loop.

4.2.2 Obtain Current Position

During this phase of operation, the UAV queries the simulation as to its actual location. The simulation responds synchronously with a three dimensional x,y,z coordinate set. The UAV converts these values to a Position data structure, and replaces the previous "current position" data structure with the new data structure.

4.2.3 Desired Movement or Action

Using the "current position" data structure, the UAV runs through a series of tests to determine if its present position violates any safety constraints. These include social potential field (SPF) force and proximity to mountains or other physical obstructions. If emergency action is required, the UAV determines the desired position to avoid collision. If emergency action is not required, the UAV determines the desired position based on current flight mode (table 4.1).

After the desired location is established, the UAV computes the required heading, altitude, and velocity changes that will move the aircraft toward the desired position.

4.2.4 Send Simulator Command

During this final phase of the control loop, the UAV sends the desired heading, altitude, and velocity changes to the simulator. After sending the command, the UAV agent signals to the Bond framework that it has finished its desired actions for the time being, and thread control is relinquished until the next pass though the control loop.

4.2.5 Class Diagram

A class diagram (figure 4.2) illustrates the Java composition of the client software, excluding the coalition-specific portions. All agents in the system inherit from Yaes' AbstractCommunicatingAgent, including all UAVs as well as the Headquarters agent. UAV

is itself an abstract class, the superclass of each specific type of UAV (e.g, "Normal," "Scout," and "Bomber"). Each UAV contains a Mission Manager, which is responsible for maintaining the squadron mission and the progress of objectives through the mission, and further described in section 4.2.6. UAV instances each communicate directly with the simulator. The logic which makes this communication possible is encapsulated in the SimProxy. This class diagram is incomplete, containing only illustrative classes. It is intended to provide a general description of the architecture.

4.2.6 Mission Manager

The Mission Manager is the software component responsible for obtaining the next mission objective and acting upon it. Mission objectives are selected according to objective priority. If acting within a squadron, each UAV obtains a role within the squadron through contract net negotiation managed by an Initiator UAV. Once a role is assigned, a combination of current objective and role are used by the Mission Manager for determining UAV behavior at any given time. If a UAV is acting alone or is leader of a squadron, the Mission Manager determines the desired UAV position from the current objective, and the heading and altitude changes required to navigate the UAV to this point. If the UAV is acting as a follower within a formation, the Mission Manager determines the desired position for this UAV as an offset from the leader (to the rear and either side of the leader). If the UAV is executing simple follow-the-leader behavior, the Mission Manager controls the following UAV mimicking the behavior of the leader.

The Mission Manager keeps track of the current mode of the UAV. During startup, one UAV has been pre-selected as a leader, and this UAV is assigned Navigation mode ("MODE-NAV"). The remaining UAVs in the squadron are assigned to either formation

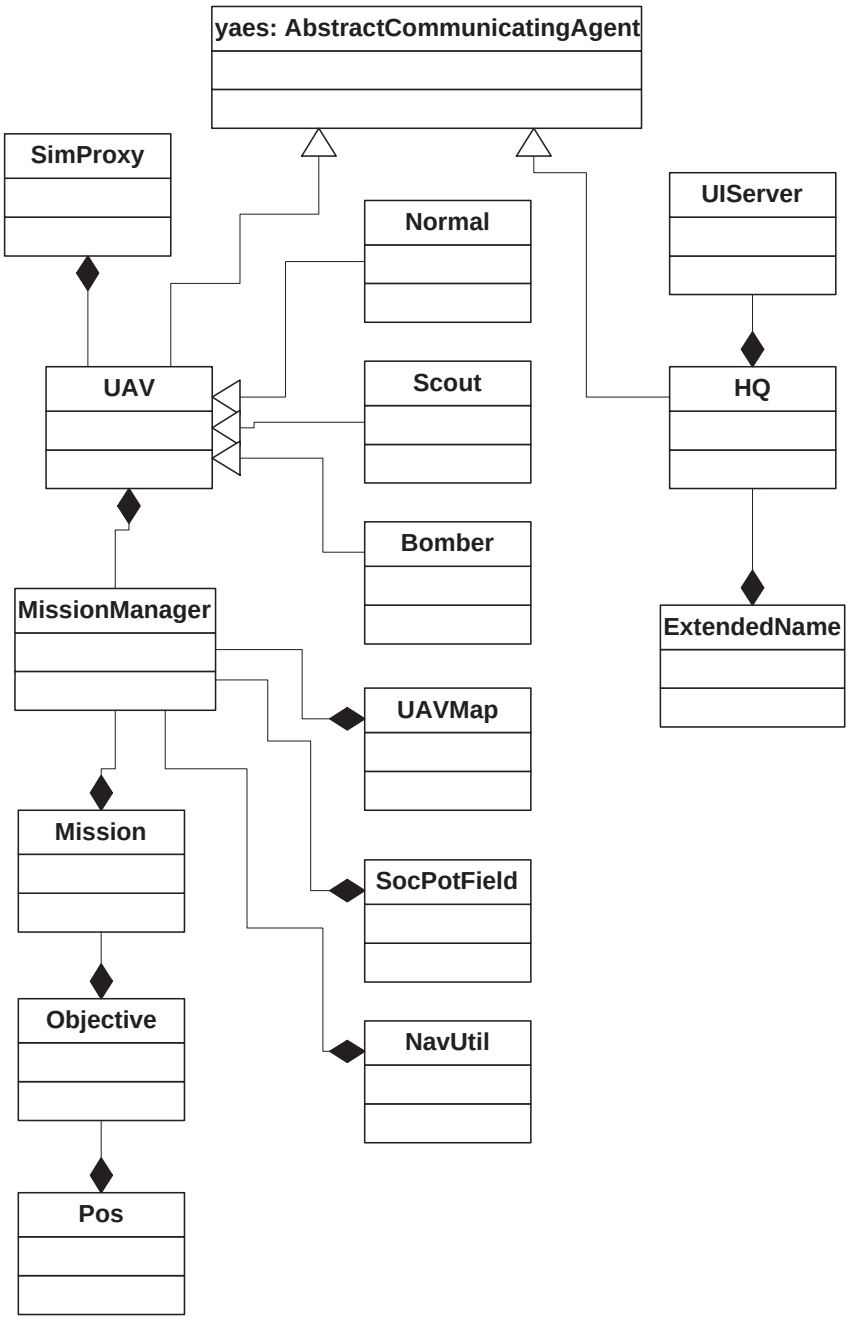

Figure 4.2: Client Software Class Diagram (Excerpts)

flight ("MODE-FORMATION") or follow-the-leader ("MODE-FOLLOW"), as configured by the user.

4.2.7 Headquarters Agent

The Headquarters agent is a special agent in the system which services user commands, sends initial assignments, and keeps track of UAV positions. It is the only agent in the system which is not a UAV. When a user sends a console command, the Headquarters agent intercepts the command, turns the command into appropriate messages from the message library, and sends the messages to the appropriate UAVs. When responses are received from the UAVs, the Headquarters agent interprets the results and formats the output for display to the console. During startup, the Headquarters agent receives the mission assignment and list of UAVs, and issues the messages which deliver the initial mission assignments to the UAVs.

4.3 User Console

The primary means for user communication with the client is the user console. This is a simple shell-like text utility which allows clients to send various commands to the client. The full command set is described in detail in Appendix B. Using the user console, the user is able to dynamically alter squadron missions which are already underway. Various tuning parameters and limits can be altered at runtime. The console can also be used to query client system state, such as the current objective of a UAV or the constituency of a squadron. Commands are sent as ASCII text over a TCP/IP connection to the executing

client software. The connection is initiated by using the telnet command and connecting to port 1202. Synchronous responses are returned to the user by the client software.

4.4 Flight Modes

The UAV state is described by a single mode taken from an enumerated set of flight modes, which can be modeled as an FSM (Finite State Machine) with almost any transition between states being allowed. Flight modes are listed and described in table 4.1.

Transitions between flight modes can be triggered by a variety of events. Some transitions occur as a result of user input from the user console. However, most state changes occur based on simulated stimuli. Close proximity to a mountain will trigger MODE-EMERGENCY. Proximity to a target will trigger MODE-BOMBRUN or MODE-WAIT, depending on the role of the UAV within the squadron.

4.4.1 Navigation

A UAV in Navigation Mode ("MODE-NAV") is in the process of navigating toward a target of some sort. The UAV is either leader of a squadron, or is not part of a squadron at all. While in Navigation Mode, the UAV will periodically compute heading and altitude changes necessary to reach a position specified by the current mission objective, and send these adjustments to the simulation. MODE-NAV ends when the UAV has reached the first boundary of the mission objective. All objectives are composed of at least two modes:

Table 4.1: Flight Modes

Flight Mode	Description
MODE-NAV	Navigate toward a target (leader or solo)
MODE-FOLLOW	Follow a leader by mimicking behavior
MODE-FORMATION	Fix position relative to leader
MODE-BOMBRUN	Approach and descend toward target
MODE-WAIT	Loiter in place awaiting bombrun completion
MODE-SPF	Turn away from UAV to avoid collision
MODE-EMERGENCY	Turn away from obstacle to avoid collision
MODE-LOITER-SINGLEPOINT	Loiter in place
MODE-LOITER-LAWNMOWER	Search area in lawnmower pattern

Navigation toward a target ("MODE-NAV"), followed by execution of the objective itself (e.g, "MODE-LOITER-SINGLEPOINT" or "MODE-BOMBRUN").

4.4.2 Loiter

A loitering UAV stays within a confined area awaiting a future event. Loiter can either take place in a pattern forming the circumference of a circle ("MODE-LOITER-SINGLEPOINT"), or within a rectangular area ("MODE-LOITER-LAWNMOWER") within which the UAV makes progressive adjacent passes until the entire area is covered. Loitering behavior can either take place as part of a mission objective, or can be executed by members of a squadron in a wait state while another UAV carries out the actual mis-

sion objective. For example, a large squadron can navigate toward a bomb target, but a single UAV carries out the bombing while the rest of the squadron loiters.

4.4.3 Bomb Run

After navigating within reach of a bomb target, a single UAV descends to the appropriate altitude and instructs the simulation that a bomb is dropped on the target. The simulation responds with an acknowledgment if the target was destroyed.

Figure 4.3 shows two UAVs loitering while the third performs a bomb run.

4.4.4 Follow-the-Leader

While a leader of a squadron is in navigation mode, the remaining members of the squadron can mimic the behavior of the leader by setting their heading, velocity and attitude equal to that of the leader. Each UAV in the squadron maintains the complete mission, but only the leader navigates directly toward the target by computing necessary heading and altitude changes. By mimicking the behavior of the leader, the UAVs essentially follow the leader to the mission objective.

Figure 4.3: Bomb Run

4.4.5 Formation Flight

Formation flight is a more sophisticated behavior than follow-the-leader, but accomplishes the same goal of navigating a squadron toward a mission objective. The leader assumes the same role as in follow-the-leader, by periodically adjusting heading and altitude based on its present position and that of the target, and feeding these adjustments as commands to

the simulation. Each follower is assigned a positive integer rank (e.g, 1) and side (either left or right), together called a "slot." The follower maintains a certain configurable distance behind the leader for each rank increment, plus one-half this distance perpendicular to (i.e., to left or right of) the aircraft. Thus for any rank which has both left and right slots filled, the leader and the two aircraft in that rank complete the three points of an isosceles triangle.

The desired slot for each aircraft is periodically computed using basic trigonometry (figure 4.4). Once the exact position of the slot has been established, the follower determines the heading required to reach the desired slot, as well as a velocity adjustment if the follower is not within a configurable distance tolerance. The adjustments are sent as commands to the simulation. Because of the reaction time of the followers and the time taken to reach the desired slot, the slot position is never quite reached before a new position is computed for the slot.

Figure 4.5 shows four UAVs flying in formation.

4.4.6 Collision Avoidance

UAVs use the simulator's laser range finder command to scan the altitude of the surrounding terrain, and then internally model this terrain in a map. The internal map is periodically consulted during flight. If the UAV is projected to hit an obstacle within a certain distance if the present flight path is maintained, it enters Emergency Mode (MODE-EMERGENCY). The Emergency Mode causes the UAV to turn 180 degrees and ascend for a small amount of time, and then continue with its previous mode (e.g, navigation, or formation flight).

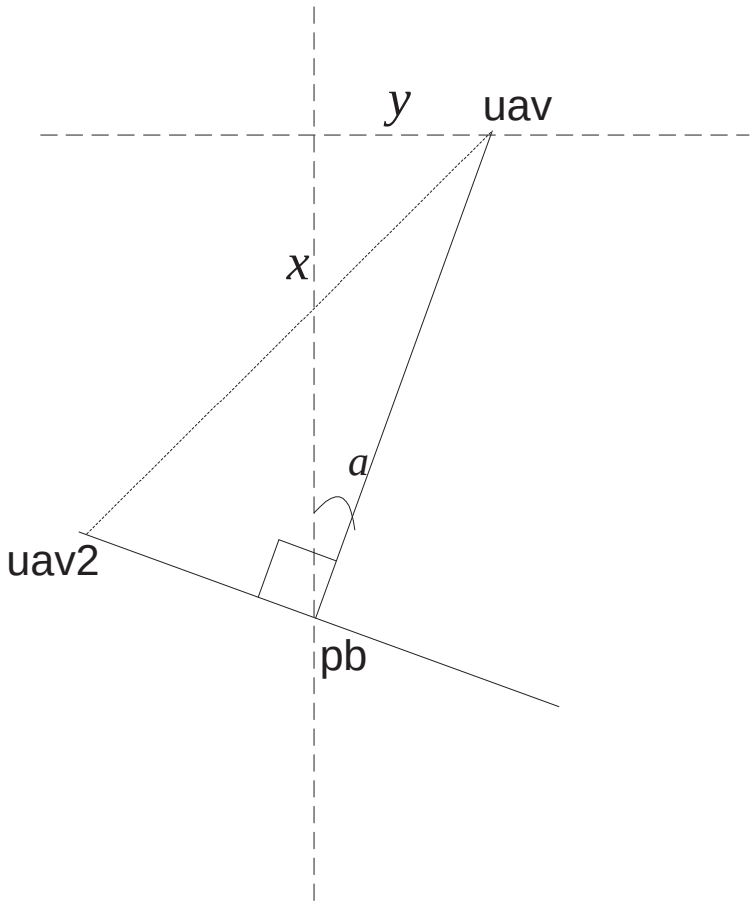

Figure 4.4: Formation Trigonometry

UAVs will also seek to avoid collision with other UAVs. This behavior is modeled as a social potential field (SPF) between itself and other UAVs. The social potential field is modeled according to the formulas described by Reif et al. [RW99]. When the force resulting from the SPF equation exceeds a certain configurable threshold, the UAV will enter "SPF-HOLD." This mode is similar to emergency handling, and causes UAV to turn away from the direction of the force in order to avoid a collision.

Figure 4.5: Formation Flight

In order to implement the SPF behavior, vectors were used so that a direction for the force can be determined in addition to a simple force magnitude. Where the SPF equation calls for a summation of distances between two points, the point pairs were replaced with vectors. Once added together using vector addition, the resultant vector provides a single force magnitude and direction between a given UAV and all other UAVs. The resulting force magnitude is used to determine when an SPF force has reached a certain

threshold where it must be acted upon, and the force direction determines which heading the threatened UAV will assume to avoid collision.

The Reif work calls for two forces working against each other, such that in addition to repulsive forces when entities get too close, an attractive force keeps the UAVs together [RW99]. However, the attractive force diminishes as space between two entities increases. Since the behavior of the UAVs is primarily mission management and not collision avoidance, tolerance thresholds were set above which UAVs will exit mission-handling modes and enter temporary collision avoidance modes. The decreasing SPF force between UAVs which move apart is not compatible with a threshold value, and would cause UAVs to ignore their missions and simply congregate together. Therefore, while always computed as part of the SPF calculation, attractive force is typically ignored by the UAV since it falls below the force threshold required for action.

4.5 Missions

4.5.1 Mapping Model

UAVs periodically execute the simulation laser range finder command, which obtains the altitude of the surrounding terrain. The UAV arranges this information in an internal map which it then refers to for navigation and obstacle avoidance. The internal map is implemented as a multidimensional array of integers, whose dimensions refer to the X and Y Cartesian coordinates of the simulated world, and whose value refers to the Z value at that specific X and Y location. The map is consulted during navigation so that a collision between the UAV and the surrounding terrain does not occur.

An additional data structure keeps track of other UAVs in the simulated world. Each UAV periodically sends out a message containing its position to the other UAVs in the squadron. The other UAVs store this information in a map, and refer to it when computing the social potential field between themselves and other UAVs. This map is also used for locating the squadron leader when formation flight or follow-the-leader behavior is required.

4.5.2 Mission Model

Missions are modeled internally inside every UAV within a squadron, regardless of whether the UAV is itself navigating toward the objective target (leader) or following the leader, possibly in a formation (follower). A mission consists of an ordered set of mission objectives. Each objective has a priority, a type and possibly a subtype, at least one position, and a duration. The type identifies the high-level concept behind the objective (bomb, loiter, reconnaissance, etc). The priority is used when determining the order of objectives within a mission. The positions are points in space which identify targets, or bound reconnaissance or loiter positions. The duration is the maximum time the UAV will perform the current objective before transitioning to the next objective.

In addition to holding objectives, missions also hold an overall mission priority and a breakdown of the types of UAVs in the squadron required to carry out the mission. Missions are initially loaded through an XML file, which is parsed into an internal model. In addition to the initial mission, the user can send missions directly to a squadron, which are then merged with the squadron's existing mission. Alternatively, the user can send a set of missions to all UAVs and let the UAVs work out assignment of the missions using the coalition formation approaches presented in chapter 3.

4.5.3 Mission Execution

A Mission Manager is responsible for opening missions, obtaining objectives, and determining the correct mode of flight for acting on those objectives. Each UAV contains its own Mission Manager. The Mission Manager follows the progress of the UAV though the mission, computing heading and velocity adjustments necessary for reaching or fulfilling the next objective, and supplying these adjustments to the UAV which relays them to the simulation. The Mission Manager uses the internal maps to compute social potential fields, detect close obstacles which threaten the UAV, and changes the current mode as appropriate when emergency handling results from these conditions.

4.5.4 Mission and Squadron Split / Merge / Migrate

The user may send additional missions to a squadron using the user console. Once received, the individual objectives are merged into the current mission, determining the order of the objective by their priority. If the squadron is not executing a mission and therefore does not already have a leader, a leader is determined through contract net negotiation.

Missions can be migrated from one squadron to another using the user console. If the receiving squadron already has a mission, the new objectives are merged into the current mission by mission priority as described above. The sending squadron is left without a mission, causing the leader to be demoted and all UAVs to loiter in place until a mission is assigned.

A split is caused by the separation of a squadron into two distinct squadrons, once again instigated by a user command sent through the user console. Once the constituencies of the squadrons are determined, objectives are assigned to the squadrons according to which squadron is most capable of carrying out the mission. Leaders are elected through contract net negotiation. Each squadron is assigned a new squadron name according to the agent naming technique described by Bölöni [B03] and summarized in section 4.7. The two squadrons then carry out their missions independently. A split is shown in figure 4.6.

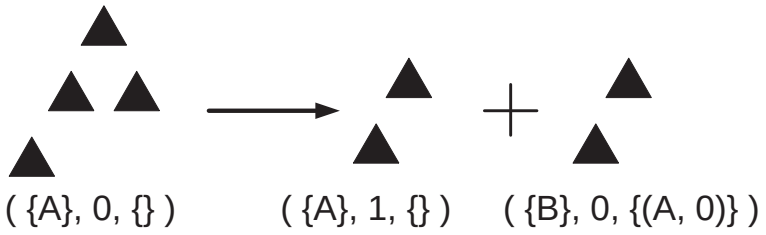

Figure 4.6: A squadron undergoes a split

Two squadrons can also be merged into one squadron. The new squadron is given a new name according to the agent naming rules. The two missions are merged into one mission according to objective priority, and a new leader is selected.

4.6 Agent Negotiation

Agent negotiation is carried out according to the contract net agent negotiation protocol [Smi80]. The two reasons for agent negotiation within the client system are the selection of a squadron leader and selection of a bomber for a bomb run.

The "Initiator" role in the protocol is performed by the first UAV to detect the need for the negotiation, while the "Participant" roles are performed by the remaining UAVs within the squadron. The initiator sends a call-for-proposal message to all UAVs within the squadron. The UAVs each either reject the proposal, and indicate their eligibility by supplying pertinent UAV attributes (max velocity, their UAV type, etc). The initiator then collects the proposals and selects the most capable UAV for that particular mission, and informs all UAVs within the squadron of the result. The initiator itself also acts as a participant, since the initiator is itself a UAV and may be the best leader or bomber candidate.

Figure 4.7 shows a sequence diagram of the leader selection contract net.

4.7 Squadron Naming

Squadron naming conforms to the unique naming strategy proposed by Bölöni [B03]. This naming strategy assures a unique name within a system, and maintains identity even after the named entity has undergone splitting and merging. In the case of the UAV client software, the initial squadron is assigned an identity according to the naming rules. After a split occurs, each of the two new squadrons are assigned a new name based on the original and following the unique agent naming algorithm. The names assigned to

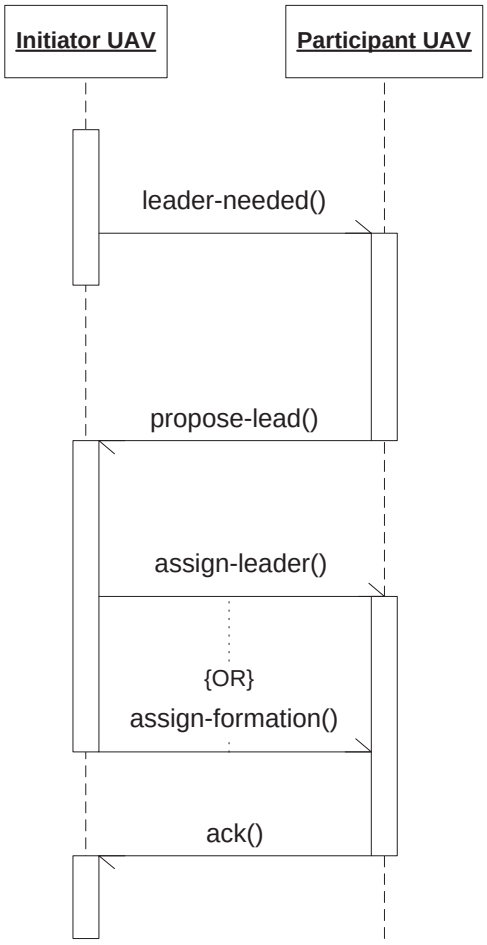

Figure 4.7: Leader selection contract net negotiation

squadrons are used by the user when addressing squadrons for mission assignments and other commands.

4.8 Coalition Formation

This section describes the implementation of the various coalition formation strategies used in the software. Coalitions of UAVs are formed in the system to perform missions. One of three algorithms are used to generate combinations of UAVs. For each combination, a cost is determined. After the algorithm completes execution, the best candidate combination (i.e., the coalition with the lowest cost) is selected and the coalition is formed. The UAVs which form the coalition are removed from the available UAV pool, and the process repeats if additional missions require coalitions.

4.8.1 Coalition Formation Architecture

A class diagram of the coalition formation components of the system is shown in figure 4.8. The CoalitionFormation abstract class provides the interface for the forming coalitions, regardless of algorithm. Its purpose is to form coalitions of UAVs containing missions, and these are represented by the Coalition, UAVRepresentations and MissionRepresentation classes respectively. Three concrete implementations of CoalitionFormation exist. DepthFirstCoalitionFormation forms coalitions using a depth-first approach, either with or without heuristics. It makes use of the DepthFirstAlgorithm to generate combinations. The DFIDCoalitionFormation forms coalition using the Depth-First Iterative Deepening algorithm, making use of the DFIDAlgorithm class to generate combinations. The GACoalitionFormation class forms coalitions using a genetic algorithm. The YAES framework provides basic genetic algorithm support using the GeneticPool class and a collection of interfaces used by GeneticPool. The interfaces are all implemented in problem-specific

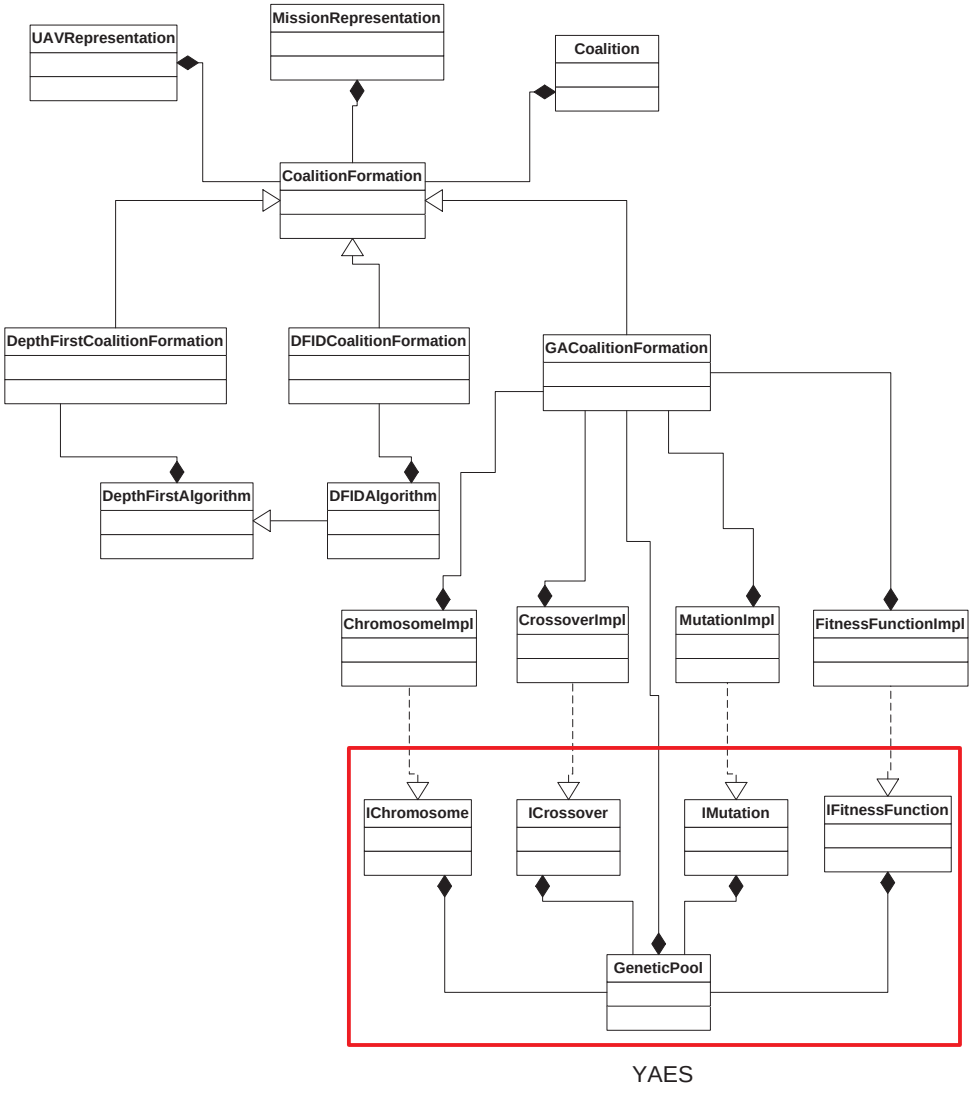

Figure 4.8: Coalition Formation Class Diagram (Excerpts)

manner, providing the specific crossover rates, mutation rates, fitness function and other components required by the genetic algorithm.

4.8.2 Coalition Representation

Combinations of UAVs are generated by the three algorithms in the system, Depth-First, Depth-First Iterative Deepening, and Genetic Algorithm. Each of these algorithms generate combinations which represent possible squadrons. The combinations are simple bitmaps whose indexes each represent a particular UAV drawn from the pool of available UAVs. A zero (0) in an indexed position means the UAV is not present, while a one (1) indicates that the UAV is part of the squadron.

4.8.3 Coalition Cost

Coalition cost is determined for each combination produced by the coalition algorithm. Cost is determined according to the formula described in section 3.4. The cost formula is a "best-fit" formula: It ensures not only that the requirements of a mission are met, but also that the coalition is not "over-qualified" for the mission, since an over-qualified coalition may produce a situation where starvation will occur. Non-judicious assignment of UAVs to squadrons will unnecessarily remove candidate UAVs from the available pool which may be required for future missions.

4.8.4 Test Harness

The Coalition Formation software can be used either as part of the entire simulation or standalone. When used as part of the entire simulation, coalitions are formed when

one or more missions are assigned by the user but no UAVs are assigned to the mission. Each UAV is capable of performing the coalition formation algorithms and notifying other UAVs of their assignments once the results have been determined.

For standalone testing, a test harness was developed to produce experiments. This provided an environment where various strategies and algorithms could be employed during coalition formation without the overhead of running the entire simulation. The experiments are detailed in chapter 5.

CHAPTER 5

EXPERIMENTS AND SIMULATION RESULTS

5.1 Coalition Formation Efficiency Metrics

5.1.1 Description

Five different algorithmic variations were employed to test the coalition formation of UAV squadrons to meet mission requirements. The algorithms are described in more detail in section 3.3. The search space for each of the algorithms is every possible combination of UAVs from the available UAV pool. Each state produced by the search algorithms is therefore a unique candidate coalition, and $2^n - 1$ unique coalition candidates exist for a UAV pool of size n.

The first algorithm, Depth-First, is a brute-force algorithm which exhaustively covers the search space by considering every possible coalition of UAVs. The second algorithm applies heuristics to the Depth-First search, removing entire branches from the search space when it is determined that no possible solution can be derived from a given root coalition. The third algorithm, Depth-First Iterative Deepening, increases the maximum size of the coalition gradually, since smaller coalitions typically result in lower cost solutions. The fourth algorithm, Depth-First Iterative Deepening plus heuristics, employs identical heuristics as the second algorithm to reduce search space, but applies the heuristics to

the Depth-First Iterative-Deepening algorithm. The final algorithm is a Genetic Algorithm with a random single point crossover at rate=.125, mutation at rate=.125, random selection, selecting the best generated coalition after one-hundred fifty generations.

Each algorithm generates a sequence of possible coalitions, and determines a cost for each algorithm. The cost is used as a the basis of comparison between coalitions. Cost is described in section 3.4.

Two metrics were employed in considering coalition formation algorithms: time to first solution and quality of solution over time. Time to first solution is the time taken for the algorithm to produce the first viable result; i.e., the first squadron which has the capability levels necessary to meet mission capability requirements. UAV capabilities are dynamic as time spent in the theater of operations progresses, requiring solutions in real-time, and in the shortest time possible. Extra time spent forming coalitions may mean invalidation of the mission requirement capabilities or UAV capabilities which were used to generate the coalitions. Time to first solution provides the time required to form the first coalition which can meet the minimum mission requirements, regardless of coalition cost.

Quality of solution over time records the cost and elapsed time of progressively improving solutions. Since each algorithm may generate a great number of possible coalitions, this metric is more practical than recording every single solution which is generated by the algorithm. Coalitions are only recorded if they are an improvement over the previous best candidate coalition. This metric provides a means of comparing algorithms by how fast they generate quality solutions. For complete algorithms (e.g., all four depth-first variations), the metric also measures how long it takes the algorithm to generate the optimal solution. The genetic algorithm is not a complete search and is therefore not guaranteed to find the optimal solution.

The specific missions to be completed, the number of UAVs available to form squadrons, and the ordering of those aircraft are important factors in determining search time, as they determine the number of iterations the algorithm must perform, as well as the location of the solutions within the search space. The algorithm must be run completely to ensure that either the optimal solution, or for genetic algorithms the best possible solution based on algorithmic limitations, has been found.

Two sets of two experiments each were devised in order to measure time to first solution and quality of solution over time. For time to first solution, the first experiment measured the time to first solution for four different missions for each of the five algorithms. A fixed size aircraft pool was used with random ordering. The second experiment measured the time to first solution using a set of increasing aircraft pool sizes with a fixed single mission. For each pool size, ten runs were performed with random aircraft ordering for each run. Averages for each of the pool sizes were determined after the runs completed.

Two experiments were devised in order to measure quality of solution over time. The first experiment measured the cost and elapsed time of each solution that costed less than the previous lowest-cost solution. A fixed single mission and random aircraft ordering were used. All four depth first-based algorithms were measured. The second experiment measured the cost and elapsed time of each solution which cost less than the previous lowest-cost solution using a genetic algorithm. Five runs were performed and results were recorded individually, since genetic algorithms can produce dissimilar results for identical inputs. A fixed single mission and random aircraft were determined beforehand, and used during each of the five runs.

For all experiments, the initial depth limit of the iterative deepening algorithms was set to three (3). This value was chosen since it seemed to represent the low end of the size range for most coalitions produced by the algorithms during development.

5.2 Coalition Formation, Time to First Solution

5.2.1 Results

Table 5.1 shows the average amount of time taken to produce the first viable coalition for four separate missions using five algorithms. Time shown is in seconds. The five algorithms tested were Depth-First (DF), Depth-First plus heuristics (DF+H), Depth-First Iterative Deepening (DFID), Depth-First Iterative Deepening plus heuristics (DFID+H), and a genetic algorithm (GA). The missions used were mission-bombrun, mission-recon, mission-delivery and mission-assault. These missions are described in detail in Appendix C.

Table 5.1: Algorithmic Time To First Solution (sec), by Mission

Algo.	Bombrun	Recon	Delivery	Assault
DF	0.01912087	1.06356981	0.00001117	0.00011985
DF+H	0.00146806	0.00024025	0.00000587	0.00240226
DFID	0.20350352	0.00003492	0.0000243	3.31529789
DFID+H	0.16809762	0.00003548	0.00000643	3.38535456
GA	0.01412163	0.01270189	0.03087711	0.00176698

Figure 5.1 plots the time taken to produce the first viable coalition for mission-recon using the five coalition-formation algorithms. Random UAVs were selected from the pool, with pool size beginning at ten and increasing to twenty-one. All five algorithms are displayed in the plot: Depth-First (DF), Depth-First plus heuristics (DF+H), Depth-First

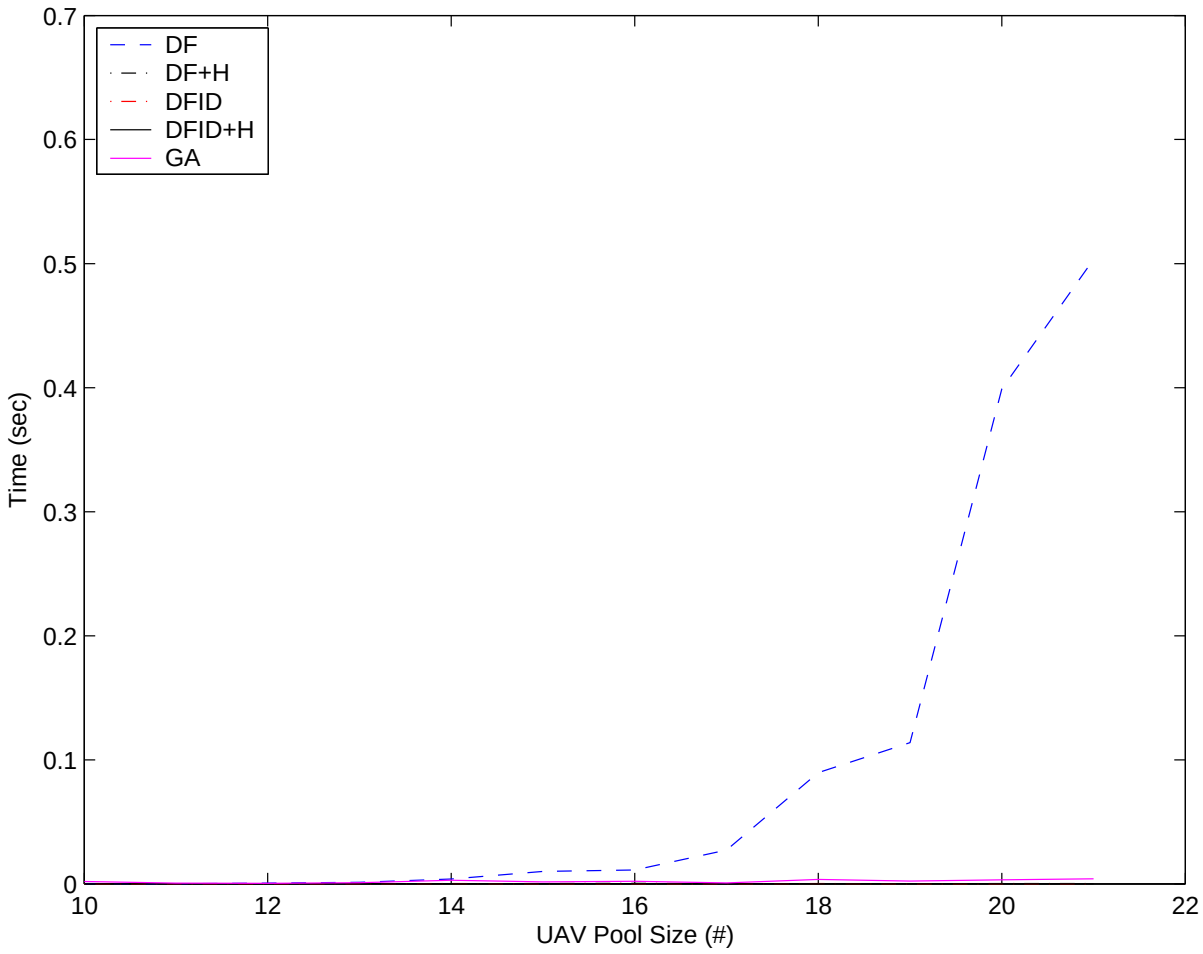

Figure 5.1: Coalition Formation, Time to First Solution vs Increasing Poolsize

Iterative Deepening (DFID), Depth-First Iterative Deepening plus heuristics (DFID+H), and genetic algorithm (GA).

Figure 5.2 is an additional plot displaying results for the same experiment as shown in figure 5.1. However, the Depth-First algorithm results are omitted because of differences in scale. The remaining algorithms appear in the plot: Depth-First plus heuristics (DF+H), Depth-First Iterative Deepening (DFID), Depth-First Iterative Deepening plus heuristics (DFID+H), and genetic algorithm (GA).

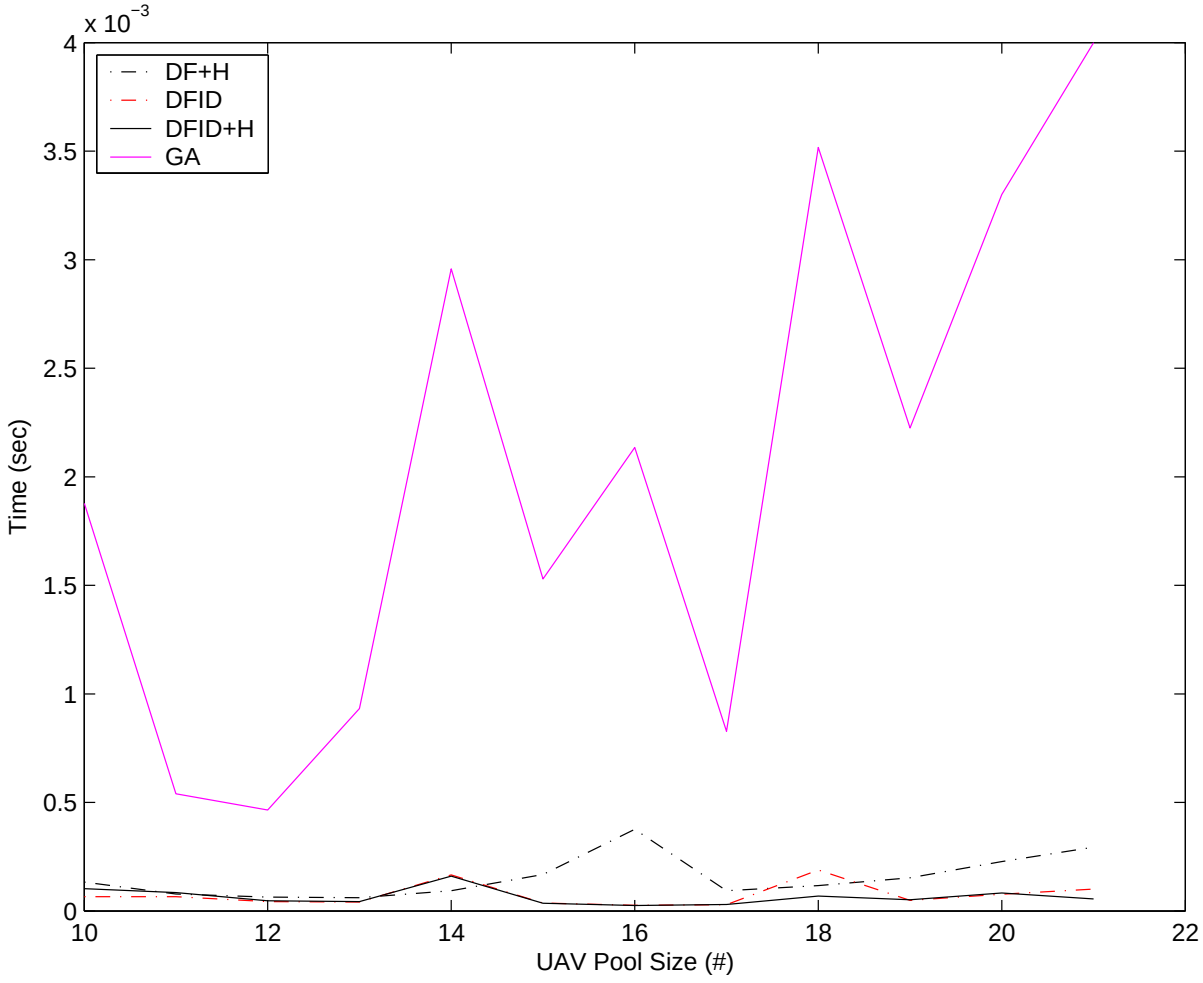

Figure 5.2: Coalition Formation, Time to First Solution vs Increasing Poolsize (DF, DF+H, GA)

5.2.2 Discussion

The results in table 5.1 clearly show the effect mission selection has on first solution time. For the assault mission, the DF algorithm found its first coalition in the shortest amount of time. Coalitions for this mission had to be quite large in order to meet mission require-

ments. This can possibly be explained by the nature of iterative deepening algorithms and the specifics of the missions being solved. Iterative deepening algorithms consider all possible coalitions up to a certain size before moving on the next maximum size. Since the mission required several UAVs, the iterative deepening algorithm was less efficient at reaching the coalition size where the first possible solution occurred.

For the recon and delivery missions, the iterative deepening algorithms (DFID and DFID+H) were found to generate first solutions the fastest. These two missions both required fairly small coalitions but with specific requirements. Since the iterative deepening algorithms consider smaller pool sizes first, and exhaust the possibilities at these sizes before continuing on to larger coalitions, solutions were found sooner than with the DF-based algorithms. The DF algorithm performed particularly poorly on the recon mission, meaning the solution coalition happened to occur toward the end of the search space.

The application of heuristics to DF and DFID algorithms sometimes caused slightly longer times, but occasionally improved times by several orders of magnitude. As explained in section 3.3.2, heuristics are only applicable for mission capabilities which are reducible to $min()$ and stealth (S) operations. The recon mission shows the DF+H algorithm operating several thousand times more quickly than the plain DF algorithm. The extra time taken by the heuristics in the other missions implies that few circumstances existed for applying heuristics, and the additional overhead resulting from evaluating each generated combination for heuristic applicability resulted in extra time.

In general, the results for the DF algorithm varied greatly. The DF algorithm considers all coalition sizes, and the shear number of large coalition possibilities means much time is spent considering these large coalitions. However, while some missions require large coalitions, many do not, and large coalitions carry a higher cost than small coalitions.

Therefore, without applying heuristics or iterative deepening, DF results are unpredictable and sometimes quite poor.

While the quality of the best solution found varied with the genetic algorithm, it was quite consistent in its first solution time. Genetic algorithms may be a good choice if the most important criteria for algorithmic selection is a bounded execution time, since most of the other algorithms had at least one type of mission which took a relatively long time to produce a first solution.

The second experiment showed the effect of pool size on algorithmic execution time. Figure 5.1 shows time to first solution for all five algorithms plotted against a UAV pool size ranging from ten to twenty-one. Pool size twenty-one, at the right side of the plot, shows similar results to those seen in table 5.1 for the recon mission, the mission that was used for the second experiment. The variations that are present can be attributed to the random UAV ordering within the aircraft pool. The variations in first solution time due to random UAV ordering are especially noticeable in the DF algorithm. The plot versus pool size also shows the disadvantage in first solution time of the DF algorithm as pool size increases. This is intuitive, as the DF algorithm suffers from the fastest explosion in search space size, since there are no heuristics to limit the search space.

Figure 5.1 shows the remaining four algorithms in closer detail. The DFID, DFID+H and DF+H algorithms were consistently faster in producing first solutions than the GA. The efficiency of the non-GA algorithms may be attributable to the specific mission for which coalitions were being generated. The recon mission contains both stealth (type S) and a velocity (type $min()$) components, so this mission was potentially able to benefit from heuristics. Additionally, this mission was solvable with small, lower cost coalitions, favoring the iterative deepening algorithms most of all. The GA algorithm exhibits random-appearing variation in first solution time, since first solution time sometimes rose

and sometimes fell as pool size increased. The variations are likely due to the stochastic nature of genetic algorithms.

5.3 Coalition Formation, Quality of Solution versus Time

5.3.1 Results

Quality of solution versus time was measured with two experiments. In the first experiment, the four depth first-based algorithms were run with a random ordering of all twenty-one available aircraft on the bombrun mission. Time and cost were recorded each time a lower cost solution was found than the previous least expensive solution. The results for all four algorithms are seen in figure 5.3. A more detailed view of the same data for the two iterative deepening algorithms is seen in figure 5.4.

The second experiment also used a random ordering of all twenty-one available aircraft as well as the bombrun mission. The genetic algorithm was run five times and each of the five results are shown in figure 5.5.

For both experiments, the * and + characters in the plots show instances in time where an improved coalition was discovered.

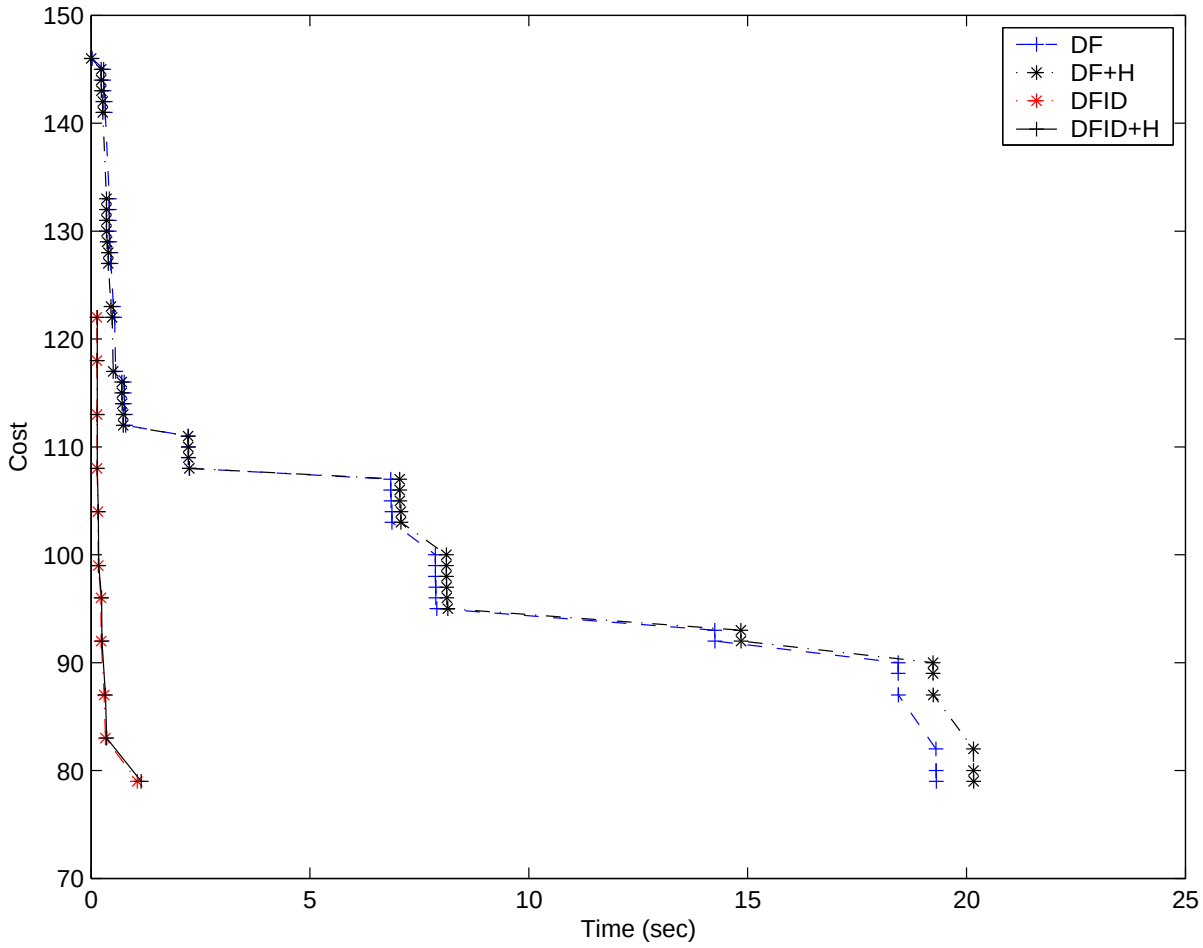

Figure 5.3: Coalition Formation, Quality of Solution vs Time (DF, DF+H, DFID, DFID+H)

5.3.2 Discussion

Each algorithm showed considerable improvement in solution quality over time, and the rate of improvement was highest at the beginning of the algorithm's execution. The DFID and DFID+H algorithms both found the optimal solution of cost=79 in about one second. The DF and DF+H algorithms both took considerably longer to find the optimal

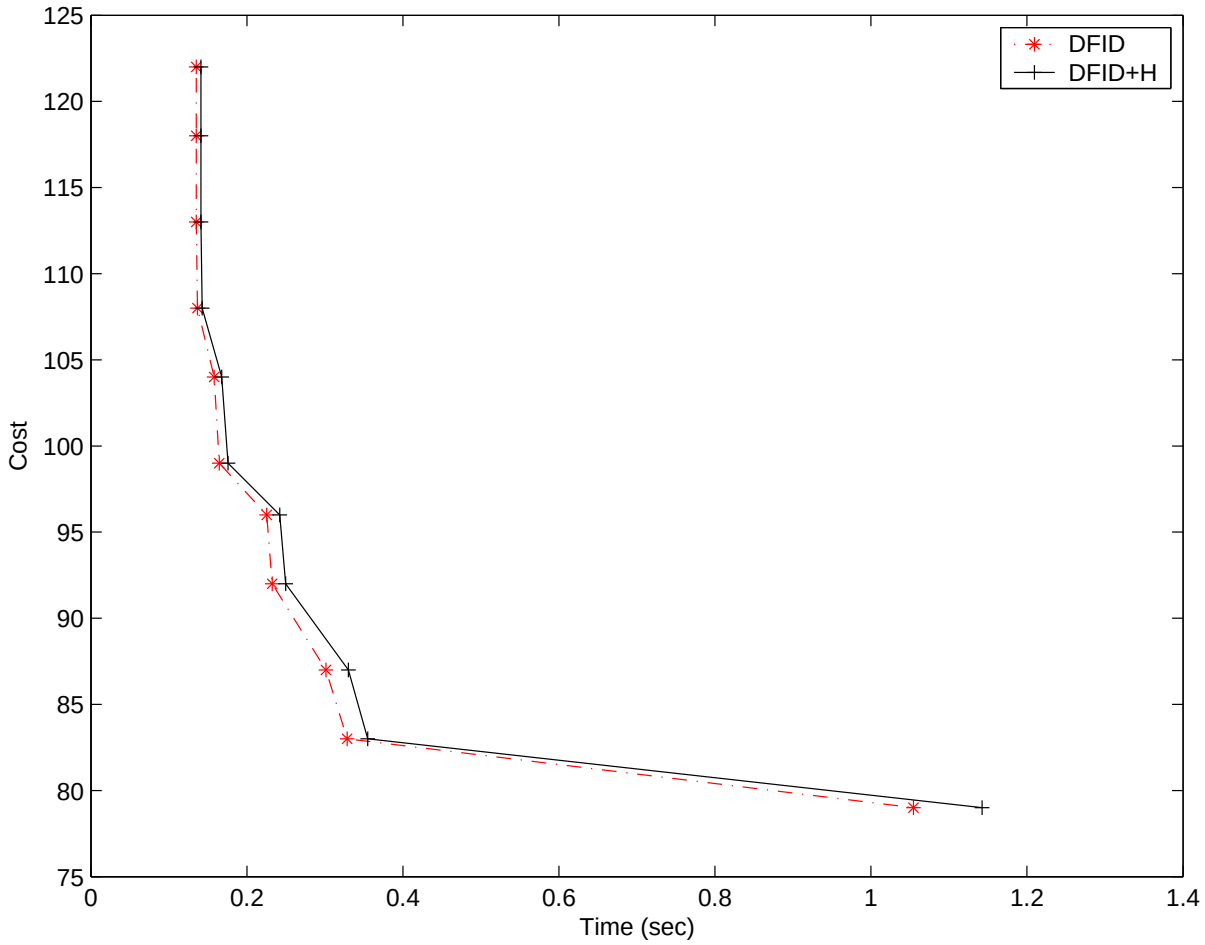

Figure 5.4: Coalition Formation, Quality of Solution vs Time (DFID, DFID+H)

solution, both at around twenty seconds. However, the results toward the beginning of each algorithm's execution were found to be quite similar for all four algorithms.

Noticeable for this experiment were how closely the DF and DF+H algorithms, as well as the DFID and DFID+H algorithms, paralleled each other in quality of solution and time found. In each case, the heuristic alternatives took slightly longer for each solution than their non-heuristic counterparts, but the same number of improved coalitions were found by each algorithm within each set of similar algorithms. This is a clear indication that

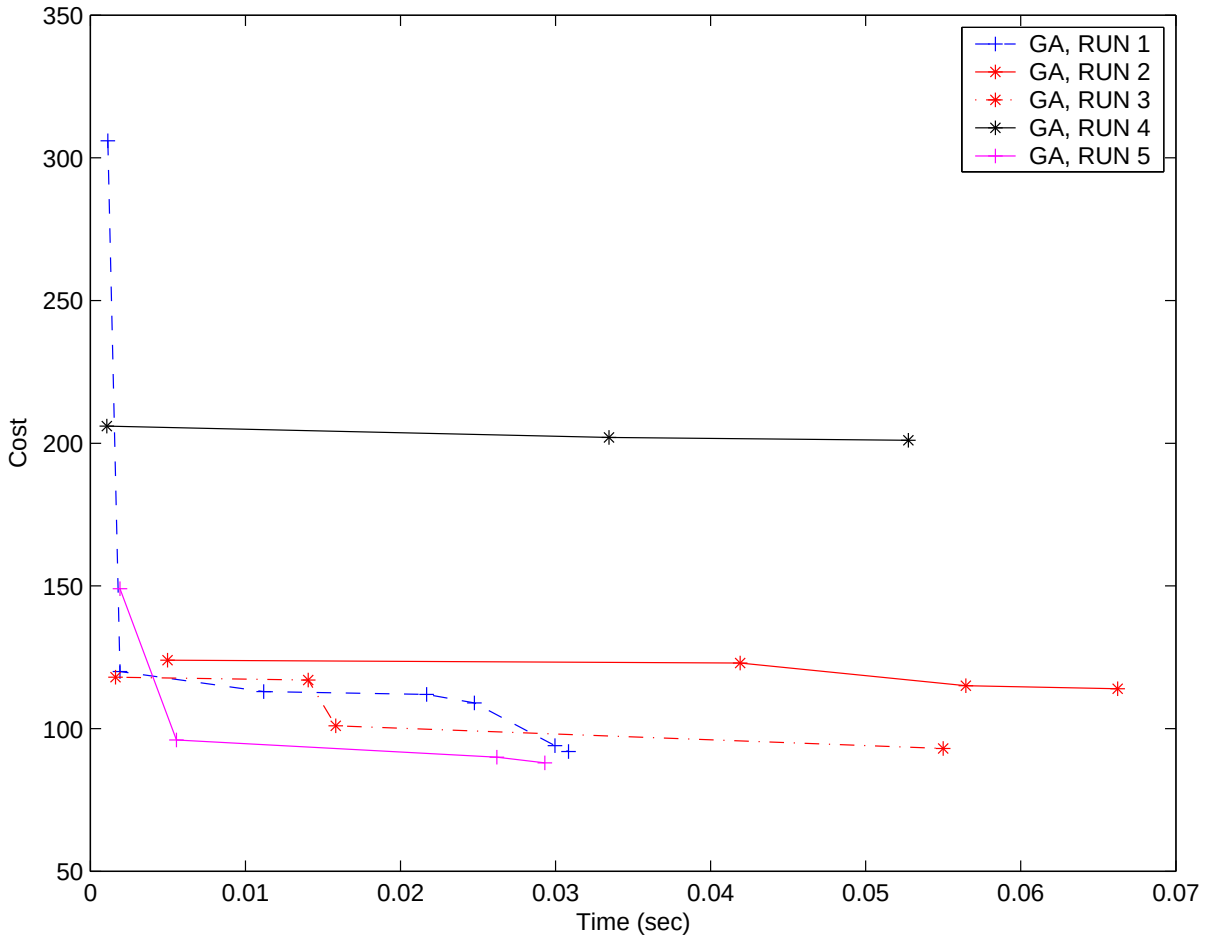

Figure 5.5: Coalition Formation, Quality of Solution vs Time (GA)

the mission was not heuristic-compatible, as can be seen by mission-bombrun capabilities listed in Appendix C. The slightly longer execution time for the heuristic algorithms is attributable to the overhead taken to determine if heuristics apply to each generated coalition.

The iterative deepening algorithms were found to discover the optimal solution in much less time than their non-iterative deepening counterparts. The optimal coalition was small and therefore was found more quickly by the iterative deepening algorithms. The solution

happened to be located later in the search space for the depth-first algorithms. The selection of the mission plays an important role in the efficiency of depth-first algorithms in generating solutions. This is particularly noticeable with the optimal solution.

Figure 5.5 demonstrates the wide variety of results produced by genetic algorithms in coalition formation. The best solution found was cost=88 by Run 5. This solution was nine points greater than the optimal solution of cost=79 discovered by all of the depth first-based searches.

Also evident are the differences in time taken to find the best solution produced by the run. Run 5 produced the best solution, at cost=88, at about .03 seconds. However, Run 4 only managed to produce a best result of cost=201, and it took over .05 seconds to find this solution, almost twice as long as the much better Run 5 result. In all, genetic algorithms tend to produce results within a predictable window of time, but the quality of results can vary widely. Perhaps most importantly, the genetic algorithm employed in coalition formation always failed to find the optimal solution, regardless of the number of runs or make-up of the aircraft and mission pools.

5.4 Effect of Squadron Formation and UAV Count on Social Potential Field Force

5.4.1 Description

Social potential fields were employed to prevent collision between UAVs. During the course of a mission, the lead UAV makes several changes in heading. When this occurs,

the desired positions of the slots of the formation UAVs change. In attempting to reach the new positions, the paths of UAVs following the leader may nearly cross, raising the social potential field force between the UAVs and forcing collision avoidance strategies to be employed. It is also possible that the positional tolerances built into the system allow UAVs to "drift" close together. Such an event causes an increase in the social potential field repulsive force, possibly enough to cross a tolerance threshold and trigger collision avoidance behavior. Changing the threshold itself, or changing the distance between UAVs in formation, can reduce the need for collision avoidance.

These experiments attempted to measure the affect of two variables on the repulsive force of social potential fields: Formation size and distance maintained between UAVs in formation. A simple mission was devised which contained a single objective. The objective was located at the area of the map furthest from the starting location, causing the UAVs to traverse the map in a relatively direct path. It was hoped that this direct flight would reduce the chance of incidental social potential field threshold crossing since UAVs are more likely to cross paths during turns, and therefore provide more accurate results.

The variables in this set of experiments were number of UAVs, Formation Distance (shortest distance between two ranks in a formation), and social potential field threshold (the minimum force required to change a UAV into collision avoidance state). The outputs measured were total time spent in social potential field avoidance for all UAVs, average time spent in social potential field avoidance per UAV, and maximum social potential field magnitude recorded during the run.

Table 5.2: Observed SPF Time, SPF Time per UAV, and Max SPF Force

UAVs	Distance	SPF Threshold	Mission (s)	SPF (s)	SPF/UAV	Max SPF
4	20	7.5	0	0	0	0
4	20	10	414	258	64.5	277.79
4	20	12.5	282	117	29.25	13.78
4	20	15	311	67	16.75	15.19
4	30	7.5	534	407	101.75	39.15
4	30	10	293	153	38.25	13.35
4	30	12.5	369	162	40.5	14.61
4	30	15	341	10	2.5	15.27
3	20	7.5	281	180	60	9.17
3	20	10	322	73	24.33	11.69
3	20	12.5	321	6	2	12.53
3	20	15	323	0	0	12.76
3	30	7.5	288	178	59.33	8.88
3	30	10	327	89	29.67	10.46
3	30	12.5	332	0	0	10.66
3	30	15	324	0	0	10.94

5.4.2 Results

Table 5.6 measures total collision avoidance time for varying number of UAVs and formation distance value (distance between UAV and PB in figure 4.4). Note that the results for UAVs=4 and Distance=20 show a collision avoidance time of 0. The mission was

never completed during this run because of excessive time spent in social potential field collision avoidance on the part of the leader, preventing navigation toward the target.

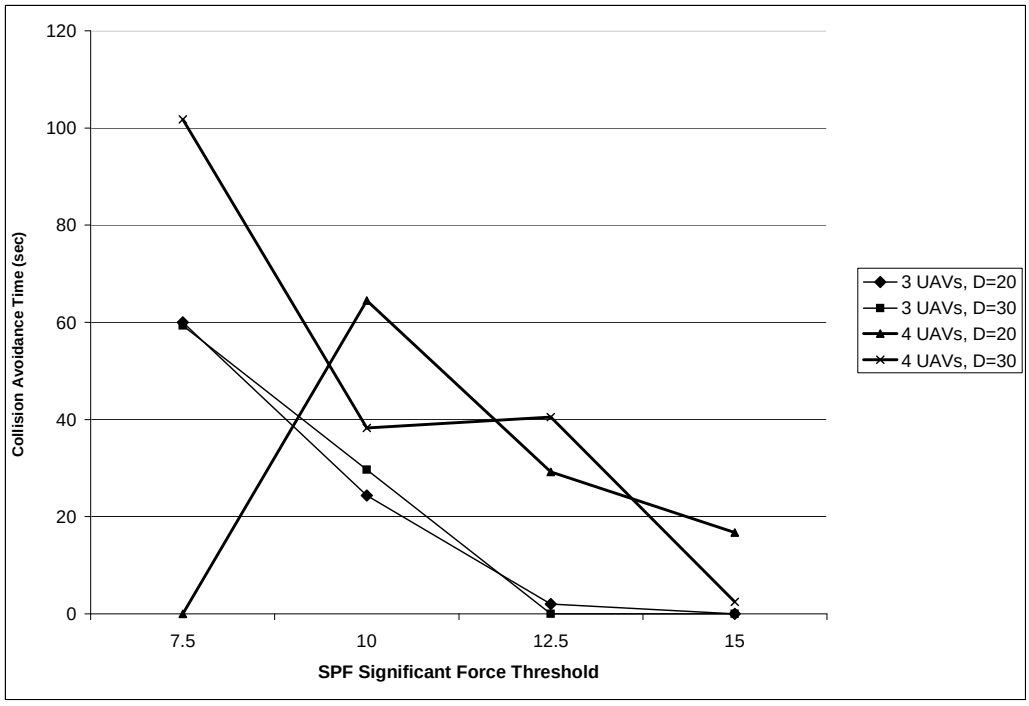

Figure 5.6: Social Potential Field Collision Avoidance Time

5.4.3 Discussion

As expected, increasing the social potential field threshold reduced the amount of time spent in social potential field collision avoidance. For a given event of one UAV passing close to another and triggering SPF collision avoidance, an increased threshold would often filter out an identical event, avoiding collision avoidance behavior altogether (for example, two UAVs flying close together but not on a collision course). Collision avoidance behavior was rarely avoided altogether, since events such as two vehicles flying directly toward one another would cause collision avoidance regardless of the threshold value.

The results show a clear correlation between the number of UAVs assigned to the formation and the amount of time spent reacting to social potential field force (table 5.2). Figure 5.6 shows 20-40 seconds additional SPF avoidance time spent by the runs involving four UAVs (the thicker pair of lines in the chart) versus the runs involving three UAVs (the thinner pair of lines).

However, the distance maintained between UAVs in formation do not appear to substantially affect time spent reacting to social potential field force. This result was somewhat surprising, since the triggering event for social potential field collision avoidance is close proximity. In practice, one UAV frequently trails the formation during startup, before the formation is fully formed. While attempting the catch up to the formation, the UAV chooses the shortest path between itself and its assigned formation slot. This path often makes the UAV pass close to other UAVs which may correctly be in their formation slots and therefore are flying more slowly. Since these slot recovery paths brought UAVs within twenty coordinate units, SPF collision avoidance was triggered. Once UAVs lock in on their formation slots, crossing the SPF threshold is less common.

CHAPTER 6

CONCLUSIONS

This work explored the formation of coalitions in multi-agent UAV systems. Classical coalitions as presented by Shehory are formed on the basis of additive capabilities; i.e., like capabilities belonging to multiple agents are simply added together when forming coalitions of agents. This work presented a new capability operator, $\oplus$, and showed that the behavior of this operator is context-sensitive, depending on the nature of the capability being considered. A collection of realistic capabilities were devised and applied to UAVs. The effect of applying the new operator to UAVs holding each of the new capabilities was determined.

Realistic missions were devised which required some or all of the UAV capabilities. A pool of UAVs were devised which held a certain level of proficiency in some or all of the capabilities. Coalitions of UAVs, effectively squadrons, were formed which contained the correct distribution of capabilities to meet the mission requirements of the missions. Coalitions were formed using one of three algorithms: Depth-First (DF), Depth-First Iterative Deepening (DFID), and Genetic Algorithm (GA). Heuristics were devised which were capable of "short-circuiting" algorithmic computation for both the DF and DFID algorithms. The heuristics were found to be beneficial at reducing algorithmic execution time for capabilities which had $min()$ or S (stealth) behavior.

Several other issues associated with UAV systems were explored. These include simulation using multi-agent systems, agent identity, mission management, agent split/merge/migrate behavior, formation flight, and the use of social potential fields for aircraft collision avoidance. A software simulation was created in order to test the coalition formation and other ideas. The software system made use of an existing 3-D graphical simulator, as well as the YAES simulation framework. The software is a complex system modeling each of the UAV-related issues described above.

Several experiments were devised. The experiments demonstrated that none of the applied algorithms were inherently superior to the others, and situations existed in which each algorithm was found to generate solutions more quickly than the others. The first experiment measured the time to first solution for each of the three algorithms, both with and without the application of heuristics. It was discovered that non-complete algorithmic solutions, such as GA, provide a more predictable time to first solution than complete algorithms. However, variations in the nature of the mission and ordering of UAVs within the available pool provided situations where each of the different algorithms produced the fastest time to first solution.

A second experiment incrementally increased the number of UAVs available in the UAV pool to see the effect on time to first solution and total execution time. It was discovered that the DF algorithm should only be used for relatively small UAV pool size due to the huge explosion in search space associated with this algorithm. Heuristics often increased solution time slightly due to the overhead of constantly determining heuristic applicability. However, when heuristics were compatible with mission capability requirements, they provided several orders of magnitude improvement in first solution time.

The third and fourth experiments measured the rate of increase in quality of solution over time as the algorithm progressed. The DFID algorithms, both with and without

heuristics, were found to produce better results faster than the the DF-based algorithms. The optimal solution was found twenty times faster by the DFID-based algorithms than their DF-based counterparts. The GA algorithm was quite efficient in improving quality over time, but produced variable quality best solution results. The GA algorithm failed to find the optimal solution in all cases.

A final experiment explored the effects of varying social potential field (SPF) force threshold and distance between UAVs in formation on total time spent in "SPF Hold" during a short mission. It was discovered that fine-tuning these variables can have a great effect on SPF behavior and therefore the speed and efficiency of mission execution.

CHAPTER 7

FUTURE WORK

This work introduced a new capability operator used when combining UAVs during coalition formation. Only four operator behaviors (+, $min()$, $max()$, and $S()$), and five UAV capabilities were examined while exploringing the behavior of the new capability operator. Certainly more operator behaviors than these four exist, and countless more capabilities which are applications of these operator behaviors remain to be identified.

Three algorithms were presented for forming coalitions, two of which (Depth-First and Depth-First Iterative Deepening) are closely related. More complicated, diverse and intelligently applied algorithms may generate coalition combinations more efficiently and therefore result in reduced search times. Identifying and testing these algorithms could offer both faster first solution time as well as total execution time. The size of the pool of available UAVs could then be increased without causing unacceptably long computation times.

A software enhancement could allow split and merge operations to be triggered by the system without input from the user. Work required to meet mission objectives could be continually evaluated. When changing the composition of the squadron could benefit mission success or completion time, the make-up of the squadron could be altered accordingly. "Shortest path" algorithms could be employed in order to evaluate the best make-up of a squadron to meet mission objectives in the shortest amount of time.

The present system is made less realistic by its use of OpenGL coordinates for measurements. A simple layer of software between the client system and simulation could provide a translation layer which maps OpenGL coordinates into meters or other standard unit of measurement. The system also currently operates as fast as possible, with certain decision made based on the number of iterations though this loop. A better and more realistic approach would make decisions based on time. This layer of software could also map loop iterations to real time. Both of these improvements would make the system more realistic and allow for more predictability and repeatability.

Missions are currently limited to navigation, single-point loiter, lawnmower loiter, and bombing. Adding more complicated missions types such as reconnaissance, efficient algorithm-based searching, terrain mapping, etc. would enhance the usefulness of the client software as a testbed for investigating social UAV behavior. Similarly, the number of states a UAV can enter is currently quite limited, as is evident from table 4.1. An increased number of states would become necessary as the number and types of missions increased.

APPENDIX A

MESSAGE SET

This section describes the messages used for UAV-to-UAV, UAV-to-base and base-to-UAV communications in the simulation software. All messages include a sender agent name and a receiver agent name as parameters. Additional parameters are message-dependent and are listed below.

A.1 New Mission

The New Mission message assigns a new set of objectives to the receiver. This message is sent as the result of a split or merge operation.

Additional Parameters: mission, the new Mission being assigned to a UAV

A.2 Request Mission

The Request Mission message is sent when servicing a split or merge command. One UAV leads the split or merge procedure by gathering all Missions belonging to UAVs involved in the operation, prior to performing the actual split or merge.

Additional Parameters: none

A.3 Mission

The Mission message sends a copy of a UAV's current Mission to the requester. This message is sent in response to a Request Mission message.

Additional Parameters: mission, the UAV's currently assigned Mission objectiveIndex, indicating which mission objective is currently being executed by the UAV

A.4 New Squadron

The New Squadron message assigns a UAV to a squadron. Upon receipt, the recipient will quit their current squadron and seek a leader in the newly assigned squadron. If no leader is available, the UAV initiates a leader negotiation. The New Squadron message is sent as the result of a split or merge operation.

Additional Parameters: squadron, a unique identifier for the squadron following the naming rules of agent identity members, a String array of names of UAVs currently assigned to the squadron

A.5 Get Current Objective

Sent as a result of a user command, the Get Current Objective message requests that the recipient send detailed information on the mission objective currently being executed. Once received, the objective data will be displayed to the user console.

Additional Parameters: none

A.6 Current Objective

The current objective message sends a copy of the current objective data structure to the requester, which will be displayed on the user console. The Current Objective message is sent in response to the Get Current Objective message.

Additional Parameters: objective, the current mission objective being executed by the sender

A.7 Get Objectives

The Get Objectives message requests that a readable listing of all objectives within the mission currently being executed by the recipient be returned to the center. This message is sent as the result of a user command, when a textual display of the mission objectives to the console is desired.

Additional Parameters: none

A.8 Objectives

The Objectives message sends a textual display of the mission objectives currently being executed by the sender. The objectives are displayed on the user console once received by the headquarters agent.

Additional Parameters: objectivesDescription, a String describing the UAV's current mission objectives

A.9 Leader Needed

The Leader Needed message is sent by a UAV to all other UAVs in the squadron when the UAV detects that their present squadron has no leader. This is the first step in leader contract net negotiation.

Additional Parameters: none

A.10 Propose Lead

Sent as a the result of a Leader Needed message, the Propose Lead message is sent by a UAV volunteering for squadron leadership. This message is part of the leader contract net negotiation.

Additional Parameters: pos, the current position of the sender uavType, this UAV's type: "normal," "bomber," or "scout"

A.11 Assign Leader

The Assign Leader message is sent by a UAV acting in the initiator role in a leader contract net negotiation. This message indicates that the recipient is assigned leadership of the squadron, as the recipient is the most appropriate UAV from among those proposing leadership to lead the squadron. The leader always assumes the "point" position in the squadron, with other UAVs falling in formation behind the leader.

Additional Parameters: none

A.12 Assign Formation

The Assign Leader message is sent by a UAV acting in the initiator role in a leader contract net negotiation. This message is sent to all UAVs in the squadron except the one assigned leader, indicating the recipients are assigned to a "slot" in the formation behind the leader.

Additional Parameters: leader, the name of the UAV which has been assigned leader row, the row number behind the leader the recipient should assume in the formation side, the side relative to center of the leader which the recipient should assume in the row. Either "left" or "right."

A.13 Bomber Needed

The Bomber Needed message is sent by UAV which detects a bomb target, initiating a bomber contract net negotiation. The message is sent to every UAV in the initiator's squadron.

Additional Parameters: none

A.14 Propose Bomb

The Propose Bomb message is sent as the result of a bomber contract net negotiation. This message is sent by a UAV of type "bomber," volunteering to bomb the target.

Additional Parameters: pos, the current position of the sender

A.15 Decline Bomb

The Propose Bomb message is sent as the result of a bomber contract net negotiation. This message indicates that the sender is unable or unwilling to bomb a target.

Additional Parameters: none

A.16 Assign Bomb

The Assign Bomb message is sent by a UAV acting in the initiator role in a bomber contract net negotiation. This message indicates that the recipient is assigned the bomber of a bomb target, as the recipient is the most appropriate UAV from among those volunteering to bomb a target.

Additional Parameters: pos, position of the target to bomb

A.17 Assign Wait

The Assign Bomb message is sent by a UAV acting in the initiator role in a bomber contract net negotiation. The Assign Wait message indicates that the recipient should loiter in place while another UAV carries out a bomb run. This message is sent to all squadron UAVs other than the UAV selected as the bomber.

Additional Parameters: none

A.18 Resume Mission

The Resume Mission message is sent by the bomber upon completion of a bomb run, indicating that recipients should switch from a loiter state to carrying out the current mission objective. The Resume Mission message is sent to all squadron members.

Additional Parameters: none

A.19 Position

The Position message sends positional information about the sender to a recipient UAV. This message is used in many circumstances, including a squadron leader updating its wingmen with its current location so they can follow in formation.

Additional Parameters: Pos, the sender UAV's current positional information: x,y,z coordinates; plus optionally heading, velocity, and altitude

A.20 Objective Number

The Objective Number message sends a number representing the mission objective currently being executed to a recipient. This message is sent by a leader to squadron mates when a mission objective has been completed and the leader is transitioning to the next objective. In this way, all UAVs in the squadron stay synchronized in case a new leader is required due to a split/merge operation or some other reason.

Additional Parameters: objectiveIndex, a number which represents the mission objective currently being executed by the squadron

APPENDIX B

USER COMMAND SET

B.1 New Mission Command

The new message command instructs a squadron to load a new mission from secondary storage. The squadron handle is a shortcut for specifying the a squadron rather than typing the entire extended squadron name. The squadron handle for each squadron can be obtained with the "who" command.

Format: cmd-new-mission [mission_file] [squadron_handle]

B.2 Merge Mission Command

The merge mission command instructs a squadron to merge a mission from secondary storage into the current mission. Objectives will be re-ordered according to objective priority.

Format: cmd-merge-mission [mission_file] [squadron_handle]

B.3 Merge Squadron Command

The merge squadron command instructs two squadrons to merge into one. The new squadron will be assigned a new squadron name according to the extended name naming conventions. Objectives will be re-ordered according to objective priority.

Format: cmd-merge [squadron_handle_1] [squadron_handle_2]

B.4 Split Squadron Command

The split squadron command instructs a squadron to split into two squadrons. The new squadrons will be assigned a new names according to the extended name naming conventions. Objectives in each of the two squadrons will be re-ordered according to objective priority.

Format: cmd-split [squadron_handle]

B.5 Get Current Objective Command

The get current objective command requests a string representation of the current objective from a squadron. The response will be displayed on the user console.

Format: cmd-get-current-objective [squadron_handle]

B.6 Get Objectives List Command

The get objectives list command requests a string list of all objectives in the current mission from a squadron. The response will be displayed on the user console.

Format: cmd-get-objectives [squadron_handle]

B.7 Who Command

The who command requests a list of all squadrons, a handle for each squadron, and the names of all UAVs in each squadron. The results are displayed on the user console.

Format: cmd-who

B.8 Quit Command

The quit command closes the current session and exits the user console.

Format: quit

B.9 Test Command

The test command displays sample text to the user console.

Format: test

B.10 Form Coalition Command

The form coalition command instructs the recipients to form a coalition to complete the objectives in coalition_mission. The selected coalition forms a squadron which is assigned

to the coalition_mission. The UAVs which are not selected form a second squadron and are assigned to reject_mission.

Format: cmd-form-coalition [coalition_mission] [reject_mission] [squadron 1] ... [squadron n]

APPENDIX C

MISSION POOL

C.1 Mission Pool

The pool of missions was established in order to provide an assortment of missions on which experiments could draw. Each mission is based on a plausible scenario and an attempt was made to assign capability requirements in keeping with this description.

Table C.1: Mission Pool

Name	Firepower	Num Bombs	Visibility	Max Velocity	Stealth
mission-bombrun	200.0	5	50.0	2.0	15.0
mission-recon	35.0	0	75.0	2.5	75.0
mission-delivery	10.0	0	0.0	4.0	0.0
mission-assault	300.0	7	25	1.0	0
mission-impossible	500.0	50	100.0	10.0	100.0

C.1.1 mission-bombrun

Description: Advance toward a suspected bombing target. Enemy fighters may be in the area.

C.1.2 mission-recon

Description: Perform reconnaissance in a mountainous, uncharted area. Attempt to remain unspotted. Carry only enough firepower as is necessary for self defense if spotted.

C.1.3 mission-delivery

Description: Deliver urgent supplies to a remote location. Speed of delivery is the most important factor.

C.1.4 mission-assault

Description: Perform an all-out unmanned assault on a known target which is like to be heavily guarded.

C.1.5 mission-impossible

Description: A mission which cannot be carried out with a reasonable squadron. Provides a worst-case search scenario, since any non-genetic algorithm will iterate though all coalition possibilities without stopping for a successful coalition.

APPENDIX D

AIRCRAFT POOL

D.1 Aircraft Pool

The pool of aircraft was designed to distribute an assortment of capabilities across UAVs. Categories of UAVs were created to reflect various genres of UAVs in operation today, and capabilities were assigned to best estimate UAV abilities relative to other types of UAVs.

D.1.1 large bomber

Description: Contains large payload bombs and very little defense. Large, clumsy and slow. Usually escorted by fighters.

D.1.2 large fighter

Description: This UAV is equipped with air-to-air munitions including missiles and machine gun. It has a stealthy design, and contains radar-jamming devices, further lowering its stealth rating. However, it is also quite large.

D.1.3 large recon

Description: This UAV is large and fast, but equipped only with sensors and no munitions.

Table D.1: Aircraft Pool

Name	Firepower	Num Bombs	Visibility	Max Velocity	Stealth
large bomber 1	10.0	4	10.0	2.5	15.0
large bomber 2	10.0	4	10.0	2.5	15.0
large bomber 3 v	5.0	4	10.0	2.5	25.0
large fighter 1	80.0	0	20.0	3.5	45.0
large fighter 2	80.0	0	20.0	3.5	45.0
large fighter 3 v	90.0	0	20.0	3.5	35.0
large recon 1	0.0	0	65.0	3.7	75.0
medium bomber 1	15.0	2	15.0	2.7	20.0
medium bomber 2	15.0	2	15.0	2.7	20.0
medium bomber 3	15.0	2	15.0	2.7	20.0
medium fighter 1	55.0	0	15.0	3.0	60.0
medium fighter 2	55.0	0	15.0	3.0	60.0
medium fighter 3	55.0	0	15.0	3.0	60.0
medium fighter 4 v	50.0	0	15.0	3.3	60.0
medium fighter 5 v	50.0	0	15.0	3.3	60.0
small recon 1	0.0	0	55.0	3.4	95.0
small recon 2	0.0	0	55.0	3.4	95.0
small recon 3	0.0	0	55.0	3.4	95.0
small recon 4 v	0.0	0	75.0	2.9	90.0
small recon 5 v	0.0	0	75.0	2.9	90.0
small recon 6 v	0.0	0	75.0	3.4	60.0

D.1.4 medium bomber

Description: This UAV uses its payload for air-to-ground missiles or smart-bombs.

D.1.5 medium fighter

Description: This UAV is primarily intended for use escorting bombers or other UAVs or fighting other aircraft. It is agile and stealthy.

D.1.6 small recon

Description: This UAV has an extremely small footprint and contains a camera and infrared sensor. However, its range is quite limited.

LIST OF REFERENCES

[AGL02] "Aglets." URL http://www.trl.ibm.com/aglets/, 2002.

[B03] Ladislau Bölöni. "From the Philosopy of Personal Identity to the Laws of Agent Societies.", 2003.

[BDT04] Ladislau Bölöni, Paul DeJong, and Damla Turgut. "Agents with non-anthropomorphic lifecycles." In *Intelligent Agent Architectures: Combining the Strengths of Software Engineering and Cognitive Systems: Papers from the 2004 AAAI Workshop*. AAAI Press, July 2004.

[Bec02] Steffi Beckhaus. *Dynamic Potential Fields for Guided Exploration in Virtual Environments*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2002.

[BKB04] Ladislau B̈olöni, Majid Ali Khan, Xin Bai, Guoqiang Wang, Yongchang Ji, and Dan C Marinescu. "Software engineering challenges for mutable agent systems." In Carlos Lucena, Alessandro Garcia, Alexander Romanovsky, Jaelson Castro, and Paulo Alencar, editors, *Advances in Software Engineering for Multi-Agent Systems*. Springer Verlag, 2004.

[BM04] Ladislau Bölöni and Dan C Marinescu. "Adaptation and Mutation in Multi-Agent Systems and Beyond." In Laksmi Jain, editor, *Learning, Coordination and Communication in Multi-Agent Systems*. World Scientific, 2004.

[BON02] "BOND: A Multi-Agent System." URL http://bond.cs.ucf.edu/, 2002.

[Cha03] L.E. Champagne. "Bay of Biscay: Extensions into Modern Military Issues." In *Simulation Conference, 2003. Proceedings of the 2003 Winter , Volume: 1*, pp. 1004–1012, 2003.

[CS03] Amit K. Chopra and Munindar P. Singh. "Nonmonotonic Commitment Machines." In *Proceedings of the International Workshop on Agent Communication Languages and Conversation Policies (ACL)*, Melbourne, Austrialia, July 2003. Springer (in press).

[DAG02] "D'Agents: Mobile Agents at Dartmouth College." URL http://agent.cs.dartmouth.edu/, 2002.

[DWK01] Dwight Deugo, Michael Weiss, and Elizabeth Kendall. "Reusable Patterns for Agent Coordination." In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents*. Springer, 2001.

[FIP02] "FIPA Contract Net Interaction Protocol Specification." URL http://www.fipa.org/specs/fipa00029/SC00029H.pdf, 2002.

[FIP05] "FIPA Agent Communication Language Specification." URL http://www.fipa.org/repository/aclspecs.php3, 2005.

[GHK04] Erol Gelenbe, Khaled Hussain, and Verol Kaptan. "Simulating Autonomous Agents in Augmented Reality." *Accepted for publication, J. Software and Systems*, 2004.

[GN69] RS Garfinkel and GL Nemhauser. "The set partitioning problem: set covering with equality constraints." *Operations Research*, **17**:848–856, 1969.

[HMI03] M. Hyodo, Y. Matsuo, and T. Ito. "An optimal coalition formation algorithm for electronic group buying." In *SICE 2003 Annual Conference*, volume 27, pp. 3402–3407, 2003.

[Hol75] J.H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[JAD05] "Java Agent DEvelopment Framework." URL http://jade.tilab.com/, 2005.

[Kor95] Richard E. Korf. "Space-efficient search algorithms." *ACM Comput. Surv.*, **27**(3):337–339, 1995.

[KQM93] "KQML Specification Document." URL http://www.cs.umbc.edu/kqml/kqmlspec.ps, 1993.

[KST03] Sarit Kraus, Onn Shehory, and Gilad Taase. "Coalition formation with uncertain heterogeneous information." In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 1–8. ACM Press, 2003.

[Luo04] Linus Jan Luotsinen. *"Autonomous Environmental Mapping in Multi-Agent UAV Systems."*. Master's thesis, University of Central Florida, 2004.

[Pap01] George A. Papadopoulos. "Models and Technologies for the Coordination of Internet Agents: A Survey." In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents*. Springer, 2001.

[Pas97] Vangelis T. Paschos. "A survey of approximately optimal solutions to some covering and packing problems." *ACM Comput. Surv.*, **29**(2):171–209, 1997.

[RW99] John H. Reif and Hongyan Wang. "Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots." *Robotics and Autonomous Systems*, **27**:171–194, 1999.

[SD00] Sandip Sen and Partha Sarathi Dutta. "Searching for optimal coalition structures." In *Proceedings of the Fourth International Conference on Multiagent Systems*, pp. 286–292. IEEE, 2000.

[SK95] Onn Shehory and Sarit Kraus. "Task Allocation via Coalition Formation among Autonomous Agents." In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 655–661, Montréal, Québec, Canada, 1995.

[Smi80] R.H. Smith. "The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver." *IEEE Transactions on Computers*, 1980.

[SML04] Ted Scully, Michael G. Madden, and Gerard Lyons. "Coalition calculation in a dynamic agent environment." In *ICML '04: Twenty-first international conference on Machine learning*. ACM Press, 2004.

[SSJ97] Onn Shehory, S. K. Sycara, and Somesh Jha. "Multi-agent Coordination through Coalition Formation." In *Intelligent Agents IV: Agent Theories, Architectures and Languages, Lecture Notes in Artificial Intelligence*, number 1365, pp. 143–154. Springer, 1997.

[ST85] M. E. Stickel and W. M. Tyson. "An analysis of consecutively bounded depth-first search with applications in automated deduction." In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-85*, pp. 1073–1075, 1985.

[XS01] Jie Xing and Munindar P. Singh. "Formalization of Commitment-Based Agent Interaction." In *Proceedings of the 2001 ACM Symposium on Applied Computing*. ACM Press, 2001.

[YAE05] "YAES: Yet Another Extensible Simulator." URL http://netmoc.cpe.ucf.edu/Yaes/Yaes.html, 2005.